SIEMENS

SIMATIC

SCL pour S7-300/400
Programmation de blocs

Manuel

Ce manuel a le numéro de référence suivant :

6ES7811-1CA02-8CA0

Informations relatives à la sécurité

Ce manuel donne des consignes que vous devez respecter pour votre propre sécurité ainsi que pour éviter des dommages matériels. Elles sont mises en évidence par un triangle d'avertissement et sont présentées, selon le risque encouru, de la façon suivante :

Danger

signifie que la non-application des mesures de sécurité appropriées **conduit** à la mort, à des lésions corporelles graves ou à un dommage matériel important.

Attention

signifie que la non-application des mesures de sécurité appropriées **peut conduire** à la mort, à des lésions corporelles graves ou à un dommage matériel important.

Avertissement

signifie que la non-application des mesures de sécurité appropriées peut conduire à des lésions corporelles légères ou à un dommage matériel.

Nota

doit vous rendre tout particulièrement attentif à des informations importantes sur le produit, aux manipulations à effectuer avec le produit ou à la partie de la documentation correspondante.

Personnel qualifié

La mise en service et l'utilisation de la console ne doivent être effectuées que conformément au manuel.

Seules des **personnes qualifiées** sont autorisées à effectuer des interventions sur la console. Il s'agit de personnes qui ont l'autorisation de mettre en service, de mettre à la terre et de repérer des appareils, systèmes et circuits électriques conformément aux règles de sécurité en vigueur.

Utilisation conforme aux dispositions

Tenez compte des points suivants :

Attention

Le produit ne doit être utilisé que pour les applications spécifiées dans le catalogue ou dans la description technique, et exclusivement avec des périphériques et composants recommandés par Siemens.

Marques

SIMATIC®, SIMATIC NET® et SMATIC HMI® sont des marques déposées de SIEMENS AG.
Les autres désignations dans cette publication peuvent être des marques dont l'utilisation par des tiers pour leur compte peut léser les droits des propriétaires.

Siemens AG
Bereich Automatisierungs- und Antriebstechnik
Geschaeftsgebiet Industrie-Automatisierungssysteme
Postfach 4848, D-90327 Nuernberg

Exclusion de responsabilité

Nous avons vérifié la conformité du contenu du présent manuel avec le matériel et le logiciel qui y sont décrits. Or des divergences n'étant pas exclues, nous ne pouvons pas nous porter garants pour la conformité intégrale. Si l'usage de ce manuel devait révéler des erreurs, nous en tiendrons compte et apporterons les corrections nécessaires dès la prochaine édition. Veuillez nous faire part de vos suggestions.

Avant-propos

Objet du manuel

Ce manuel constitue une aide précieuse lorsque vous écrivez vos programmes utilisateur dans le langage de programmation SCL. Il traite des procédures de principe concernant la création de programmes avec l'éditeur, le compilateur et le débogueur de SCL.

Ce manuel comporte en outre une partie de référence traitant des éléments de langage de SCL, en particulier de leur syntaxe et de leur fonction.

Groupe cible

Le présent manuel s'adresse aux programmeurs de programmes S7, ainsi qu'au personnel de mise en service et de maintenance. Des connaissances générales dans le domaine de la technique d'automatisation s'avèrent utiles.

Champ d'application

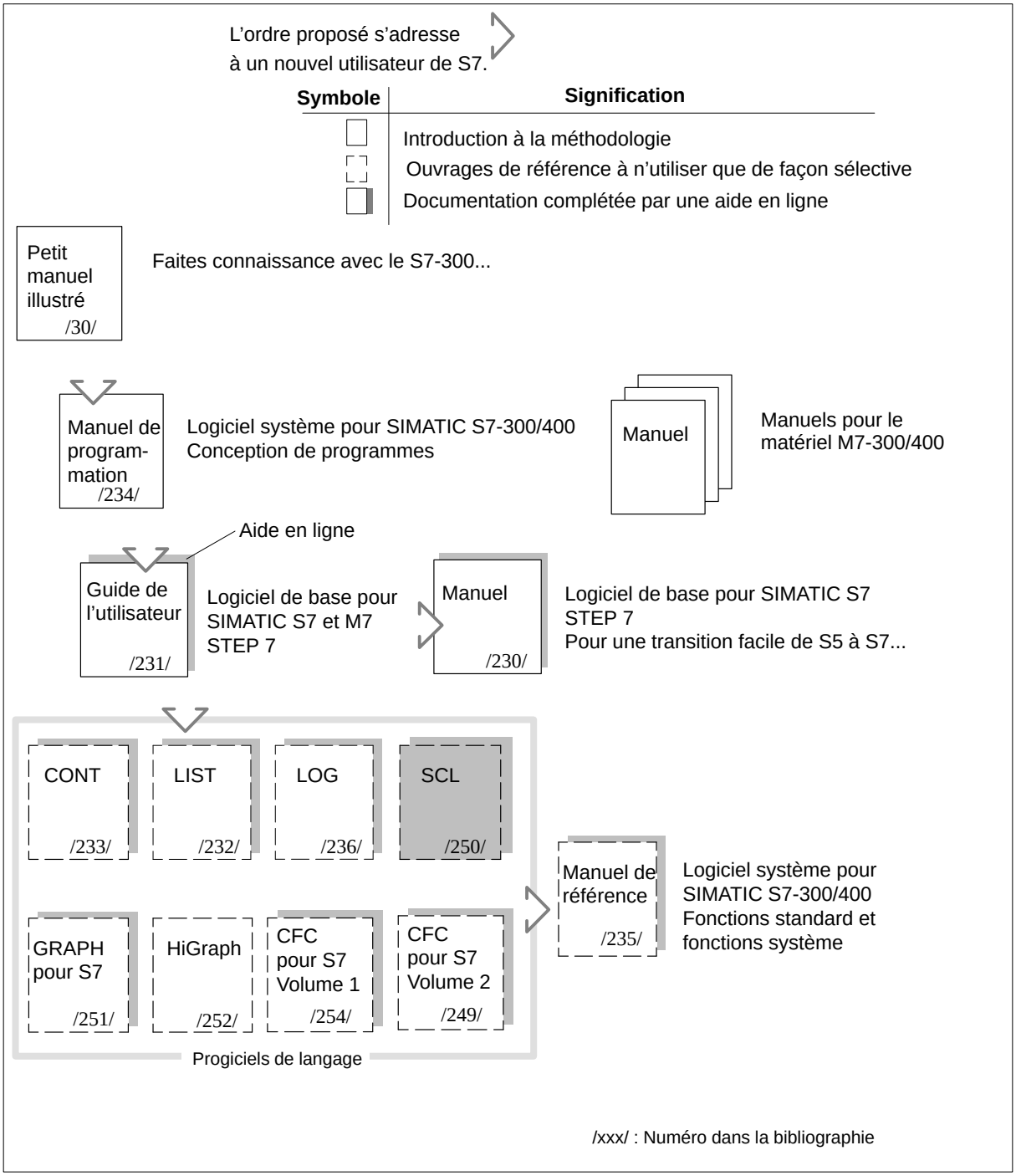
Ce manuel s'applique au logiciel de programmation STEP 7, à partir de la version 3.0. Il fait référence au logiciel de base STEP 7.

Conformité à la norme CEI 1131-3

SCL correspond au langage « liste d'instructions » (Structured Control Language) défini dans la norme DIN EN 61131-3 (CEI 1131-3), en présentant toutefois d'importantes différences pour ce qui est des opérations. Vous trouverez plus de détails à ce sujet dans la table de correspondance à la norme dans le fichier NORM_TBL.WRI (anglais) ou NORM_TAB.WRI (allemand) de STEP 7.

Vue d'ensemble de la documentation utilisateur

Il existe une importante documentation utilisateur destinée à vous aider pour la configuration et la programmation d'un automate programmable S7 et dont vous vous servirez de manière sélective. Les explications de la figure ci-après doivent faciliter l'utilisation de cette documentation.



Titre	Contenu
Petit manuel illustré Faites connaissance avec le S7-300...	Ce cahier constitue une introduction très simple à la méthodologie de configuration et de programmation d'un automate S7-300. Il s'adresse tout particulièrement aux utilisateurs ne connaissant pas les automates programmables S7.
Manuel de programmation Conception de programmes S7-300/400	Ce manuel de programmation présente les connaissances de base sur l'organisation du système d'exploitation et d'un programme utilisateur d'une CPU S7. Il est conseillé aux nouveaux utilisateurs des S7-300/400 de l'utiliser pour avoir une vue d'ensemble de la méthodologie de programmation et pour concevoir, ensuite, leur programme utilisateur.
Manuel de référence Fonctions standard et fonctions système S7-300/400	Les CPU S7 disposent de blocs d'organisation et de fonctions système intégrés dont vous pouvez vous servir lors de la programmation. Ce manuel présente une vue d'ensemble des fonctions système, blocs d'organisation et fonctions standard chargeables disponibles dans S7, ainsi que – comme informations de référence – des descriptions d'interface détaillées pour leur utilisation dans le programme utilisateur.
Guide de l'utilisateur STEP 7	Ce guide de l'utilisateur STEP 7 explique le principe d'utilisation et les fonctions du logiciel d'automatisation STEP 7. Que vous soyez un utilisateur débutant de STEP 7 ou que vous connaissiez bien STEP 5, il vous donne une vue d'ensemble sur la marche à suivre pour la configuration, la programmation et la mise en œuvre d'un automate S7-300/S7-400. Vous pouvez, lors de l'utilisation du logiciel, accéder de manière sélective à l'aide en ligne qui répondra à vos questions précises sur le logiciel.
STEP 7 Pour une conversion facile de S5 à S7... Manuel	Vous aurez besoin de ce guide si vous avez l'intention de convertir des programmes S5 existants afin de les exécuter dans des CPU S7. Ce guide vous donne une vue d'ensemble du mode de fonctionnement et de l'utilisation du convertisseur ; vous trouverez des informations détaillées sur l'utilisation des fonctions du convertisseur dans l'aide en ligne. Cette dernière contient également la description d'interface des fonctions S7 converties disponibles.
Manuels LIST, CONT, LOG, SCL[1]	Les manuels concernant les progiciels de langage LIST, CONT, LOG et SCL contiennent aussi bien des instructions pour l'utilisateur que la description du langage. Vous n'avez besoin, pour la programmation d'un S7-300/400, que de l'un de ces langages, mais pouvez les mélanger à l'intérieur d'un projet si besoin est. Il est conseillé, lors de la première utilisation des langages, de se familiariser avec la méthodologie de la création de programmes à l'aide du manuel. Dans le logiciel, vous pouvez appeler l'aide en ligne qui répondra à vos questions détaillées sur l'utilisation des éditeurs et compilateur associés.
Manuels GRAPH[1], HiGraph[1], CFC[1]	Les langages GRAPH, HiGraph et CFC offrent des possibilités supplémentaires pour la réalisation de commandes séquentielles, de graphes d'état ou de câblages graphiques de blocs. Ces manuels contiennent aussi bien des instructions pour l'utilisateur que la description du langage. Il est conseillé, lors de la première utilisation de ces langages, de se familiariser avec la méthodologie de la création de programmes à l'aide du manuel. Dans le logiciel, vous pouvez appeler l'aide en ligne (excepté pour HiGraph) qui répondra à vos questions détaillées sur l'utilisation des éditeurs et compilateurs associés.

1 Progiciels optionnels pour le logiciel système de S7-300/400

Structure du manuel

La connaissance théorique des programmes S7 est supposée acquise. Elle est présentée dans le manuel de programmation **/234/**. Puisque la mise en œuvre des progiciels de langage requiert le logiciel de base, vous devriez être familiarisé avec l'utilisation de ce dernier, décrite dans le guide de l'utilisateur**/234/**.

Ce manuel est organisé d'après les thèmes suivants :

- Le chapitre 1 contient des informations générales sur la programmation avec SCL.
- La chapitre 2 traite de la conception d'un processus en s'appuyant sur un exemple que vous avez la possibilité de mettre en application.
- Le paragraphe 3-6 vous montre comment utiliser l'environnement de développement de SCL. Vous y apprendrez à vous servir de l'éditeur, du compilateur et du débogueur de SCL.
- Les chapitres 7 à 9 constituent la partie de référence, qui vous donne des informations détaillées sur la fonction de chaque instruction de SCL.

En annexe vous trouverez :

- La représentation syntaxique complète de SCL.
- Un glossaire dans lequel les termes essentiels sont expliqués.
- Un index vous permettant de localiser rapidement les termes principaux.

Conventions

Les renvois à d'autres publications sont indiqués à l'aide de numéros entre barres obliques /.../. Vous trouverez, à l'aide de ces numéros, le titre exact de ces publications dans la bibliographie à l'annexe D.

Aide supplémentaire

Adressez-vous à votre agence Siemens pour toute question sur le logiciel décrit à laquelle vous ne trouveriez pas de réponse dans la documentation papier ou dans l'aide en ligne. Vous trouverez les adresses des agences Siemens dans le monde à l'annexe des publications **/70/** ou **/100/**, dans les catalogues ainsi que sur Compuserve (`go autforum`).

En outre, notre ligne rouge est également à votre disposition :

Tel. +49(911) 895-7000 (Fax 7001)

Si vous avez des questions ou des remarques sur le présent manuel, nous vous prions de compléter le formulaire à la fin du manuel et de l'envoyer à l'adresse indiquée. N'hésitez pas à également indiquer votre appréciation personnelle du manuel.

Nous proposons des cours pour faciliter l'apprentissage des automates programmables SIMATIC S7. Adressez-vous à votre centre de formation ou à notre centre principal à :

D-90327 Nürnberg, tél. 19/ 49 / 911 / 895 3154.

Remarque

La partie utilisateur de ce manuel ne contient pas de notice d'instructions à effectuer, mais fournit des principes de base pour la programmation avec SCL. De plus amples informations sur l'interface du logiciel et sur son utilisation sont données dans l'aide en ligne.

Sommaire

Pemière partie : Conception

Deuxième partie : Utilisation et test

Troisième partie : Description du langage

Annexes

Première partie : Conception

Présentation du produit

1

Langage de programmation SCL

Il est de plus en plus fréquent que l'on exige des systèmes d'automatisation qu'ils soient capables de résoudre non seulement les tâches de commande classiques, mais également des tâches de gestion de données ainsi que des fonctions mathématiques complexes. C'est la raison pour laquelle nous vous proposons SCL pour S7-300/400 (Structured Control Language), le langage de programmation tout particulièrement conçu pour faciliter la programmation de telles tâches, conformément à la norme CEI 113-3.

SCL vous permet, outre la résolution de tâches de commande « normales », le traitement d'applications particulièrement volumineuses et est de fait supérieur aux langages de programmation « classiques » dans les domaines suivants :

- gestion de données
- optimisation de processus
- gestion de formulations
- fonctions mathématiques/statistiques

Caractéristiques techniques

Pour pouvoir utiliser SCL, il vous faut une console de programmation SIMATIC ou un PC (à partir du processeur 80486, mémoire RAM de 16 Mo).

Eléments de langage

Opérateurs	`Puissance/arithmétique`
	`Comparaisons/`
	`Combinaisons`

Fonctions	`Temporisations/Compteurs` `Appels de blocs fonctionnels`
Structures de contrôle	`Boucles (FOR/WHILE/REPEAT)` `Branchements (IF THEN/CASE/GOTO)`

Types de données

simples	`BOOL/BYTE/WORD/DWORD`
	`INT/DINT/REAL`
	`DATE/TIME/TIME_OF_DAY/S5TIME`

complexes	`Chaînes de caractères/tableaux/`
	`structures/structures utilisateur/`
	`DATE_AND_TIME`

Structure du chapitre

1.1 Présentation de SCL

Langage de programmation évolué

SCL (*Structured Control Language*) est un langage de programmation évolué proche du PASCAL, conforme à une norme sur les automates programmables.

La norme DIN EN 61131-3 (qui correspond à la norme internationale CEI 1131-3) normalise les langages de programmation destinés aux automates programmables. La partie traitant du « texte structuré » constitue le fondement de SCL. Vous trouverez plus de détails à ce sujet dans la table de correspondance à la norme dans le fichier NORM_TBL.WRI (anglais) ou NORM_TAB.WRI (allemand) de STEP 7.

Outre les éléments du langage évolué, SCL dispose également d'éléments de langage spécifiques aux automates programmables, comme les entrées, sorties, temporisations, mémentos, appels de blocs etc.. SCL complète et élargit ainsi le spectre du logiciel de programmation STEP 7 avec ses langages de programmation CONT, LOG et LIST.

Environnement de développement

Un environnement de développement répondant à la fois aux exigences de SCL et de STEP 7 permet d'optimiser et de simplifier l'utilisation de SCL. Il comprend :

- un **éditeur**, permettant d'écrire des programmes avec des fonctions (FC), des blocs fonctionnels (FB), des blocs d'organisation (OB), des blocs de données (DB) et des types de données utilisateur (UDT). Des fonctions puissantes assistent le programmeur dans sa tâche.
- un **compilateur séquentiel** pour compiler le programme préalablement édité en code machine MC7. Celui-ci est exécutable sur toutes les CPU des systèmes d'automatisation S7-300/400 à partir de la CPU 314.
- un **débogueur**, permettant de rechercher des erreurs de programmation logiques dans un programme correctement compilé. La recherche d'erreurs s'effectue dans le langage source.

L'utilisation de ces divers éléments est simple et conviviale, puisqu'ils sont tous exécutables sous Windows 95 et présentent ainsi tous les avantages offerts par ce système d'exploitation.

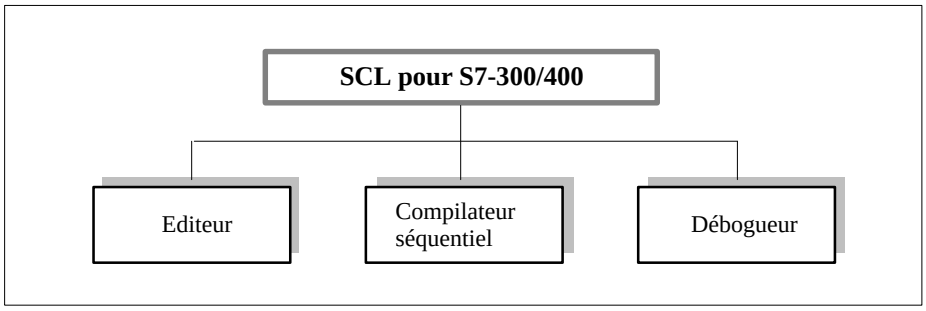


Figure 1-1 *Environnement de développement de SCL*

1.2 Avantages offerts par SCL

Langage de programmation évolué

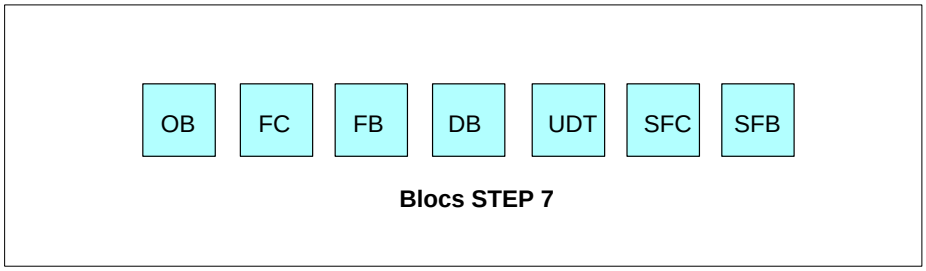
SCL propose non seulement tous les avantages d'un langage de programmation évolué, mais également des propriétés qui ont tout particulièrement été conçues pour permettre une programmation structurée, comme par exemple :

- la structure de blocs de STEP 7,
- des modèles de blocs,
- la compatibilité avec STEP 5.

Conservation de la structure de bloc de STEP 7

SCL permet de résoudre de manière optimale les diverses tâches qui composent un projet d'automatisation, si bien qu'il peut être mis en œuvre efficacement avec STEP 7 dans chaque phase du projet.

En appliquant en particulier le concept de blocs de STEP 7, SCL permet comme LIST, LOG et CONT, la programmation normée de blocs.



Blocs STEP 7

Types de blocs

De par leur fonction, leur structure ou leur utilisation, les blocs de STEP 7 sont des sections limitées d'un programme utilisateur. SCL vous permet de créer les blocs suivants :

Nom	Type de bloc	Fonction
OB	Bloc d'organisation	Interface entre le système d'exploitation et le programme utilisateur.
FC	Fonction	Bloc permettant la transmission de paramètres sans mémoire.
FB	Bloc fonctionnel	Bloc permettant la transmission de paramètres et possédant une mémoire.
DB	Bloc de données	Bloc permettant d'enregistrer des données utilisateur.
UDT	Type de données utilisateur	Bloc permettant d'enregistrer un type de données utilisateur.

Blocs prédéfinis

Vous n'êtes pas amenés à définir chaque fonction par vous même, mais avez la possibilité d'utiliser des blocs prédéfinis. Ils se trouvent dans le système d'exploitation de l'unité centrale ou dans des bibliothèques (*S7lib*) du logiciel de base STEP 7 et peuvent être utilisés pour programmer des fonctions de communication. Il s'agit des types de blocs suivants :

Nom	Type de bloc	Fonction
SFC	Fonction système	Mêmes propriétés qu'une fonction (FC)
SFB	Bloc fonctionnel système	Mêmes propriétés qu'un bloc fonctionnel (FB)

Compatibilité des blocs

Les blocs programmés dans SCL sont compatibles avec les blocs LIST, CONT et LOG. Ceci signifie qu'un bloc programmé dans SCL peut appeler un bloc programmé dans LIST, CONT ou LOG et inversement. Ainsi, les langages de programmation STEP 7 et SCL (logiciel optionnel) se complètent parfaitement.

Décompilation

La décompilation de blocs de SCL dans le langage de programmation LIST (liste d'instructions) de STEP 7 est possible, la décompilation dans SCL ne l'étant pas.

Compatibilité avec STEP 5

A quelques rares exceptions près, la compatibilité ascendante des blocs créés avec SCL dans STEP 5 est assurée, ce qui signifie que ces blocs peuvent également être édités, compilés et testés avec SCL pour STEP 7.

Méthodes de programmation

Grâce à des méthodes modernes de génie logiciel, SCL applique la programmation structurée.

Apprentissage

Une expérience succincte dans un langage de programmation évolué est suffisante pour pouvoir aisément apprendre à utiliser SCL, car l'éventail des éléments de langage de SCL est orienté vers de tels langages évolués.

1.3 Caractéristiques de l'environnement de développement

Editeur

L'éditeur de SCL sert à éditer des textes quelconques. Il vous permet principalement de créer et d'éditer des fichiers source pour vos programmes STEP 7. Vous pouvez programmer un ou plusieurs blocs dans un fichier source :

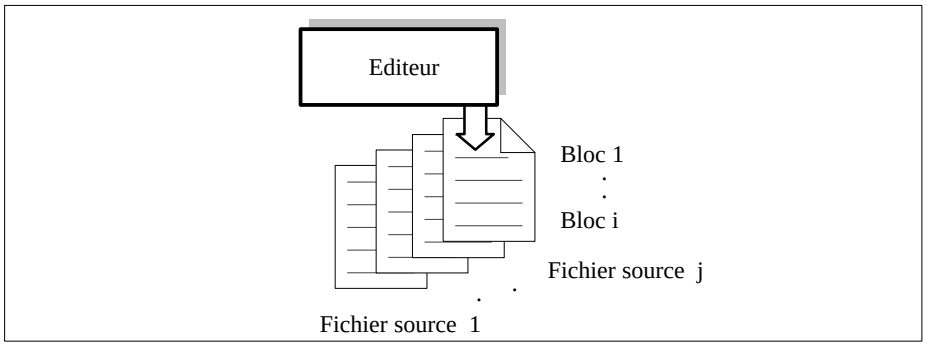


Figure 1-2 *Editeur de SCL*

Avec l'éditeur de SCL vous pouvez :

- éditer un fichier source complet contenant un ou plusieurs blocs,
- éditer un fichier d'informations compilation vous permettant d'automatiser la compilation de plusieurs fichiers source,
- utiliser des fonctions supplémentaires vous rendant l'édition du texte source plus aisée, comme par exemple les fonctions de recherche et de remplacement,
- optimiser les options de l'éditeur selon vos exigences particulières.

La vérification de la syntaxe n'est pas réalisée durant la saisie.

Compilateur

Après avoir créé votre fichier source avec l'éditeur de SCL, vous devez le compiler en code machine.

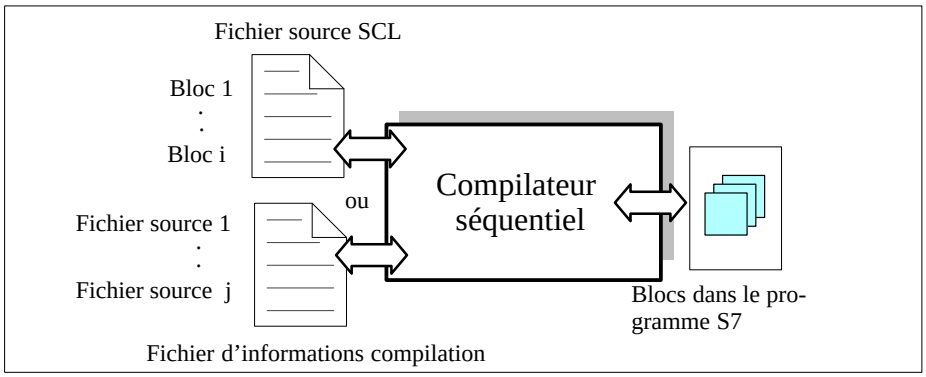


Figure 1-3 *Compilateur de SCL*

Avec le compilateur de SCL vous pouvez :

- compiler un fichier source SCL comportant plusieurs blocs en une seule procédure,
- compiler plusieurs fichiers source SCL à l'aide d'un fichier d'informations compilation contenant les noms des fichiers source,
- optimiser les options du compilateur selon vos exigences particulières,
- faire afficher toutes les erreurs et avertissements qui se produisent durant la compilation,
- localiser aisément l'erreur dans le texte source, et obtenir optionnellement une description de l'erreur et des informations permettant d'y remédier.

Débogueur

Le débogueur de SCL offre la possibilité de contrôler l'exécution d'un programme dans l'AP afin d'y déceler d'éventuelles erreurs logiques.

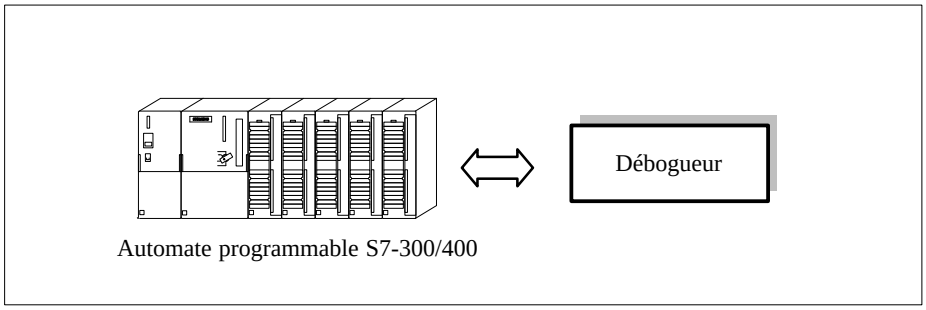


Figure 1-4 *Débogueur de SCL*

SCL dispose de deux modes de test :

- la **visualisation séquentielle** qui permet de suivre l'exécution logique du programme. Vous pouvez exécuter l'algorithme du programme instruction après instruction et visualiser simultanément la modification des contenus des variables édités dans une fenêtre de résultat.
- la **visualisation continue** qui permet de tester un groupe d'instructions dans un bloc du fichier source. Durant le test, les valeurs des variables et des paramètres s'affichent chronologiquement et sont – si cela est possible – actualisées cycliquement.

Logiciel de base STEP 7

L'environnement de développement de SCL offre la possibilité d'exploiter directement à partir de SCL des fonctions du logiciel de base STEP 7 comme, par exemple, l'affichage et la modification de l'état de fonctionnement de la CPU et le réglage de l'heure.

Développement d’un programme SCL

2

Conception

La pratique le montre : vous programmez plus facilement et plus rapidement lorsque vous divisez votre tâche en entités plus petites afin de la structurer. SCL optimise et rationalise la programmation de blocs.

Le présent chapitre traite du développement et de la mise en œuvre d’un programme utilisateur dans SCL. La description s’appuye sur un exemple de programmation que vous pourrez tester avec vos modules d’entrée et de sortie, grâce aux données de test fournies.

Structure du chapitre

Ce chapitre traite des thèmes suivants :

2.1 Présentation

Objectifs

L'exemple proposé vous montre comment utiliser au mieux les possibilités offertes par SCL. Le présent chapitre a pour but de répondre aux questions – énoncées ci-après – que vous vous posez le plus fréquemment :

- comment puis-je procéder pour développer un programme dans SCL ?
- quels sont les éléments de langage de SCL appropriés à la résolution de cette tâche ?
- quelles sont les fonctions de test mises à ma disposition ?

Eléments du langage SCL

L'exemple proposé présente les éléments suivants du langage SCL :

- structure et utilisation des divers types de blocs
- appel de blocs avec transmission et exploitation des paramètres
- divers formats d'entrée et de sortie
- programmation à l'aide de types de données simples et de tableaux
- initialisation de variables
- structure d'un programme avec utilisation de branchements et de boucles

Configuration matérielle

Vous pouvez exécuter le programme donné en exemple dans un automate SIMATIC S7-300 ou SIMATIC S7-400 en utilisant comme périphéries :

- un module d'entrée avec 16 voies
- un module de sortie avec 16 voies

Fonctions de test

Le programme a été conçu pour pouvoir être testé rapidement à l'aide des commutateurs de l'entrée et des organes d'affichage de la sortie. Pour effectuer un test complet en utilisant les fonctions de test de SCL, reportez-vous au chapitre 6.

Vous disposez en outre des fonctions du logiciel de base STEP 7 quel que soit le langage utilisé.

2.2 Enoncé de la tâche à résoudre

Présentation

Il s'agit de saisir, de trier puis de traiter des valeurs de mesure à l'aide du module d'entrée. Pour une plage de valeurs de mesure comprise entre 0 et 255, il faut un octet pour effectuer la saisie. Comme fonctions, l'on utilisera la racine carrée et le carré. Les résultats doivent être affichés sur le module de sortie, ce qui nécessite un mot. C'est un octet d'entrée qui doit commander le programme.

Saisie des valeurs de mesure

Une valeur de mesure que vous sélectionnez sur les 8 commutateurs de saisie doit être enregistrée dans le tablau des valeurs de mesure de la mémoire au moment où un front est détecté sur le commutateur de saisie – voir figure 2-1. Ce tableau de valeurs de mesure doit être organisé en mémoire circulante à 8 entrées au maximum.

Traitement des valeurs de mesure

Le traitement doit comporter les deux étapes suivantes : tri des valeurs de mesure et calcul des résultats, voir figure 2-1 :

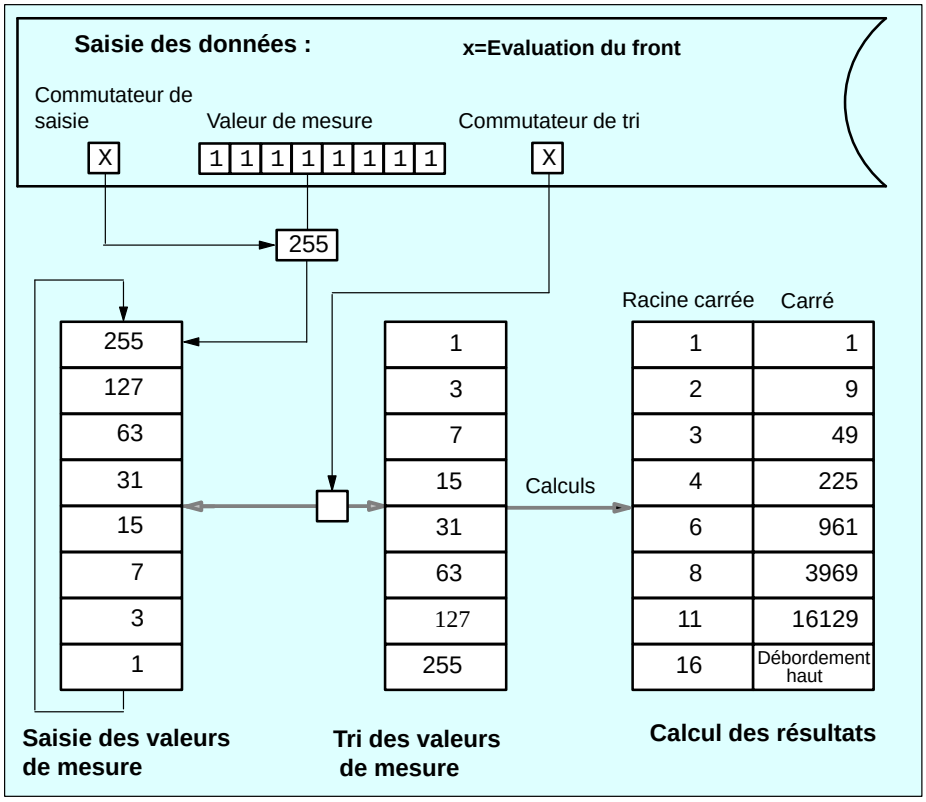


Figure 2-1 *Saisie et traitement des valeurs de mesure*

Sélection de la sortie

Puisque la sortie ne peut afficher qu'une seule valeur à la fois, vous devez pouvoir choisir entre :

- la sélection d'un élément dans une liste,
- la sélection de la valeur de mesure, de la racine carrée ou du carré.

Les réglages suivants des commutateurs doivent permettre d'adresser un élément dans une liste afin de le sélectionner :

- Trois commutateurs permettent de sélectionner un codage qui est validé lorsqu'un front est détecté sur le quatrième commutateur, le commutateur de codage. Ceci permet de déterminer l'adresse de la sortie – voir figure 2-2.
- La même adresse fournit trois valeurs de sortie, à savoir la valeur de mesure, la racine carrée et le carré. Deux inverseurs – représentés à la figure 2-2 – vous permettent de sélectionner l'une d'entre-elles.

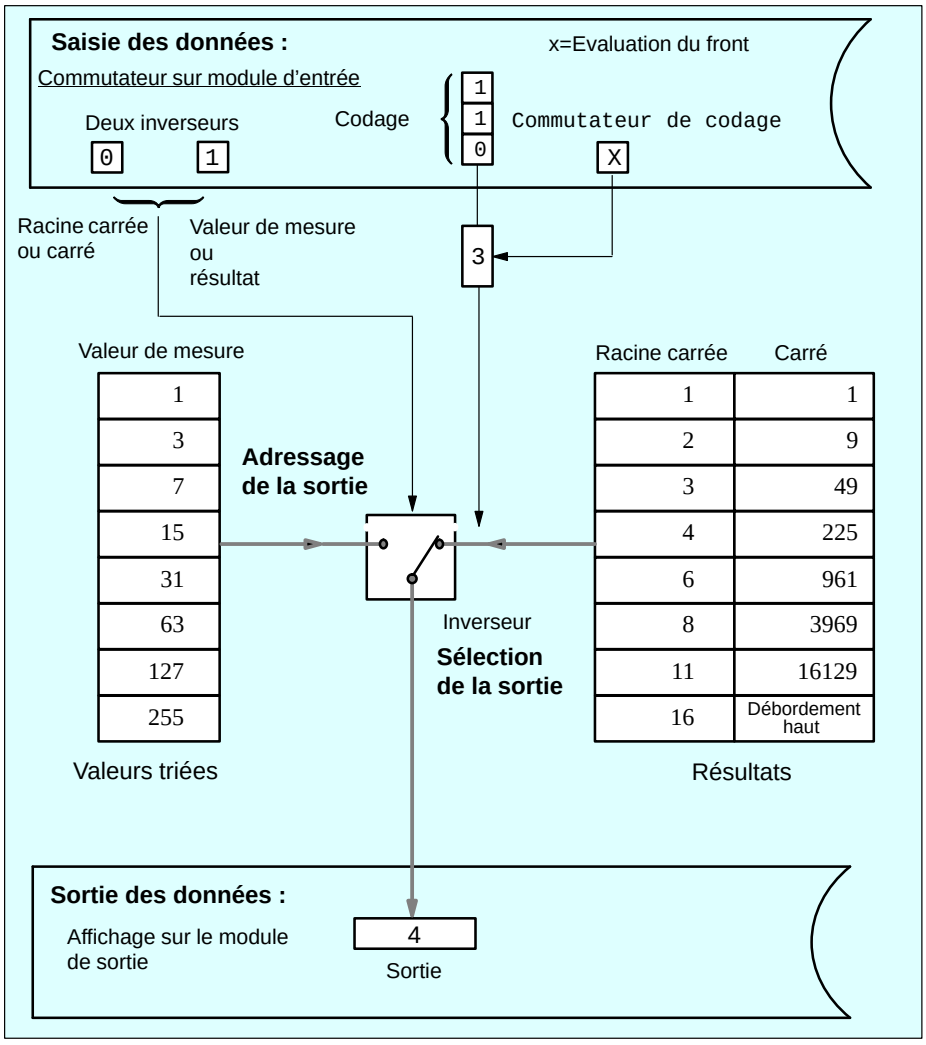


Figure 2-2 *Sélection de la sortie*

2.3 Solution utilisant des blocs SCL

Présentation

Nous vous proposons de résoudre la tâche énoncée sous forme de **programme SCL structuré**. Il s'agit d'un programme modulaire, c'est-à-dire divisé en blocs traitant une ou plusieurs tâches partielles. Comme dans les langages de programmation de STEP 7, vous disposez dans SCL de plusieurs types de blocs. Pour plus de détails, reportez-vous aux chapitres 1, 7 et 8.

Différentes étapes

Vous pouvez procéder par les étapes suivantes :

1. Définition des tâches partielles
2. Choix et affectation des blocs possibles
3. Définition des interfaces entre les blocs
4. Définition de l'interface d'entrée/sortie
5. Programmation des blocs

2.3.1 Délimitation des tâches partielles

Présentation

Les tâches partielles sont représentées à la figure 2-3 sous forme de pavés. Les zones rectangulaires grises représentent les blocs. La disposition des blocs de code de la gauche vers la droite correspond à leur ordre d'appel :

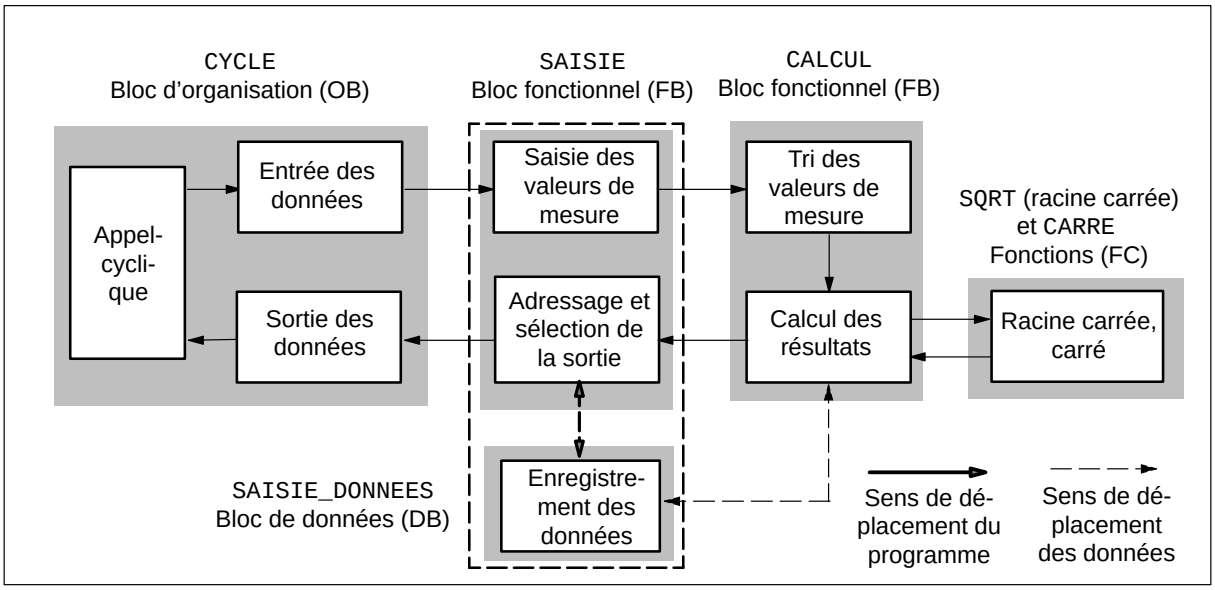


Figure 2-3 *Formation de blocs pour la résolution des diverses tâches partielles*

2.3.2 Choix et affectation des blocs possibles

Présentation

Voici les critères ayant permis de choisir les blocs :

CYCLE

Un programme utilisateur ne peut être démarré que dans un OB. Puisque les valeurs de mesure vont continuellement être fournies et devoir être saisies, vous devez utiliser un OB d'*appel cyclique* (OB1). Une partie de l'exécution – l'*entrée des données* et de la *sortie des données* – va être programmée dans l'OB.

SAISIE

Pour résoudre la tâche partielle de *saisie des valeurs de mesure* vous devez utiliser un bloc possédant une mémoire, c'est-à-dire un FB, car certaines données locales (par exemple la mémoire circulante) doivent être conservées d'un cycle de programme au suivant. C'est dans le bloc de données d'instance SAISIE_DONNEES, également appelé mémoire que va s'effectuer l'*enregistrement des données*.

Le même FB peut également réaliser l'*adressage et la sélection de la sortie*, puisqu'il dispose des données requises.

CALCUL

Dans le choix du type de bloc à utiliser pour effectuer le *tri des valeurs de mesure* et le *calcul des résultats*, il faut tenir compte du fait qu'une mémoire de sortie doit être créée pour contenir les résultats des calculs de la racine carrée et du carré de chaque valeur de mesure.

Seul un FB répond à ces critères. Puisqu'il est appelé par un autre FB de niveau hiérarchique supérieur, l'utilisation de son propre DB s'avère inutile et ses données d'instance peuvent être inscrites dans le bloc de données d'instance du FB appelant.

SQRT (racine carrée) et CARRE

Pour la résolution de la tâche partielle de *calcul de la racine carrée ou du carré*, une FC convient le mieux, car la sortie du résultat correspond à la valeur de la fonction. En outre, ce calcul ne nécessite pas de données devant être conservées pendant plus d'un cycle de programme.

Pour le calcul de la racine carrée, vous pouvez utiliser la fonction standard de SCL, la fonction SQRT. Pour calculer le carré, vous allez créer la fonction CARRE qui doit également vérifier les limites de la plage des valeurs autorisée.

2.3.3 Définition des interfaces entre les blocs

Présentation

Pour définir l'interface d'un bloc à un autre, vous devez déclarer ses **paramètres formels**. Dans SCL, il existe les possibilités suivantes :

- paramètres d'entrée : déclaration avec VAR_INPUT
- paramètres de sortie : déclaration avec VAR_OUTPUT
- paramètres d'entrée/sortie : déclaration avec VAR_IN_OUT

Lorsque vous appelez un bloc, des données d'entrée lui sont transmises comme **paramètres effectifs**. Après retour au bloc appelant, les données de sortie sont préparées pour y être validées. Une FC peut fournir son résultat sous forme de **valeur de la fonction** (pour plus d'informations, reportez-vous au chapitre 16).

SAISIE

L'OB CYCLE ne possède pas lui-même de paramètres formels. Il appelle le FB SAISIE pour transmettre la valeur de mesure et les données de commande aux paramètres formels de ce dernier (tableau 2-1) :

Tableau 2-1 Paramètres formels du FB SAISIE

Nom du paramètre	Type de données	Type de déclaration	Description
entrée_val_mesure	INT	VAR_INPUT	Valeur de mesure
nouv_val	BOOL	VAR_INPUT	Commutateur pour inscrire la valeur de mesure dans la mémoire circulante
nouv_tri	BOOL	VAR_INPUT	Commutateur pour trier et traiter les valeurs de mesure
choix_fonction	BOOL	VAR_INPUT	Inverseur permettant de choisir la racine carrée ou le carré
sélection	WORD	VAR_INPUT	Codage permettant de sélectionner la valeur de sortie
nouv_sélection	BOOL	VAR_INPUT	Commutateur permettant de valider le codage
sortie_résultat	DWORD	VAR_OUTPUT	Sortie du résultat calculé
sortie_val_mesure	DWORD	VAR_OUTPUT	Sortie de la valeur de mesure correspondante

CALCUL

Le FB SAISIE appelle le FB CALCUL. Comme données communes, ils possèdent le tableau des valeurs de mesure à trier, qui sera donc déclaré comme paramètre d'entrée/sortie. Pour les résultats des calculs de la racine carrée et du carré, c'est un tableau structuré qui va être créé comme paramètre de sortie. Les paramètres formels sont décrits dans le tableau 2-2 :

Tableau 2-2 Paramètres formels du FB CALCUL

Nom du paramètre	Type de données	Type de déclaration	Description
mémoire_tri	ARRAY[..] OF REAL	VAR_IN_OUT	Tableau des valeurs de mesure, correspond à la mémoire circulante
mémoire_calcul	ARRAY[..] OF STRUCT	VAR_OUTPUT	Tableau des résultats : structure avec les composantes « racine_carrée » et « carré » du type INT

SQRT et CARRE

Ces fonctions sont appelées par le FB CALCUL. Elles requièrent une valeur d'entrée et fournissent leur résultat comme valeur de la fonction, voir tableau 2-3.

Tableau 2-3 Paramètres formels et valeurs des fonctions SQRT et CARRE

Nom	Type de données	Type de déclaration	Description
valeur	REAL	VAR_INPUT	Entrée pour SQRT
SQRT	REAL	Valeur de la fonction	Racine carrée de la valeur d'entrée
valeur	INT	VAR_INPUT	Entrée pour CARRE
CARRE	INT	Valeur de la fonction	Carré de la valeur d'entrée

2.3.4 Définition de l'interface d'entrée/sortie

Présentation

La figure 2-4 représente l'interface d'entrée/sortie. Observez que pour l'entrée/sortie sous forme d'octet, l'octet de poids faible se trouve en haut et l'octet de poids fort en bas. Pour l'entrée/sortie sous forme de mot c'est l'inverse :

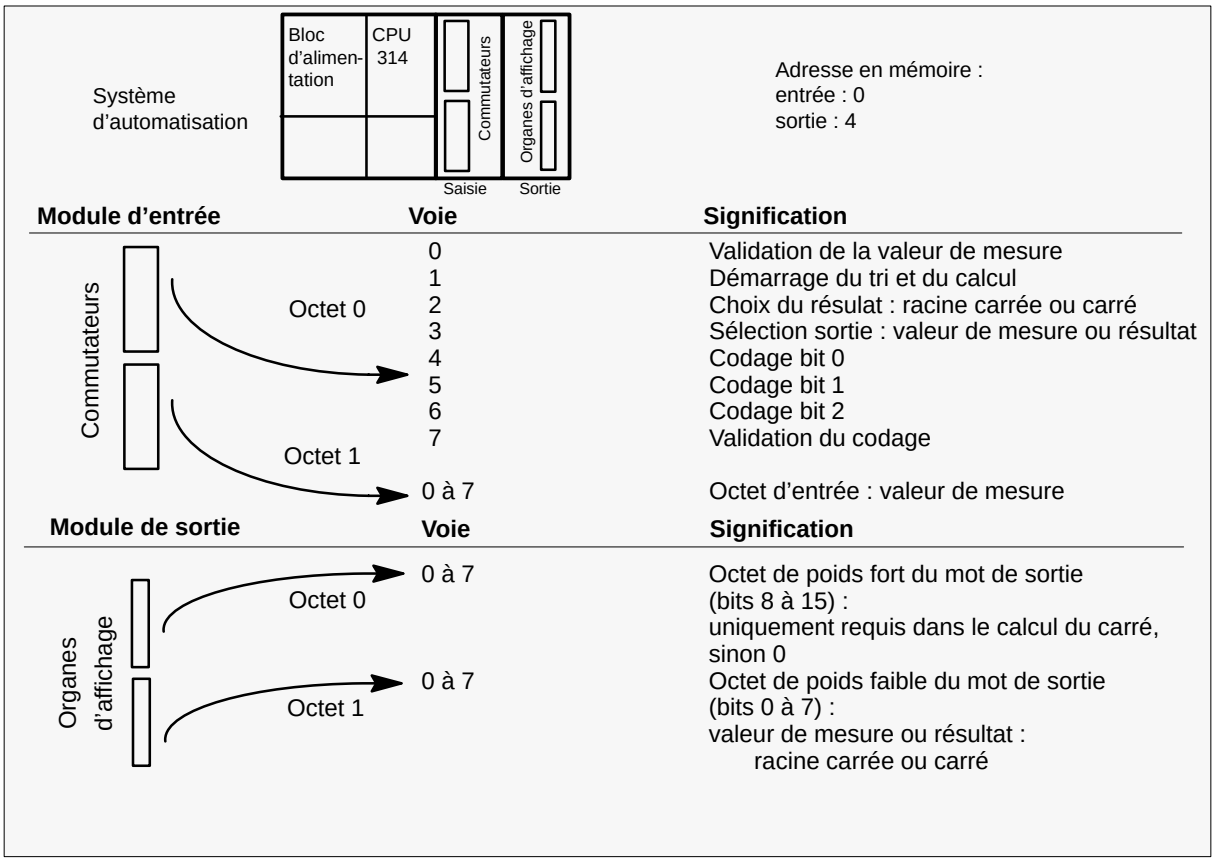


Figure 2-4 *Organes d'affichage et éléments de commande*

2.3.5 Programmation des blocs

Programmation de blocs

Une fois que vous avez défini les interfaces, vous pouvez écrire les blocs séparément. Il est préférable de procéder de haut en bas, c'est-à-dire dans l'ordre CYCLE, SAISIE, CALCUL et CARRE, comme décrit ci-après.

La compilation de blocs impose qu'un bloc ait été créé avant d'être utilisé, c'est-à-dire avant d'être appelé par un autre bloc. Dans la source SCL, l'ordre des blocs est donc le suivant : CARRE, CALCUL, SAISIE et CYCLE (pour des informations à ce sujet, reportez-vous au chapitre 8).

Programmation symbolique

Vous contribuez à une meilleure lisibilité du programme si vous affectez des **mnémoniques** aux adresses des modules et aux blocs. Vous devez pour cela compléter une table de mnémoniques, comme décrit à la figure 2-5 (voir chapitre 7). Vous pouvez former les mnémoniques en appliquant la convention de désignation des IDENTIFICATEURS ou des mnémoniques (par exemple « Entrée 0.0 »), voir annexe A.

Commentaire d'introduction avec table des mnémoniques

La figure 2-5 représente le commentaire d'introduction de la source SCL. Il décrit les noms symboliques définis dans la table des mnémoniques afin que la source puisse être compilée sans erreur.

```
(*#############################################################################################

Programme SCL pour la saisie et le traitement de valeurs de mesure :

-     L'entrée 0.0 (commutateur de saisie) permet de valider une valeur
      de mesure lue sur le module d'entrée.
-     Divers commutateurs permettent le traitement des valeurs de mesure.
-     Toutes les valeurs sont enregistrées dans la zone de traitement du bloc
      fonctionnel SAISIE, à savoir le bloc de données d'instance SAISIE_DONNEES.

Le programme utilise l'écriture symbolique. Pour qu'il puisse être compilé,
des mnémoniques doivent avoir été affectés aux adresses des modules et
aux blocs exécutés dans la CPU. C'est la fonction du tableau suivant :

Saisie                 EB1   BYTE  // Valeur de mesure
Entrée 0.0             E0.0  BOOL  // Commutateur de saisie pour valider la valeur de mesure
Commutateur_de_tri     E0.1  BOOL  // Démarrage du tri et du calcul
Commutateur_de_fonction E0.2 BOOL  // Choix du résultat racine carrée ou carré
Commutateur_de_sortie  E0.3  BOOL  // Sélection de la sortie valeur de mesure ou résultat
Codage                 EW0   WORD  // Codage, bits significatifs 12, 13 et 14
Commutateur_de_codage  E0.7  BOOL  // Validation du codage
Sortie                 AW4   INT   // Valeur de mesure ou résultat : racine carrée ou carré

SAISIE                 FB10  FB10  // Saisie des valeurs de mesure
                                   // Adressage et sélection de la sortie
SAISIE_DONNEES         DB10  FB10  // Bloc de données d'instance du FB SAISIE
CALCUL                 FB20  FB20  // Traitement des valeurs de mesure, calcul des résultats
CARRE                  FC41  FC41  // Fonction de calcul du carré
CYCLE                  OB1   OB1   // Appel cyclique et entrée/sortie

#############################################################################################*)
```

Figure 2-5 *Commentaire d'introduction avec table des mnémoniques*

2.4 Programmation du bloc d'organisation CYCLE

Etapes à suivre

Nous avons sélectionné un OB1 car STEP 7 l'appelle **cycliquement**. Il réalise les tâches suivantes dans le programme :

- appel du bloc fonctionnel SAISIE auquel il fournit les données de saisie et de commande,
- validation des données de résultat du bloc fonctionnel SAISIE,
- sortie des valeurs à afficher.

La zone de données temporaires « données_système » de 20 octets se trouve au début de la section de déclaration (voir aussi chapitre 8).

```
ORGANIZATION_BLOCK CYCLE

(*******************************************************************************
      Le FB CYCLE correspond à l'OB1, c.-à-d. il est appelé cycliquement par le système S7
      1ère partie :  appel du bloc fonctionnel et saisie des valeurs d'entrée
      2nde partie :  validation des valeurs de sortie et sortie avec sélection
*******************************************************************************)
VAR_TEMP
      données_système   :  ARRAY[0..20] OF BYTE;      // Zone de l'OB1
END_VAR

BEGIN

(*    1ère partie :  ***************************************************************)
SAISIE.SAISIE_DONNEES(
         entrée_val_mesure := WORD_TO_INT(saisie),
         nouv_val          := "Entrée 0.0",     //commutateur de saisie sous forme
                                                //de mnémonique
         nouv_tri          := Commutateur_de_tri,
         choix_fonction    := Commutateur_de_fonction,
         nouv_sélection    := Commutateur_de_codage,
         sélection         := Codage);

(*    2nde partie :  ***************************************************************)
IF Commutateur_de_sortie THEN                        //Commutation de la sortie
      Sortie   := SAISIE_DONNEES.sortie_résultat;    //Racine carrée ou carré
ELSE
      Sortie   := SAISIE_DONNEES.sortie_val_mesure;  //Valeur de mesure
END_IF;

END_ORGANIZATION_BLOCK
```

Figure 2-6 *Bloc d'organisation CYCLE (OB1)*

Conversion du type de données

La valeur de mesure fournie à l'entrée possède le type de données BYTE et doit être convertie dans le type INT : vous devez donc effectuer une conversion de WORD en INT, la conversion précédente de BYTE en WORD étant effectuée automatiquement par le compilateur (voir chapitre 18). La conversion de la sortie s'avère inutile, puisque son type de données déclaré dans la table des mnémoniques est INT, voir figure 2-5.

2.5 Programmation du bloc fonctionnel SAISIE

Etapes à suivre

C'est le type de bloc FB qui a été choisi, car des données doivent être enregistrées d'un cycle de programme au suivant. Il s'agit de variables statiques, déclarées dans la partie déclarative « VAR, END_VAR », voir tableau 2-4.

Les variables statiques sont des variables locales dont les valeurs subsistent au-delà des cycles de blocs. Elles servent à stocker les valeurs d'un **bloc fonctionnel** et sont rangées dans le bloc de données d'instance.

```
FUNCTION_BLOCK SAISIE
(*************************************************************************
      1ère partie :  Saisie des valeurs de mesure
      2ème partie :  Démarrage du tri et du calcul
      3ème partie :  Evaluation du codage et préparation de la sortie
*************************************************************************)
```

Figure 2-7 *En-tête du bloc fonctionnel SAISIE*

Variables statiques

Tableau 2-4 Variables statiques du FB SAISIE

Nom	Type de données	Type de déclaration	Valeur défaut	Description
valeurs_mesure	ARRAY [..] OF INT	VAR	8(0)	Mémoire circulante pour les valeurs de mesure
mémoire_résultat	ARRAY [..] OF STRUCT	VAR	–	Tableau pour les structures avec les composantes « racine_carrée » et « carré » de type INT
index	INT	VAR	0	Index pour la mémoire circulante, la valeur de mesure suivante va y être entrée
anc_val	BOOL	VAR	FALSE	Valeur précédente pour validation de la valeur de mesure avec « nouv_val »
anc_tri	BOOL	VAR	FALSE	Valeur précédente pour tri avec « nouv_tri »
anc_sélection	BOOL	VAR	FALSE	Valeur précédente pour validation du codage avec « nouv_sélection »
adresse	INT	VAR	0	Adresse de la sortie de la valeur de mesure ou du résultat
instance_calcul	CALCUL, = FB 20	VAR	–	Instance locale pour le FB CALCUL

Tenez compte des valeurs par défaut qui sont affectées aux variables lors de l'initialisation du bloc (après son chargement dans la CPU). L'instance locale pour le FB CALCUL est également déclarée dans la partie déclarative « VAR, END_VAR ». Vous utiliserez son nom pour appeler et adresser ultérieurement les paramètres de sortie. Comme mémoire des données, c'est l'instance globale SAISIE_DONNEES que l'on utilise.

Section de déclaration de SAISIE

La section de déclaration de ce bloc comporte les parties suivantes :

- déclaration des constantes : entre CONST et END_CONST
- paramètres d'entrée : entre VAR_INPUT et END_VAR
- paramètres de sortie : entre VAR_OUTPUT et END_VAR
- déclaration des variables statiques : entre VAR et END_VAR.
  La déclaration de l'instance locale pour le FB CALCUL en fait également partie.

```
CONST
      LIMITE            := 7;
      NOMBRE            := LIMITE + 1;
END_CONST

VAR_INPUT
      entrée_val_mesure :  INT;     //    Nouvelle valeur de mesure
      nouv_val          :  BOOL;    //    Valider la valeur de mesure dans la mémoire
                                    //    circulante "valeurs_mesure"
      nouv_tri          :  BOOL;    //    Trier les valeurs de mesure
      choix_fonction    :  BOOL;    //    Choix de la fonction de calcul de la racine
                                    //    carrée ou du carré
      nouv_sélection    :  BOOL;    //    Validation de l'adresse de sortie
      sélection         :  WORD;    //    Adresse de sortie
END_VAR

VAR_OUTPUT
      sortie_résultat   :  INT;     //    Valeur calculée
      sortie_val_mesure :  INT;     //    Valeur de mesure correspondante
END_VAR

VAR
      valeurs_mesure       :  ARRAY[0..LIMITE] OF INT := 8(0);
      mémoire_résultat     :  ARRAY[0..LIMITE] OF
         STRUCT
            racine_carrée  :  INT;
            carré          :  INT;
         END_STRUCT;
      index                :  INT   :=    0;
      anc_val              :  BOOL  :=    TRUE;
      anc_tri              :  BOOL  :=    TRUE;
      anc_sélection        :  BOOL  :=    TRUE;
      adresse              :  INT   :=    0;    //Adresse de sortie convertie
      instance_calcul      :  CALCUL;           //Déclaration de l'instance locale
END_VAR
```

Figure 2-8 *Section de déclaration du bloc fonctionnel SAISIE*

Section des instructions

La section des instructions se divise en trois parties

Saisie des valeurs de mesure

Si le paramètre d'entrée « nouv_val » est différent de « anc_val », une nouvelle valeur de mesure est inscrite dans la mémoire circulante.

Démarrage du tri et du calcul

Le tri et le calcul sont démarrés par l'appel du bloc fonctionnel CALCUL, si le paramètre d'entrée « nouv_tri » est différent de « anc_tri ».

Evaluation du codage et préparation de la sortie

La saisie du codage s'effectue par mots : d'après les conventions appliquées dans SIMATIC, ceci signifie que les 8 bits de poids fort du mot d'entrée sont attribués au groupe de commutateurs du haut (octet 0) et les 8 bits de poids faible au groupe de commutateurs du bas (octet 1). La figure 2-9 indique où se trouvent les commutateurs sur lesquels vous devez effectuer le codage :

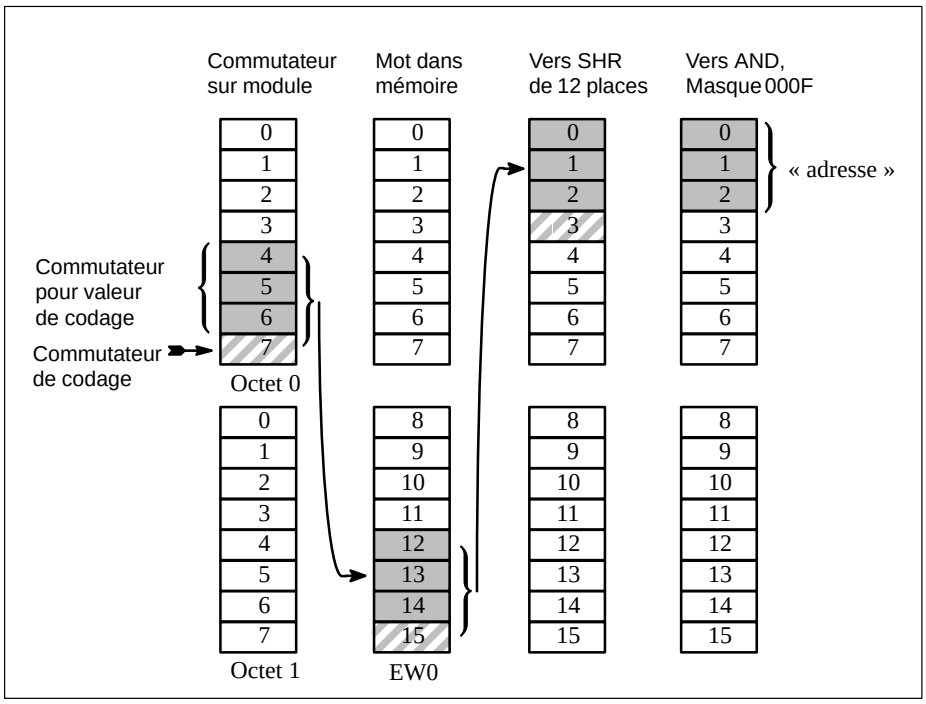


Figure 2-9 *Evaluation du codage*

Calcul de l'adresse

La figure 2-9 montre également le calcul de l'adresse : les bits 12 à 14 du mot d'entrée EW0 contiennent le codage qui est validé lorsqu'un front est détecté sur le commutateur de codage (bit 15). Le décalage vers la droite à l'aide de la fonction standard SHR et le masquage des bits significatifs à l'aide d'un masque AND permet de déterminer l'« adresse ».

C'est avec cette adresse que les composantes du tableau (résultat du calcul et valeur de mesure correspondante) sont inscrites dans les paramètres de sortie. C'est le « choix_fonction » qui détermine s'il s'agit de la racine carrée ou du carré.

Un front est détecté sur le commutateur de codage lorsque « nouv_val » est différent de « anc_val ».

Organigramme du FB SAISIE

La figure 2-10 représente l'algorithme sous forme d'organigramme :

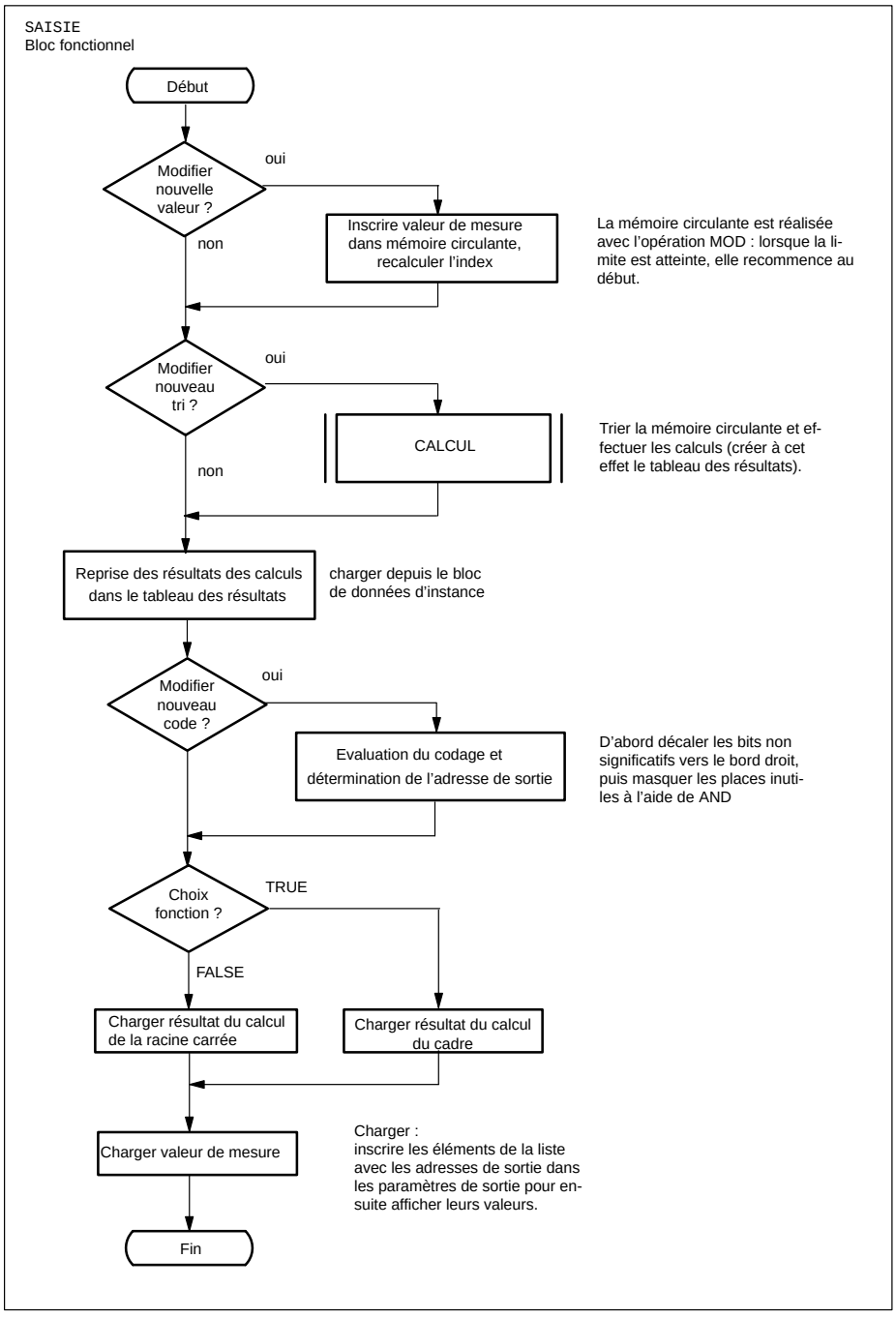


Figure 2-10 *Algorithme de saisie des valeurs de mesure*

Section des instructions du FB SAISIE

La figure 2-11 correspond à la formulation de l'organigramme représenté à la figure 2-10, c'est-à-dire à la **section des instructions** du bloc de code.

```
BEGIN

(* 1ère partie :  Saisie des valeurs de mesure*****************************************
                  La modification de "nouv_val" entraîne la saisie de la valeur de mesure.
                  L'opération MOD permet de réaliser une mémoire circulante
                  pour les valeurs de mesure.*)

IF nouv_val <> anc_val THEN
      index               := index MOD NOMBRE;
      valeurs_mesure[index]:= entrée_val_mesure;
      index               := index + 1;
END_IF;
anc_val  := nouv_val;

(* 2ème partie :  Démarrage du tri et du calcul****************************************
                  La modification de "nouv_tri" entraîne le démarrage du tri de la mémoire
                  circulante et l'exécution des calculs avec les valeurs de mesure. Les
                  résultats sont enregistrés dans un nouveau tableau, "mémoire_calcul".*)

IF nouv_tri <> anc_tri THEN
      index       := 0; //initialiser index de la mémoire circulante
      instance_calcul(mémoire_tri := valeurs_mesure); //Appel du FB CALCUL
END_IF;
anc_tri           := nouv_tri;

mémoire_résultat  := instance_calcul.mémoire_calcul; //Carré et racine carrée

(* 3ème partie    :  Evaluation du codage et préparation de la sortie :******************
                     La modification de la "nouv_sélection" fournit le codage pour
                     l'adressage de l'élément du tableau pour la sortie : les bits
                     significatifs de "sélection" sont masqués et convertis en entiers.
                     Selon la position du commutateur "choix_fonction", c'est la "racine
                     carrée" ou le "carré" qui sont préparés pour la sortie. *)

IF nouv_sélection <> anc_sélection THEN
      adresse        := WORD_TO_INT(SHR(IN := sélection, N := 12) AND 16#0007);
END_IF;
anc_sélection  :=    nouv_sélection;

IF choix_fonction THEN
      sortie_résultat   :=    mémoire_résultat[adresse].carré;
ELSE
      sortie_résultat   :=    mémoire_résultat[adresse].racine_carrée;
END_IF;


sortie_val_mesure :=    valeurs_mesure[adresse]; //Affichage de la valeur de mesure

END_FUNCTION_BLOCK
```

Figure 2-11 *Section des instructions du bloc fonctionnel SAISIE*

2.6 Programmation du bloc fonctionnel CALCUL

Section de déclaration de CALCUL

La section de déclaration de ce bloc comporte les parties suivantes :

- déclaration des constantes : entre CONST et END_CONST
- paramètres d'entrée/sortie : entre VAR_IN_OUT et END_VAR
- paramètres de sortie : entre VAR_OUTPUT et END_VAR
- déclaration des variables temporaires : entre VAR_TEMP et END_VAR

```
FUNCTION_BLOCK CALCUL

(*******************************************************************************
      1ère partie :     Tri de la mémoire circulante avec les valeurs de mesure
      2ème partie :     Démarrage du calcul des résultats
*******************************************************************************)
```

Figure 2-12 *En-tête du bloc fonctionnel CALCUL*

```
CONST
      LIMITE               := 7;
END_CONST

VAR_IN_OUT
      mémoire_tri          :  ARRAY[0..LIMITE] OF INT;
END_VAR

VAR_OUTPUT
      mémoire_calcul       :  ARRAY[0..LIMITE] OF
         STRUCT
            racine_carrée  :  INT;
            carré          :  INT;
         END_STRUCT;
END_VAR

VAR_TEMP
      échanger             :  BOOL;
      index, aux           :  INT;
      valeur_r, résultat_r :  REAL;
END_VAR
```

Figure 2-13 *Section de déclaration du bloc fonctionnel CALCUL*

Déroulement de la fonction

Le paramètre d'entrée/sortie « mémoire_tri » est combiné avec la mémoire circulante « valeurs_mesure », c'est-à-dire que le contenu initial de la mémoire est écrasé par les valeurs de mesure triées.

Le nouveau tableau « mémoire_calcul » est créé comme paramètre de sortie pour les résultats de calcul. La structure de ses composantes est telle que pour chaque valeur de mesure elles comportent la racine carrée et le carré.

La figure 2-14 illustre le lien entre les tableaux décrits.

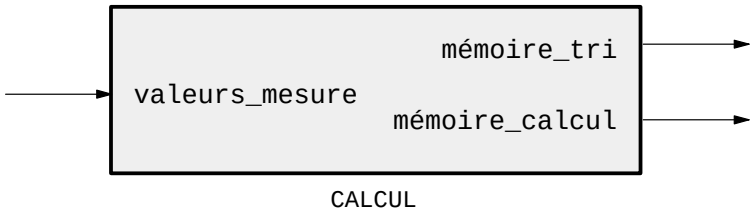


Figure 2-14 *Interface du FB CALCUL*

Cette interface représente la base de l’échange de données pour le traitement des valeurs de mesure. Les valeurs sont enregistrées dans le bloc de données d’instance SAISIE_DONNEES, puisqu’une instance locale a été créée pour le FB CALCUL dans le FB SAISIE.

Comment proocéder

Les valeurs de mesure sont tout d’abord triées dans la mémoire circulante, puis les calculs sont effectués :

- Méthode de l’algorithme de tri

Pour effectuer le tri de la mémoire circulante, la méthode utilisée est celle de l’échange permanent des valeurs où deux valeurs consécutives sont comparées entre elles et échangées jusqu’à ce que l’ordre de tri soit atteint. La mémoire utilisée est le paramètre d’entrée/sortie « mémoire_tri ».

- Démarrage du calcul

Lorsque le tri est réalisé, une boucle appelle les fonctions CARRE pour l’élévation au carré et SQRT pour le calcul de la racine. Leurs résultats sont enregistrés dans le tableau structuré « mémoire_calcul ».

Organigramme du FB CALCUL

La figure 2-15 représente l'algorithme sous forme d'organigramme :

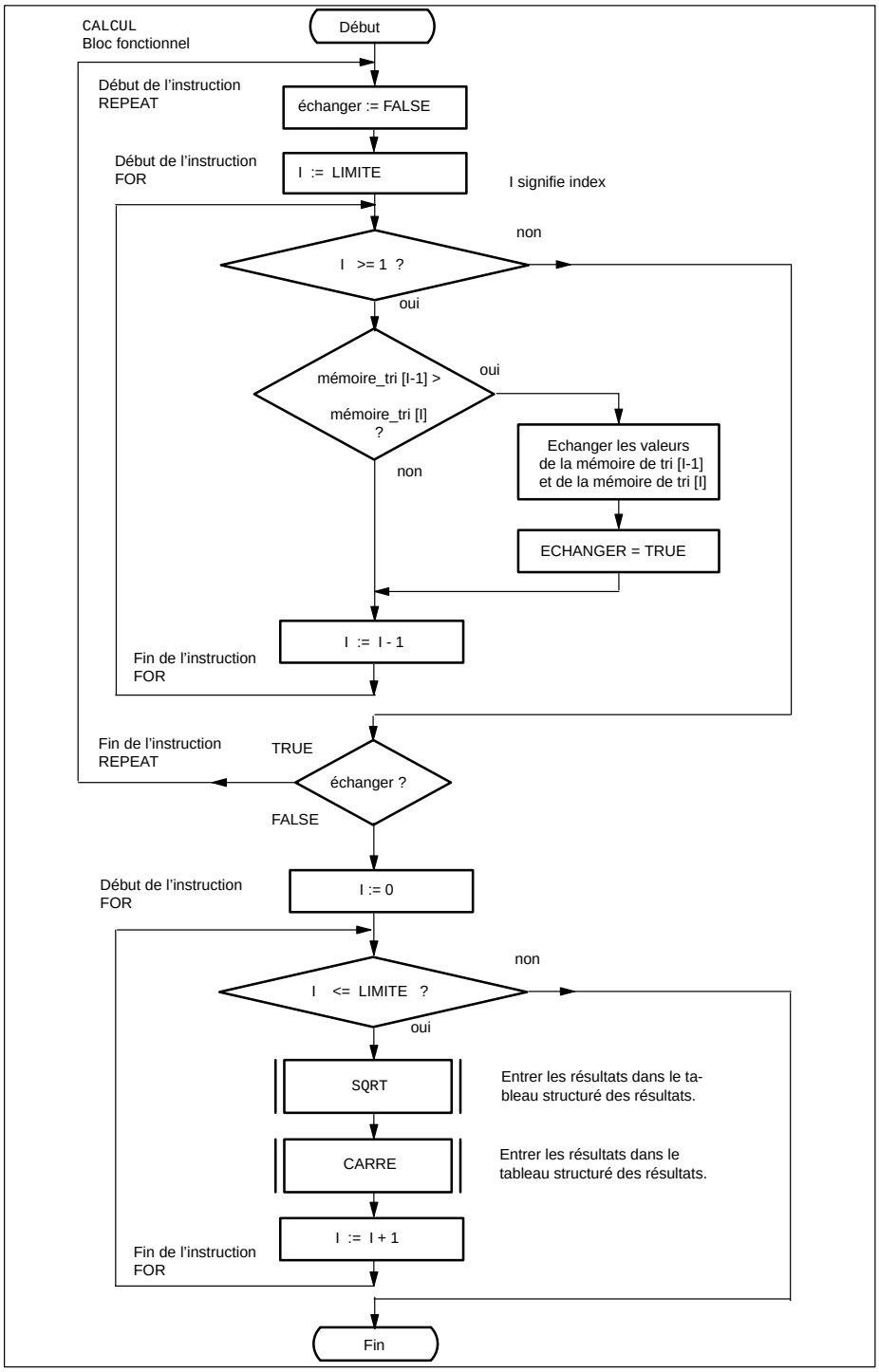


Figure 2-15 *Algorithme d'évaluation des valeurs de mesure*

Section des instructions du FB CALCUL

La figure 2-16 correspond à la formulation en SCL de l'organigramme représenté à la figure 2-15, c'est-à-dire à la **section des instructions** du bloc de code :

```
BEGIN

(* 1ère partie    Tri :*****************************************************************
                  procédé de "tri par échange de paires de clés" : échanger les valeurs par
                  paires jusqu'à ce que la mémoire des valeurs de mesure soit triée. *)

REPEAT
      échanger    := FALSE;

      FOR index   := LIMITE TO 1 BY -1 DO
         IF mémoire_tri[index-1] > mémoire_tri[index] THEN
            aux                     :=    mémoire_tri[index];
            mémoire_tri[index]      :=    mémoire_tri[index-1];
            mémoire_tri[index-1]    :=    aux;
            échanger                :=    TRUE;
         END_IF;
      END_FOR;

UNTIL NOT échanger
END_REPEAT;


(* 2ème partie    Calcul :*******************************************************************
                  Calcul de la racine carrée à l'aide de la fonction standard SQRT
                  et du carré à l'aide de la fonction CARRE. *)

FOR index         := 0 TO LIMITE BY 1 DO
      valeur_r    := INT_TO_REAL(mémoire_tri[index]);
      résultat_r  :=        SQRT(valeur_r);
      mémoire_calcul[index].racine_carrée :=    REAL_TO_INT(résultat_r);
      mémoire_calcul[index].carré         :=    CARRE(mémoire_tri[index]);
END_FOR;

END_FUNCTION_BLOCK
```

Figure 2-16 *Section des instructions du bloc fonctionnel CALCUL*

2.7 Programmation de la fonction CARRE

Comment procéder Il faut avant tout vérifier si la valeur d’entrée pour laquelle le résultat dépasserait la plage des valeurs des entiers autorisée est atteinte. Si tel est le cas, c’est la valeur maximale des entiers qui est fournie. Sinon c’est le calcul du carré qui est réalisé. Le résultat est donné comme valeur de la fonction.

```
FUNCTION CARRE : INT

(*******************************************************************************
Cette fonction fournit comme valeur le carré de la valeur d’entrée ou en cas de débordement
haut, la valeur maximale pouvant être représentée par un entier.
*******************************************************************************)

VAR_INPUT
      valeur   :  INT;
END_VAR

BEGIN
IF valeur <= 181 THEN
      CARRE    :=    valeur * valeur;     // Calcul de la valeur de la fonction
ELSE
      CARRE    :=    32_767;              // Fournir la valeur maximale en cas de
                                          // débordement haut
END_IF;

END_FUNCTION
```

Figure 2-17 *Fonction CARRE*

2.8 Données de test

Conditions

Pour réaliser le test, un module d'entrée doit être enfiché à l'adresse 0 et un module de sortie à l'adresse 4 (voir figure 2-4).

Avant de commencer le test, les 8 commutateurs du groupe supérieur doivent être positionnés vers la gauche (« 0 ») et les 8 commutateurs du groupe inférieur vers la droite (« 1 »).

Les valeurs initiales des variables étant également testées, vous devez effectuer un nouveau chargement des blocs dans la CPU.

Etapes du test

Les étapes du test sont décrites dans le tableau 2-5.

Tableau 2-5 Etapes du test

Test	Action	Résultat
1	Sélectionnez le codage « 111 » (E 0.4, E 0.5 et E 0.6), puis validez-le à l'aide du commutateur de codage (E 0.7).	Toutes les sorties de l'affichage (octet de poids faible) sont activées et les diodes sont allumées.
2	Affichez la racine carrée correspondante en positionnant le commutateur de sortie (E 0.3) sur « 1 ».	Les organes d'affichage à la sortie correspondent à la valeur binaire « 10000 » (=16).
3	Affichez le carré correspondant en positionnant le commutateur de fonction (E 0.2) sur « 1 ».	15 organes d'affichage s'allument à la sortie. Ceci signale un débordement, car 255 x 255 fournit une valeur trop grande pour la plage des entiers autorisée.
4a	Positionnez à nouveau le commutateur de sortie (E 0.3) sur « 0 ».	La valeur de mesure est à nouveau affichée : tous les organes d'affichage aux sorties de l'octet de sortie de poids faible affichent « 1 ».
4b	Sélectionnez comme nouvelle valeur de mesure à l'entrée la valeur 3, c'est-à-dire la valeur binaire « 11 ».	La sortie ne se modifie pas encore.
5a	Visualisez la validation de la valeur de mesure : sélectionnez le codage « 000 », puis validez-le à l'aide du commutateur de codage (E 0.7), afin de pouvoir ultérieurement visualiser la suite de la saisie.	La sortie affiche « 0 », c'est-à-dire qu'aucun organe d'affichage n'est allumé.
5b	Inversez le commutateur d'entrée « Entrée 0.0 » (E 0.0). La valeur sélectionnée à la 4ème étape de test est validée.	La sortie affiche la valeur de mesure 3 qui correspond à la valeur binaire « 11 ».
6	Démarrez le tri et le calcul en inversant le commutateur de tri (E 0.1).	La sortie affiche à nouveau 0, car la procédure de tri a décalé la valeur de mesure vers le haut du tableau.
7	Affichez la valeur de mesure après le tri : sélectionnez le codage « 110 » (E 0.6 = 1, E 0.5 = 1, E 0.4 = 0 pour EB0, ce qui correspond aux bit 14, bit 13, bit 12 de EW 0) puis validez-le en inversant le commutateur de codage.	La sortie affiche à présent de nouveau la valeur « 11 », puisqu'il s'agit de la seconde valeur la plus élevée du tableau.
8a	Affichez les résultats correspondants : l'inversion du commutateur de sortie (E 0.3) permet d'afficher le carré de la valeur de mesure de l'étape 7.	La valeur de sortie 9 qui correspond à la valeur binaire « 1001 » s'affiche.
8b	En inversant le commutateur de fonction (E 0.2), vous obtenez également l'affichage de la racine.	La valeur de sortie 2 qui correspond à la valeur binaire « 10 » s'affiche.

Test complémentaire

En vous inspirant des explications sur les commutateurs de commande figurant dans le tableau 2-6 et des exemples de test du tableau 2-7 vous pouvez définir vous-même les étapes que vous souhaitez tester :

- L'entrée s'effectue avec les commutateurs : les 8 du haut sont destinés aux commandes, les 8 du bas servent à sélectionner la valeur de mesure.
- La sortie a lieu sur les organes d'affichage : le groupe du haut affiche l'octet de sortie de poids fort, celui du bas l'octet de sortie de poids faible.

Tableau 2-6 Commutateurs de commande

Commutateurs de commande	Nom	Explication
Voie 0	Commutateur de saisie	Commutation pour la validation des valeurs de mesure
Voie 1	Commutateur de tri	Commutation pour le tri/calcul
Voie 2	Commutateur de fonction	Commutateur à gauche (« 0 ») : racine carrée, Commutateur à droite (« 1 ») : carré
Voie 3	Commutateur de sortie	Commutateur à gauche (« 0 ») : valeur de mesure, Commutateur à droite (« 1 ») : résultat
Voie 4	Codage	Adresse de sortie bit 0
Voie 5	Codage	Adresse de sortie bit 1
Voie 6	Codage	Adresse de sortie bit 2
Voie 7	Codage	Commutation pour la validation du codage

Le tableau 2-7 contient 8 valeurs de mesure exemple classées déjà dans le bon ordre.

Saisissez les valeurs dans l'ordre que vous voulez. Choisissez à cet effet la combinaison binaire voulue et validez la valeur en actionnant le commutateur de saisie. Une fois que toutes les valeurs ont été saisies, vous démarrez le tri et le calcul en actionnant le commutateur de tri. Vous avez ensuite la possibilité de visualiser les valeurs de mesure triées ou alors les résultats – racine carrée ou carré.

Tableau 2-7 Exemple de test pour la racine carrée et le carré

Valeur de mesure	Racine carrée	Carré
0000 0001 = 1	0, 0000 0001 = 1	0000 0000, 0000 0001 = 1
0000 0011 = 3	0, 0000 0010 = 2	0000 0000, 0000 1001 = 9
0000 0111 = 7	0, 0000 0011 = 3	0000 0000, 0011 0001 = 49
0000 1111 = 15	0, 0000 0100 = 4	0000 0000, 1110 0001 = 225
0001 1111 = 31	0, 0000 0110 = 6	0000 0011, 1100 0001 = 961
0011 1111 = 63	0, 0000 1000 = 8	0000 1111, 1000 0001 = 3969
0111 1111 = 127	0, 0000 1011 = 11	0011 1111, 0000 0001 = 16129
1111 1111 = 255	0, 0001 0000 = 16	0111 111, 1111 1111 = indication de débordement de plage

Deuxième partie : Utilisation et test

Installation du logiciel SCL

3

Présentation

Un programme d'installation vous propose des menus pour installer le logiciel SCL. Vous devez le démarrer selon la procédure habituelle d'installation des logiciels sous Windows 95.

Conditions requises

L'installation du logiciel SCL requiert :

- une console de programmation ou un PC sur lesquels le logiciel de base STEP 7 est déjà installé
  - un processeur 80486 (ou supérieur) et
  - une mémoire RAM de 16 Mo
- un écran couleur, un clavier et une souris compatibles avec Windows 95 de Microsoft
- un disque dur possédant une capacité libre d'au moins 78 Mo
  (10 Mo pour les données utilisateur, 60 Mo pour les copies sur disque, 8 Mo pour le logiciel optionnel SCL)
- une capacité libre de 1 Mo au minimum sur l'unité C: pour le programme d'installation (les fichiers d'installation sont effacés une fois celle-ci terminée)
- le système d'exploitation Windows 95.
- une interface MPI entre la PG ou le PC et l'AP :
  - soit un câble PC/MPI que vous connectez à l'interface de communication de la PG ou du PC,
  - soit une carte MPI que vous installez dans votre PG ou PC. Dans certaines consoles, l'interface MPI est déjà installée.

Structure du chapitre

3.1 Autorisation / licence d'utilisation

Introduction

L'utilisation du logiciel SCL requiert une autorisation spécifique pour le produit (licence d'utilisation). Le logiciel ainsi protégé ne peut être utilisé que si l'autorisation requise pour le programme ou le logiciel est installée sur le disque dur de la PG ou du PC correspondant.

Disquette d'autorisation

Vous devez posséder la disquette d'autorisation protégée fournie à la livraison. Elle contient l'autorisation ainsi que le programme AUTHORS nécessaire à l'afficher, l'installer et la désinstaller.

Le nombre d'autorisations possibles est défini sur cette disquette par un compteur qui est décrémenté de 1 pour chaque nouvelle autorisation utilisée. Lorsque le compteur est à zéro, il n'est plus possible d'installer d'autorisation avec cette disquette.

Vous trouverez des informations détaillées, de même que les règles relatives aux autorisations dans le guide de l'utilisateur **/231/**.

Avertissement

Tenez compte des instructions données dans le fichier LISEZMOI.TXT sur la disquette d'autorisation. En cas de leur non-respect, l'autorisation risque d'être perdue irrémédiablement.

Installation de l'autorisation à la mise en service

Vous devez fournir l'autorisation lorsque vous y êtes sollicité durant la première installation de votre logiciel par un message correspondant. Procédez de la manière suivante :

1. Introduisez la disquette d'autorisation dans le lecteur lorsque vous y êtes sollicité par un message.
2. Confirmez ensuite ce message.

L'autorisation est alors installée sur une unité physique (ceci signifie que votre PG ou PC ont en mémoire que vous êtes titulaire de l'autorisation).

Installation ultérieure de l'autorisation

Si vous démarrez le logiciel SCL sans qu'une autorisation soit installée, un message vous en informe. Pour l'installer ultérieurement, appelez le programme AUTHORS figurant sur la disquette d'autorisation. Il vous permet d'afficher, d'installer et de désinstaller les autorisations à l'aide de menus.

Nota

Pour l'installation de l'autorisation pour SCL, indiquez toujours C: comme unité cible.

Désinstallation de l'autorisation

Si une nouvelle autorisation est requise, par exemple dans le cas où vous voulez formater l'unité sur laquelle elle se trouve, vous devez préalablement la « sauvegarder ». A cet effet, il vous faut la disquette d'autorisation originale.

Pour rapatrier l'autorisation sur la disquette originale, procédez de la manière suivante :

1. Introduisez la disquette d'autorisation originale dans le lecteur de 3,5 pouces.
2. Appelez le programme AUTHORS.EXE à partir de ce lecteur.
3. Choisissez la commande **Autorisation ▸ Désinstaller**.
4. Dans la boîte de dialogue qui s'ouvre, indiquez l'unité sur laquelle se trouve l'autorisation, puis confirmez. La liste de toutes les autorisations détectées sur l'unité correspondante s'affiche.
5. Sélectionnez l'autorisation que vous souhaitez désinstaller, puis confirmez. Si la désinstallation est réussie, le message suivant s'affiche « **L'autorisation** <Nom> **a été retirée avec succès du lecteur** <X:>**.** »
6. Confirmez ce message.

   Vous obtenez ensuite à nouveau la liste des autorisations restantes sur l'unité correspondante. Fermez cette boîte de dialogue si vous ne souhaitez pas désinstaller d'autre autorisation.

Vous pouvez ensuite à nouveau utiliser cette disquette pour installer une autorisation.

En cas de défaillance de votre disque dur...

Si une défaillance se produit sur votre disque dur avant que vous n'ayez désinstallé l'autorisation, veuillez contacter le service SIEMENS compétant.

3.2 Installation / Désinstallation du logiciel SCL

Présentation

Un programme d'installation permet d'exécuter automatiquement l'installation de SCL. Durant toute cette procédure l'on vous demandera de répondre étape après étape aux messages affichés à l'écran.

Conditions

Avant de pouvoir réaliser l'installation de SCL, Windows 95 doit être démarré et le logiciel STEP 7 doit être chargé.

Démarrage du programme d'installation

Procédez de la manière suivante :

1. Dans Windows 95, démarrez le dialogue d'installation d'un logiciel en effectuant un double-clic sur l'icône « Ajout/Supression de programmes » du « Panneau de configuration ».
2. Cliquez sur « Installer ».
3. Introduisez le support de données (disquette 1 ou CD-ROM), puis cliquez sur « Suivant ». Windows 95 recherche alors lui-même le programme d'installation SETUP.EXE.
4. Suivez une à une les instructions données par le programme d'installation.

Le programme vous guide pas à pas tout au long du processus d'installation. Vous pouvez à tout instant passer à l'étape précédente ou suivante.

Au cas où une version de SCL est déjà installée

Si le programme d'installation détecte une version de SCL déjà installée sur l'outil de développement, un message vous en informe et les possibilités suivantes s'offrent à vous :

- abandonner l'installation (pour désinstaller l'ancienne version de SCL sous Windows 95 puis redémarrer l'installation) ou
- poursuivre l'installation et écraser l'ancienne version par la nouvelle.

Avant de procéder à une installation, nous vous recommandons dans tous les cas de désinstaller une éventuelle ancienne version. Le simple écrasement d'une ancienne version a pour inconvénient de conserver certaines parties lors d'une désinstallation ultérieure.

Durant la procédure d'installation, vous serez amené à répondre à des questions ou à choisir des options dans des boîtes de dialogue. Pour y parvenir le plus rapidement possible, nous vous recommandons de tenir compte des informations suivantes.

Désinstallation

Utilisez le processus habituel de désinstallation de Windows 95 :

1. Sous Windows 95, démarrez le dialogue d'installation d'un logiciel en effectuant un double-clic sur l'icône « Ajout/Supression de programmes » dans « Panneau de configuration ».
2. Sélectionnez l'entrée STEP 7 dans la liste des logiciels installés, puis confirmez en cliquant sur le bouton de suppression du logiciel.
3. Si une boîte de dialogue vous demandant de confirmer la suppression du fichier validé apparaît, cliquez sur le bouton « Non » en cas de doute.

Niveau d'installation

Toutes les langues disponibles et tous les exemples requièrent environ 8 Mo de mémoire.

Autorisation

La vérification de la présence d'une autorisation sur le disque dur est réalisée durant l'installation. Si tel n'est pas le cas, un message vous informe que le logiciel ne peut être utilisé qu'avec une autorisation. Vous avez alors la possibilité de la fournir immédiatement ou alors de poursuivre l'installation et de la fournir ultérieurement.

Dans le premier cas, il vous suffit d'introduire la disquette d'autorisation dans le lecteur lorsque l'on vous la demandera, puis de confirmer. Des informations à ce sujet vous sont données au paragraphe 3.1.

Installation réussie

Si l'installation est réussie, un message correspondant vous en informe à l'écran.

Erreur durant l'installation

Les erreurs suivantes provoquent l'abandon de l'installation :

- Si une erreur d'initialisation se produit immédiatement après le démarrage de l'installation, il est probable que le programme SETUP.EXE n'a pas été démarré sous Windows 95.
- La mémoire est insuffisante : il vous faut au minimum 8 Mo de capacité mémoire libre sur votre disque dur.
- Disquette défectueuse : si l'une des disquettes d'installation est défectueuse, veuillez contacter votre agence Siemens.
- Erreur de manipulation : renouvelez l'installation en suivant scrupuleusement les instructions.

Utilisation de SCL

4

Présentation

Le présent chapitre vous familiarise avec l'utilisation de SCL et décrit l'interface utilisateur de l'éditeur de SCL.

Structure du chapitre

4.1 Démarrage du logiciel SCL

Démarrage depuis Windows

Après avoir installé le logiciel SCL sur votre PG, vous pouvez démarrer SCL en utilisant le bouton « Démarrer » de la barre des tâches de Windows 95 (via l'option « SIMATIC / STEP 7 »)

Démarrage depuis SIMATIC Manager

La manière la plus rapide de démarrer SCL consiste à positionner le curseur sur une source SCL et d'effectuer un double-clic. Pour plus d'informations reportez-vous au guide de l'utilisateur **/231/**.

La figure 4-1 montre la fenêtre SCL après le démarrage du logiciel SCL.

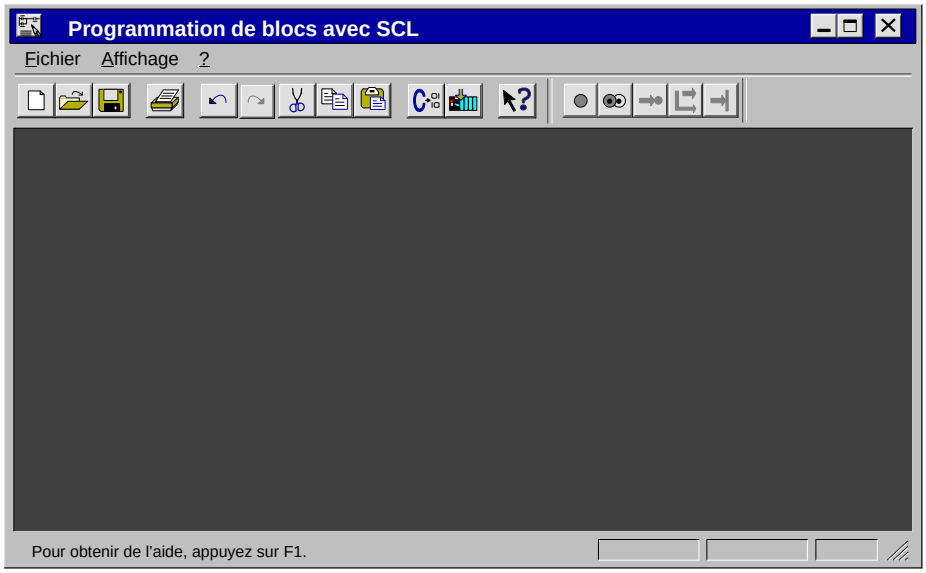


Figure 4-1 *Fenêtre de SCL*

Nota

Le guide de l'utilisateur de Windows 95 ou le didacticiel de l'aide en ligne correspondant vous donneront des informations détaillées sur l'utilisation standard et sur les options de Windows 95.

4.2 Adaptation de l’interface utilisateur

Présentation

Comme les autres fenêtres de STEP 7, les fenêtres de SCL comportent les composantes standard suivantes, voir figure 4-2 :

- **Barre de titre :**
  elle comporte le titre de la fenêtre ainsi que les boutons de commande de la fenêtre.
- **Barre des menus :**
  elle comporte tous les menus disponibles dans la fenêtre.
- **Barre d’outils :**
  elle contient tous les boutons permettant d’exécuter rapidement les commandes les plus fréquentes.
- **Zone de travail :**
  elle comporte les diverses fenêtre d’édition, de compilation ou de test.
- **Barre d’état :**
  elle affiche l’état de l’objet sélectionné ainsi que des informations supplémentaires.

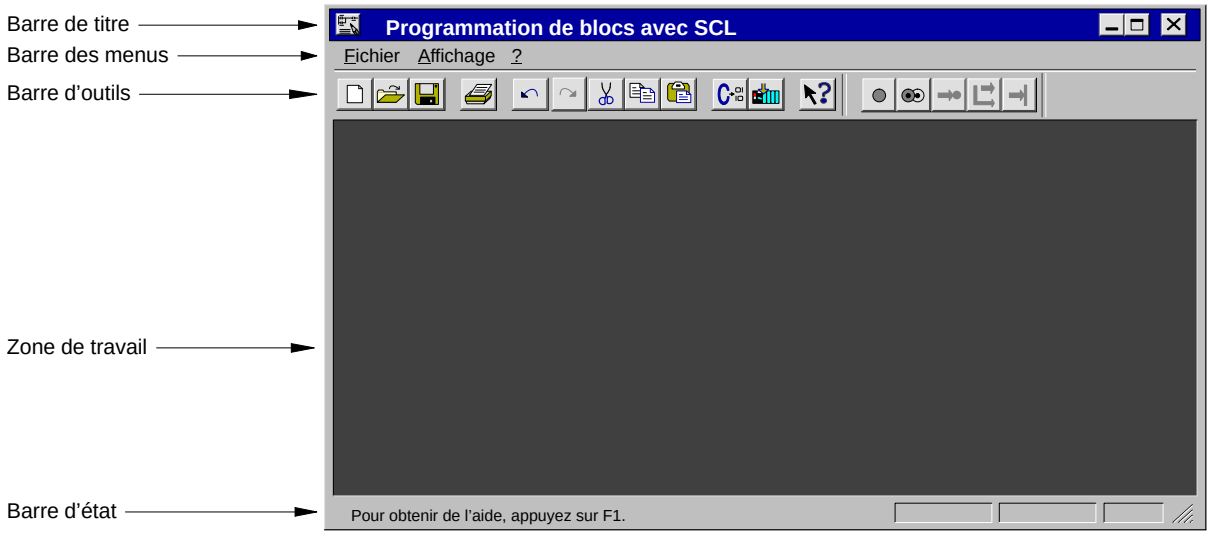


Figure 4-2 *Composantes de la fenêtre SCL*

Modification des composantes

Vous avez la possibilité d'adapter les éléments suivants à vos exigences particulières :

- © affichage de la barre d'outils
- © affichage de la barre des points d'arrêt
- © affichage de la barre d'état

Adaptation de la barre d'outils

Vous avez la possibilité d'afficher ou de masquer la barre d'outils. Il vous suffit d'activer ou de désactiver la commande **Affichage ▸ Barre d'outils**. Si la commande est activée, elle est cochée.

Adaptation de la barre des points d'arrêt

Vous pouvez également afficher ou masquer la barre des points d'arrêt. Il vous suffit d'activer ou de désactiver la commande **Affichage ▸ Barre des points d'arrêt**. Si la commande est activée, elle est cochée.

Adaptation de la barre d'état

Vous avez également la possibilité d'afficher ou de masquer la barre d'état. Il vous suffit d'activer ou de désactiver la commande **Affichage ▸ Barre d'état**. Si la commande est activée, elle est cochée.

Adaptation de l'environnement

L'éditeur et le compilateur vous offrent des options qui ont pour but de vous faciliter le travail.

- © options de création de blocs
- © options du compilateur
- © options de l'éditeur

Création du bloc

Vous pouvez, par exemple, définir si les blocs existants doivent être écrasés durant la compilation ou pas. Choisissez à cet effet la commande **Outils ▸ Paramètres** et cliquez sur l'onglet « Création du bloc » dans la boîte de dialogue « Paramètres ». Une description détaillée des options figure au paragraphe 5.5.

Adaptation du compilateur

Vous pouvez adapter la procédure de compilation à vos exigences particulières. Une description détaillée des options figure au paragraphe 5.5.

Choisissez à cet effet la commande **Outils ▸ Paramètres** et cliquez sur l'onglet « Compilateur » dans la boîte de dialogue « Paramètres ».

Adaptation de l'éditeur

Vous pouvez sélectionner la largeur de tabulation, l'enregistrement avant compilation, l'affichage des numéros de ligne et bien d'autres options. Choisissez à cet effet la commande **Outils ▸ Paramètres** et cliquez sur l'onglet « Editeur » dans la boîte de dialogue « Paramètres ».

4.3 Utilisation de l’éditeur de SCL

Présentation

Une source SCL est principalement composée de texte au kilomètre. Des fonctions d’édition spécialement développées pour SCL vous assistent lors de la saisie du texte.

Fenêtre d’édition

Vous entrez l’objet source pour votre programme utilisateur par saisie sur le clavier. Vous avez la possibilité d’ouvrir plusieurs fenêtres du même ou d’un autre objet source. La commande Fenêtre vous permet d’en modifier la disposition.

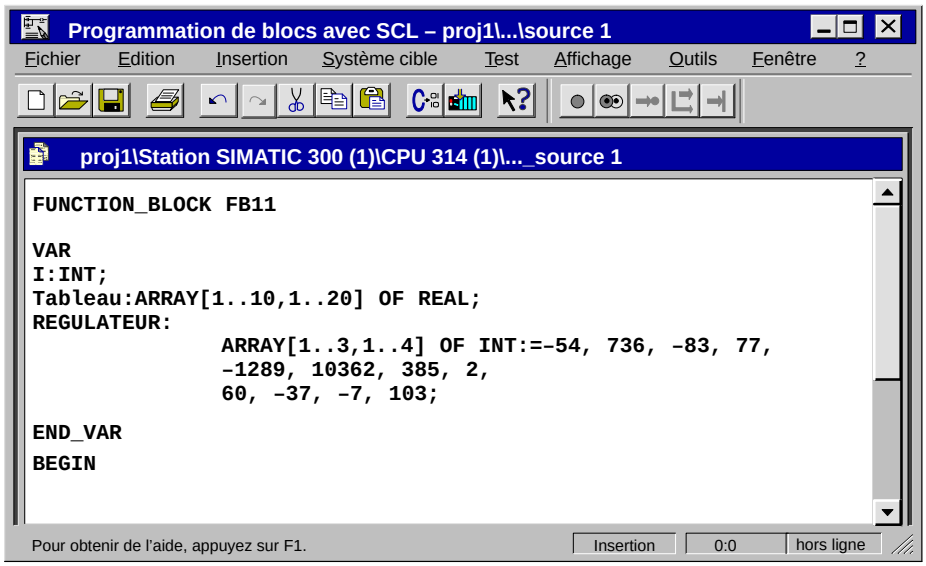


Figure 4-3 *Fenêtre d’édition de SCL*

Sélection de texte

Pour sélectionner une zone de texte dans SCL, vous positionnez le curseur au début de la zone à sélectionner, puis tout en maintenant le bouton de la souris enfoncé, vous faites glisser la souris sur la zone correspondante.

Vous avez également la possibilité de sélectionner

- l’ensemble du texte d’une source en choisissant la commande **Edition ▸ Sélectionner tout**,
- un mot en cliquant deux fois sur celui-ci ou
- une ligne entière en cliquant 3 fois dessus.

Rechercher et remplacer

La commande **Edition ▸ Rechercher /Remplacer** ouvre une boîte de dialogue dans laquelle vous pouvez rechercher une chaîne de caractères en vue de la remplacer par d’autres objets de texte.

Insertion de modèles

En vous permettant de respecter plus aisément les règles de syntaxe, l'insertion de modèles contribue à une programmation plus efficace. Dans SCL, vous pouvez :

- insérer des modèles de bloc, en choisissant la commande **Insertion ▸ Modèle de bloc**.
- insérer des structures de contrôle, en choisissant la commande **Insertion ▸ Structure de contrôle**.

Couper, copier, coller, effacer

Le menu **Edition** vous propose comme à l'accoutumée les commandes vous permettant de couper, copier, coller et effacer des objets de texte.

Vous avez également la possibilité de déplacer ou de copier des objets en les coupant/collant à l'aide de la souris (Drag&Drop).

Aller à

La commande de menu **Edition ▸ Aller à** ouvre une boîte de dialogue. Vous pouvez y indiquer le numéro de la ligne à laquelle vous désirez vous rendre, puis valider votre choix par « OK ». Le curseur se placera alors au début de la ligne voulue.

Annuler, rétablir

La commande de menu **Edition ▸ Annuler** permet d'annuler la dernière action. Vous pouvez, par exemple, rétablir une ligne effacée. La commande **Edition ▸ Rétablir** permet de rétablir une action venant d'être annulée.

Programmation avec SCL

5

Présentation

Ce chapitre décrit les différentes étapes à respecter lors de la programmation, l'ordre donné ici n'étant qu'un ordre possible parmi d'autres.

Structure du chapitre

5.1 Créer un programme utilisateur avec SCL

Conditions préalables

Avant de créer un programme avec SCL, vous devriez réaliser les tâches suivantes :

1. Créez un projet dans SIMATIC Manager.
2. Dans SIMATIC Manager, affectez à chaque CPU une adresse de communication dans le réseau.
3. Configurez et paramétrez l'unité centrale et les modules de signaux.
4. Si vous souhaitez utiliser des mnémoniques pour les zones de mémoire de la CPU ou pour les désignations de blocs, vous devez créer une table de mnémoniques.

Table de mnémoniques

SCL aura recours à cette table lors de la compilation. Vous créez et remplissez la table des mnémoniques sous STEP 7.

Vous pouvez ensuite l'ouvrir depuis le SIMATIC Manager ou directement dans SCL grâce à la commande de menu **Outils ▸ Table de mnémoniques**.

Vous pouvez également importer et modifier des tables de mnémoniques sous forme de fichiers de texte (l'éditeur utilisé étant indifférent). Pour plus d'informations à ce sujet, reportez-vous au guide de l'utilisateur **/231/**.

Comment procéder

Pour créer un programme utilisateur dans SCL, vous créez d'abord une source SCL dans laquelle vous pouvez éditer un ou plusieurs blocs (OB, FB, FC, DB et UDT) que vous allez ensuite compiler à l'aide d'une procédure séquentielle. Après compilation, ces blocs figureront dans le classeur « Programme utilisateur » (<AP-off>, voir figure 5-1) du programme S7 correspondant, dans lequel est également enregistrée la source.

Vous avez la possibilité de créer et d'éditer la source SCL avec l'éditeur intégré ou avec un éditeur standard quelconque. Dans le second cas, vous devez importer les sources dans le projet avec SIMATIC Manager. Vous pouvez ensuite les ouvrir afin de les éditer ou de les compiler.

5.2 Créer et ouvrir une source SCL

Présentation

Les sources que vous créez avec SCL s'intègrent de la manière suivante dans la structure d'un programme S7 :

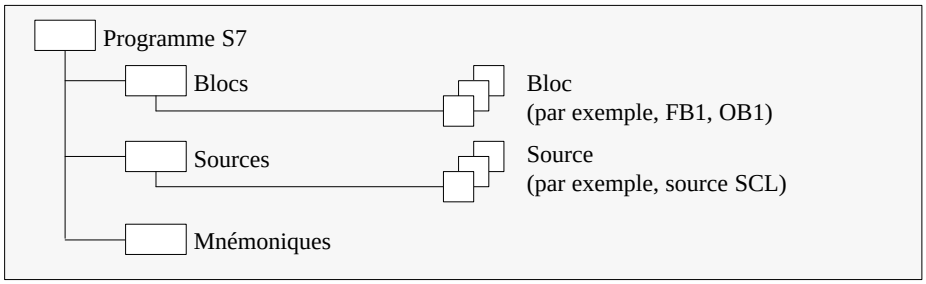


Figure 5-1 *Structure d'un programme S7 dans SIMATIC Manager*

Créer une source SCL

Pour créer une source pour SCL, procédez de la manière suivante :

1. Choisissez la commande **Fichier ▸ Nouveau** ou cliquez sur le bouton « Nouveau » dans la barre d'outils.
2. Sélectionnez dans la boîte de dialogue « Nouveau » le projet souhaité et le classeur des sources du programme S7 correspondant.
3. Ouvrez le classeur des sources et choisissez la commande **Insertion ▸ Logiciel S7 ▸ Source SCL**.
4. Sélectionnez la source et choisissez la commande **Edition ▸ Propriétés de l'objet**. Entrez le nom de l'objet source dans la page d'onglet « Fiche d'identité ». Vous disposez de 24 caractères au maximum. La distinction entre les majuscules et les minuscules joue un rôle dans les noms de sources.
5. Effectuez un double clic sur la source. Une fenêtre vide dans laquelle vous pouvez éditer la source SCL s'ouvre.

Ouvrir une source SCL

Vous pouvez ouvrir un objet source créé et enregistré avec SCL afin de l'éditer et de le compiler.

Procédez de la manière suivante :

1. Choisissez la commande **Fichier ▸ Ouvrir**, ou cliquez sur le bouton « Ouvrir ».
2. Sélectionnez le projet souhaité et le programme S7 correspondant dans la boîte de dialogue « Ouvrir ».
3. Assurez-vous que le filtre « Source SCL » a été choisi et sélectionnez le classeur des sources.
4. Toutes les sources SCL du programme S7 correspondant s'affichent dans la boîte de dialogue. Sélectionnez l'objet que vous souhaitez ouvrir, puis confirmez par « OK » ou effectuez un double-clic sur la source.

Vous pouvez ouvrir de la même manière les sources que vous avez créées avec un éditeur standard, après les avoir importées dans le projet à l'aide de SIMATIC Manager.

5.3 Saisir des déclarations, instructions et commentaires

Présentation

La saisie de sources SCL répond à des règles de syntaxe bien précises qui font partie intégrante de la *description du langage*. Elles sont énoncées en annexe.

proj1\Station SIMATIC 300 (1)\CPU 314 (1)\...\source 1

```
FUNCTION_BLOCK FB11

VAR
I:INT;
Tableau:ARRAY[1..10,1..20] OF REAL;
REGULATEUR:
             ARRAY[1..3,1..4] OF INT:=-54, 736, -83, 77,
             -1289, 10362, 385, 2,
             60, -37, -7, 103;

END_VAR
BEGIN
```

Figure 5-2 *Source SCL*

Règles

La saisie implique les respect des conventions suivantes :

- Vous pouvez formuler un nombre quelconque de blocs de code (FB, FC, OB), blocs de données (DB) et types de données utilisateur (UDT) dans une source SCL, la structure de chaque type de bloc étant particulière (voir chapitre 8).
- La distinction entre majuscules et minuscules n'intervient que pour les identificateurs symboliques (par exemple les noms de variables et les constantes littérales de type caractère).
- Les blocs appelés doivent précéder les blocs qui les appellent.
- Les types de données utilisateur (UDT) doivent précéder les blocs qui les utilisent.
- Les blocs de données globaux doivent précéder tous les blocs qui y accèdent.
- Appliquez les règles de format et de syntaxe énoncées à l'annexe A, *Description formelle du langage* du présent manuel.

5.4 Enregistrer et imprimer une source SCL

Enregistrer une source SCL

Dans SCL, l'enregistrement désigne l'enregistrement d'un fichier source. Les blocs sont créés durant la compilation du fichier source et sont quant à eux, enregistrés automatiquement dans le répertoire du programme correspondant.

Diverses possibilités s'offrent à vous pour enregistrer une source SCL :

- Choisissez la commande **Fichier ▸ Enregistrer**, ou cliquez sur le bouton « Enregistrer » dans la barre d'outils.

  La source SCL est mise à jour.

- Si vous souhaitez faire une copie du fichier SCL en cours, choisissez la commande **Fichier ▸ Enregistrer sous**. Dans la boîte de dialogue « Enregistrer sous » qui s'ouvre, vous pouvez indiquer le nom et le chemin de la copie.

- Si vous choisissez la commande **Fichier ▸ Fermer** sans avoir préalablement enregistré la source que vous avez modifiée, vous devrez spécifier si vous souhaitez enregistrer les modifications, les ignorer ou encore annuler la commande **Fermer**.

  Choisir la commande **Fichier ▸ Fermer** revient à cliquer sur le bouton « Fermer » dans la barre d'outils.

  Lorsque vous choisissez la commande **Fichier ▸ Quitter** sans que la version en cours des sources ouvertes ne soit enregistrée, un dialogue vous est proposé pour chaque source afin d'enregistrer ou d'ignorer les modifications.

Imprimer une source

Vous pouvez imprimer les blocs, déclarations et instructions d'une source SCL sur une imprimante que vous devez préalablement installer et configurer dans la gestion du système de Windows. Procédez de la manière suivante :

- Cliquez sur le bouton « Imprimer » dans la barre d'outils ou choisissez la commande **Fichier ▸ Imprimer**. Dans la boîte de dialogue qui s'ouvre, vous pouvez choisir diverses options d'impression, comme par exemple la zone à imprimer ou le nombre de copies.

- Confirmez par « OK », pour envoyer le document sur l'imprimante.

Définir la mise en page

La commande **Fichier ▸ Mise en page** vous permet de définir le format de page de votre document.

Définir les en-têtes et bas de page

Vous pouvez définir les en-têtes et bas de page du document à imprimer en choisissant la commande **Fichier ▸ En-têtes et bas de page** dans le gestionnaire de projets SIMATIC.

Aperçu avant impression

Avec la commande **Fichier ▸ Aperçu avant impression**, vous avez la possibilité de visualiser un document avant de l'imprimer. Son édition n'est cependant pas possible.

5.5 Procédure de compilation

Présentation

Avant de pouvoir exécuter ou tester votre programme, vous devez le compiler. En initiant le processus de compilation, vous activez un compilateur, qui possède les propriétés suivantes :

- Il fonctionne d'après le principe du traitement par lots, ce qui signifie qu'il considère une source SCL comme un ensemble. La compilation partielle (par exemple ligne par ligne) est impossible.
- Il vérifie la syntaxe de la source SCL, puis signale toutes les erreurs détectées durant la compilation.
- Il crée des blocs comportant des informations de déboguage si la compilation de la source est exempte d'erreur et si vous avez sélectionné l'option correspondante (voir ci-après). Vous devez choisir cette option d'information de déboguage pour chaque programme dont vous souhaitez tester le niveau de langage évolué.
- A chaque appel de bloc fonctionnel, il génère le bloc de données d'instance correspondant si ce dernier n'existe pas encore.

Sélectionner les options du compilateur

Vous avez la possibilité d'adapter la procédure de compilation à vos exigences particulières. Choisissez à cet effet la commande **Outils** ▸ **Paramètres**, puis cliquez sur l'onglet « Compilateur » dans la boîte de dialogue « Paramètres ». Vous pouvez activer ou désactiver les options proposées à l'aide de la souris.

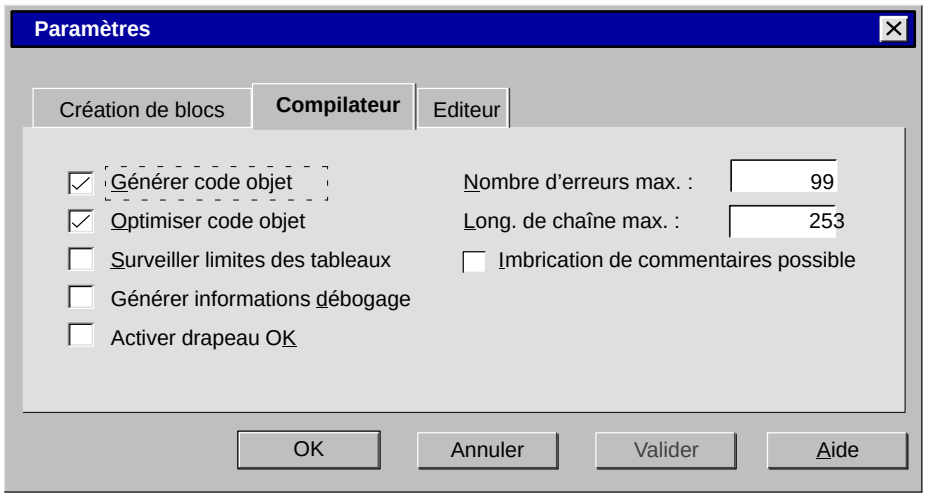


Figure 5-3 *Boîte de dialogue « Paramètres », onglet « Compilateur »*

Options

La signification des options est la suivante :

- **Nombre d'erreurs max.** : lorsque le nombre d'erreurs maximum indiqué est atteint, le compilateur abandonne la compilation de la source SCL.
- **Générer code objet** : générer un code exécutable sur un AP : oui/non
- **Optimiser code objet** : créer un code plus concis (toutes les optimisations ne sont pas possibles si vous sélectionnez l'option d'informations débogage).
- **Surveiller limites des tableaux** : vérifier si les limites des tableaux se trouvent dans la plage autorisée – définie dans la déclaration du tableau – à l'expiration du temps d'exécution. Si tel n'est pas le cas, le drapeau OK prend la valeur FALSE (à condition que l'option drapeau OK soit cochée).
- **Générer informations débogage** : générer des informations de test : oui/non. Elle sont requises pour effectuer le test avec le débogueur de niveau de langage évolué.
- **Activer drapeau OK** : à l'expiration du temps d'exécution, la variable OK doit prendre la valeur FALSE pour toute opération erronée.
- **Long. de chaîne max.** : réduire la longueur standard du type de données "STRING" qui est de 254 caractères dans la sélection de base. Pour optimiser les ressources de votre CPU, vous pouvez, dans ce cas, en réduire la longueur standard.
- **Imbrication de commentaires possible** : plusieurs commentaires peuvent être imbriqués dans la source SCL : oui/non.

Création du bloc

L'onglet « Création du bloc » vous offre encore d'autres possibilité de paramétrer le processus de compilation :

- Vous pouvez définir si les blocs existants doivent être écrasés ou pas lors de la compilation.
- Vous pouvez sélectionner la génération automatique de données de référence lors de la compilation d'une source. L'activation de cette option rallonge cependant la durée du processus de compilation.
- Vous activez l'option « Prise en compte de l'attribut système S7_server » lorsque le bloc est utilisé pour la configuration de messages ou de liaisons. Cette option rallonge également la durée de la compilation.

Démarrer la compilation

Vous pouvez démarrer la compilation de deux façons :

- Choisissez la commande **Fichier ▸ Compiler**, ou
- cliquez sur le bouton « Compiler » dans la barre d'outils.

Pour vous assurer qu'il s'agit toujours de la dernière version de votre source SCL que vous compilez, il est recommandé de choisir la commande **Outils ▸ Paramètres** et d'activer l'option « Enregistrer avant de compiler » dans l'onglet « Editeur ». La source SCL sera alors enregistrée automatiquement lorsque vous choisirez la commande **Fichier ▸ Compiler**.

Fin de la compilation

Lorsque la compilation est terminée, une fenêtre s'ouvre pour indiquer soit que la compilation s'est correctement déroulée, soit que d'éventuelles erreurs et avertissements sont apparus, voir figure 5-4 :

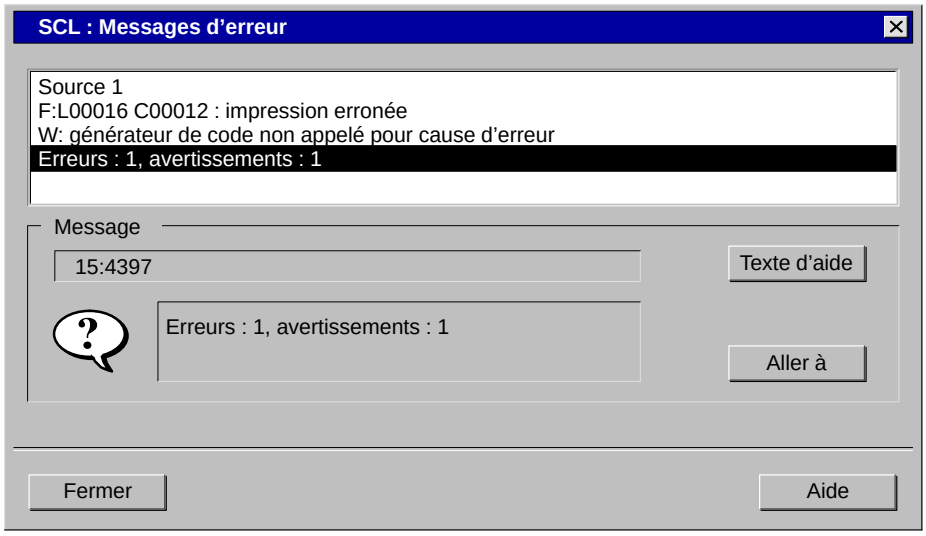


Figure 5-4 *Fenêtre de messages d'erreurs et d'avertissements*

Recherche des causes d'erreurs et d'avertissements

Tout message est indiqué avec sa position (ligne et colonne) de même qu'avec une description succincte. Pour obtenir une description détaillée de l'erreur ou de l'avertissement survenus, sélectionnez le message correspondant et cliquez sur le bouton « Aide ».

En effectuant un double-clic sur un message, vous pouvez positionner le curseur à l'endroit correspondant dans la source SCL.

Ces deux possibilités vous permettent de localiser et de corriger rapidement les erreurs et avertissements.

5.6 Charger le programme utilisateur créé dans l'AP

Présentation

Durant la compilation d'une source SCL, les blocs sont créés à partir de la source et sauvegardés dans le classeur des blocs du programme S7 correspondant. Dans SCL, vous pouvez ensuite charger exclusivement ces blocs depuis la console de programmation dans la CPU.

Si vous souhaitez transmettre d'autres blocs du programme S7 dans votre automate programmable, utilisez SIMATIC Manager.

Conditions préalables

Les conditions suivantes doivent être remplies pour permettre la transmission du programme utilisateur dans l'AP :

- une liaison doit être établie entre la console de programmation et l'automate programmable.
- les blocs que vous voulez charger doivent avoir été compilés sans erreurs.

Effacement général de la mémoire de la CPU

La fonction d'effacement général vous permet d'effacer l'ensemble du programme utilisateur dans une CPU en ligne. Sachez que non seulement la CPU est initialisée, mais que toutes les liaisons avec la CPU sont également annulées et que, si une carte mémoire est enfichée, son contenu est copié dans la mémoire de chargement interne. Procédez de la manière suivante :

1. Choisissez la commande **Système cible ▸ Etat de fonctionnement** et mettez la CPU à l'arrêt.
2. Choisissez la commande **Système cible ▸ Effacement général**.
3. Confirmez l'effacement général dans la boîte de dialogue qui s'ouvre.

Charger dans le système cible

Il est recommandé de charger les blocs à l'état d'arrêt, car à l'état de marche, l'écrasement d'un ancien programme est susceptible de provoquer des erreurs. Procédez de la manière suivante :

1. Choisissez la commande **Système cible ▸ Charger**.
2. Si le bloc est déjà présent dans la RAM de la CPU, confirmez son écrasement lorsque vous y être sollicité.

5.7 Créer un fichier d'informations compilation

Présentation

Vous pouvez automatiser la compilation de plusieurs sources SCL en créant un fichier d'informations compilation.

Fichier d'informations compilation

Vous pouvez créer un fichier d'informations compilation pour votre projet STEP 7. Vous y indiquez les noms des sources SCL contenues dans le projet. Ces sources SCL vont ensuite être compilées dans une procédure de traitement par lots.

Créer

Procédez aux étapes suivantes :

- Si vous créez un fichier avec la commande « Nouveau » ou « Ouvrir », vous devez créer le filtre du « Fichier d'informations compilation ».
- Le fichier que vous utilisez à présent est caractérisé par l'extension « .inp ».
- Lorsque vous compilez ce fichier, les fichiers dont vous avez spécifié le nom sont compilés dans l'ordre indiqué.

Compiler

Durant la compilation, les blocs créés sont rangés dans le classeur des blocs du programme S7.

Test d’un programme

6

Présentation

Les fonctions de test de SCL vous offrent la possibilité de contrôler l’exécution d’un programme dans l’AP pour y déceler d’éventuelles erreurs.

Durant la compilation, SCL reconnaît et affiche les erreurs de syntaxe. Les erreurs de temps de cycle durant l’exécution du programme sont signalées par des alarmes système ; quant aux erreurs de programmation logiques, vous pouvez les rechercher à l’aide des fonctions de test de SCL.

Informations supplémentaires

L’aide en ligne de SCL fournit des informations détaillées sur le test et répond aux questions précises que vous vous posez lorsque vous utilisez SCL.

Structure du chapitre

6.1 Présentation

Niveau de langage évolué

Les fonctions de test de SCL vous permettent de tester les programmes utilisateur que vous avez programmés pour ce qui est de leur niveau de langage évolué. Avec ce type de test vous pouvez :

- déceler des erreurs de programmation,
- visualiser et contrôler l'action d'un programme utilisateur sur l'exécution dans la CPU.

Conditions préalables

Avant de tester un programme SCL, vous devez procéder aux étapes suivantes :

1. Vous devez compiler le programme en ayant activé les options de compilation « Création du code objet » et « Création de l'information Déboguage ». Vous les sélectionnez dans l'onglet « Compilateur » de la boîte de dialogue que vous obtenez en choisissant la commande **Outils ▸ Paramètres**.
2. Vous devez établir une liaison en ligne entre votre PG/PC et la CPU.
3. Vous devez en outre charger le programme dans la CPU. C'est ce que vous réalisez avec la commande **Système cible ▸ Charger**.

Fonctions de test de SCL

Le tableau 6-1 désigne et résume les caractéristiques des principales fonctions de test que vous pouvez appeler dans SCL.

Tableau 6-1 Présentation des fonctions de test

Fonction	Caractéristiques
Visualisation contine (CPU S7-300/400)	Affichage des noms et des valeurs actuelles des variables d'un domaine de visualisation
Activer les points d'arrêt (uniquement CPU S7-400)	Définition, suppression et édition des points d'arrêt : test pas à pas
Visualiser/forçer des variables	Visualisation/forçage des valeurs actuelles des données globales
Génération des données de référence	Création d'un aperçu du programme utilisateur
Fonctions de test du logiciel de base STEP 7	Consultation/modification de l'état de fonctionnement de la CPU

Avertissement

La défaillance d'une fonction ou une erreur de programmation survenant durant l'exécution du test lorsque l'installation est en marche peuvent occasionner des dommages matériels et personnels ! Avant d'exécuter une fonction de test, assurez-vous qu'aucune situation dangereuse ne soit susceptible de se produire.

6.2 Fonction de test « Visualisation continue »

Présentation

La visualisation continue d’un programme consiste à tester un ensemble d’instructions, également appelé domaine de visualisation.

Durant la phase de test, les valeurs des variables et des paramètres de ce domaine sont affichées chronologiquement et actualisées cycliquement. Si le domaine de visualisation se trouve dans une section de programme qui s’exécute à chaque cycle, les valeurs des variables ne peuvent en principe pas être lues pour des cycles consécutifs.

Les valeurs qui se sont modifiées durant le cycle actuel sont affichées en noir, celle qui ne se sont pas modifiées en gris clair.

Le nombre d’instructions pouvant être testées est fonction de la puissance des CPU connectées. Comme les instructions SCL du code source sont représentées dans un grand nombre d’instructions différentes du code machine, la longueur du domaine de visualisation est variable. C’est SCL qui la détermine et la sélectionne lorsque vous choisissez la première instruction du domaine de visualisation souhaité.

Mode de test

Lorsque vous effectuez un test en mode de « Visualisation continue », les valeurs actuelles des données comprises dans le domaine de visualisation sont consultées et affichées. La consultation a lieu pendant l’exécution du domaine de visualisation. Elle entraîne généralement une augmentation du temps de cycle, que vous pouvez cependant influencer grâce aux deux environnements de test proposés par SCL :

- **Environnement de test « Processus » :**

  Dans l’environnement de test « Processus », le débogueur de SCL limite le domaine de visualisation maximal, de sorte à ce que durant le test, le temps de cycle ne dépasse pas ou à peine le temps d’exécution réel du processus.

- **Environnement de test « Laboratoire » :**

  Dans l’environnement de test « Laboratoire », le domaine de visualisation n’est limité que par la puissance de la CPU connectée. Le temps de cycle peut cependant être plus long que le processus réel, mais le domaine de visualisation maximal est supérieur à celui de l’environnement de test « Processus ».

Mise en œuvre de la « Visualisation continue »

Pour exécuter la fonction « Visualisation continue », procédez de la manière suivante :

1. Assurez-vous que les conditions préalables enoncées au paragraphe 6.1 sont bien remplies.
2. Activez la fenêtre qui contient la source du programme à tester.
3. Si vous souhaitez changer l'environnement de test prédéfini (processus), choisissez la commande **Test ▸ Environnement de test ▸ Laboratoire**.
4. Positionnez le point d'insertion dans la ligne du texte source contenant la première instruction du domaine à tester.
5. Choisissez la commande **Test ▸ Visualisation continue**.

   Résultat : le domaine de visualisation le plus grand possible va alors être déterminé et signalé par une barre grise au bord gauche de la fenêtre. La fenêtre se scinde en deux et sa partie droite affiche ligne par ligne les noms et valeurs actuelles des variables comprises dans le domaine de visualisation.
6. Choisissez la commande **Test ▸ Visualisation continue**, pour interrompre le test et le poursuivre ultérieurement.
7. Choisissez la commande **Test ▸ Mettre fin au test**, pour terminer le test.

6.3 Fonction de test « Activer les points d'arrêt »

Présentation

Le test avec la fonction « Activer les points d'arrêt » s'effectue pas à pas. Vous pouvez exécuter le programme instruction par instruction et visualiser comment les valeurs des variables éditées varient.

Après avoir défini des points d'arrêt, vous pouvez dans un premier temps exécuter le programme jusqu'à l'un d'entre eux, et à partir de là, commencer la visualisation pas à pas.

Points d'arrêt

Vous pouvez définir des points d'arrêt en divers endroits dans la section d'instructions du texte source.

Les points d'arrêt ne sont chargés dans l'automate programmable et activés que lorsque vous choisissez la commande **Test ▸ Activer les points d'arrêt**. Le programme est alors exécuté jusqu'à ce que le premier point d'arrêt est atteint.

Le nombre de points d'arrêt actifs dépend de la CPU :

- CPU 416 : au maximum 4 points d'arrêt actifs
- CPU 414 : au maximum 2 points d'arrêt actifs
- CPU 314 : aucun point d'arrêt actif possible

Fonctions de test pas à pas

Après avoir appelé la fonction de test **Activer les points d'arrêt**, vous pouvez exécuter les fonctions suivantes :

- **Instruction suivante**

  Exécution de l'instruction suivante. Permet d'afficher les valeurs des variables.

- **Poursuivre**

  Poursuite de l'exécution jusqu'au prochain point d'arrêt activé.

- **Jusqu'à la sélection**

  Poursuite de l'exécution jusqu'à la position du curseur que vous définissez dans la source.

Nota

Veillez à ne pas dépasser le nombre maximal de points d'arrêt actifs autorisés lorsque vous utilisez les commandes **Instruction suivante** ou **Jusqu'à la sélection**, ces dernières créant et activant automatiquement un point d'arrêt.

Mise en œuvre de « Activer les points d'arrêt »

Avant de débuter le test, assurez-vous que les conditions citées au chapitre 6.1 sont remplies. Vous pouvez alors tester votre programme pas à pas avec la fonction « Activer les points d'arrêt ». La manière de procéder est décrite dans les étapes suivantes et représentée à la figure 6-1 :

1. Sélectionnez la fenêtre qui contient la source du bloc à tester.
2. Sélectionnez les points d'arrêt en positionnant le curseur à l'endroit voulu dans la source du programme et en choisissant soit la commande **Test ▸ Définir un point d'arrêt**. Les points d'arrêt sont représentés sous forme de cercle rouge au bord gauche de la fenêtre.
3. Démarrez ensuite le test pas à pas en choisissant la commande **Test ▸ Activer les points d'arrêt**.

   Résultat : la fenêtre se divise verticalement en deux parties et le programme démarre. Le point d'arrêt suivant est recherché. Quand il est atteint, la CPU passe à l'état de fonctionnement ATTENTE et la position recherchée est marquée par une flèche jaune.
4. Vous disposez des fonctions de test pas à pas suivantes :
   - Choisissez la commande **Test ▸ Instruction suivante** (4a) :
     Résultat : la CPU se met brièvement à l'état de marche. Lorsque l'instruction suivante est atteinte, elle se met à nouveau à l'arrêt et affiche ligne par ligne dans la partie gauche de la fenêtre la **valeur des variables** éditées dans l'instruction précédente.
   - Choisissez la commande **Test ▸ Poursuivre** (4b) :
     Résultat : la CPU se met à l'état de marche. Lorsque le point d'arrêt actif suivant est atteint, elle se met à nouveau à l'arrêt et signale le point d'arrêt au bord gauche de la fenêtre. Pour afficher les valeurs des variables, vous devez choisir à nouveau la commande **Test ▸ Instruction suivante**.
   - Choisissez la commande **Test ▸ Jusqu'à la sélection** (4c) :
     Un point d'arrêt est automatiquement défini et activé à l'endroit de la sélection. La CPU se met en marche. Lorsque la sélection est atteinte, elle se met à nouveau à l'arrêt et sélectionne ce point d'arrêt. Pour afficher les valeurs des variables, vous devez choisir à nouveau la commande **Test ▸ Instruction suivante**.
5. Si vous souhaitez poursuivre le test avec des points d'arrêt modifiés, retournez à l'étape 2. Vous pouvez y définir de nouveaux points d'arrêt ou encore effacer des points d'arrêt existants.
6. Choisissez la commande de menu **Test ▸ Activer les points d'arrêt** pour interrompre la boucle de test.
7. Si vous ne souhaitez pas tester d'autre instruction dans la source, mettez fin au test en choisissant la commande de menu **Test ▸ Mettre fin au test**.

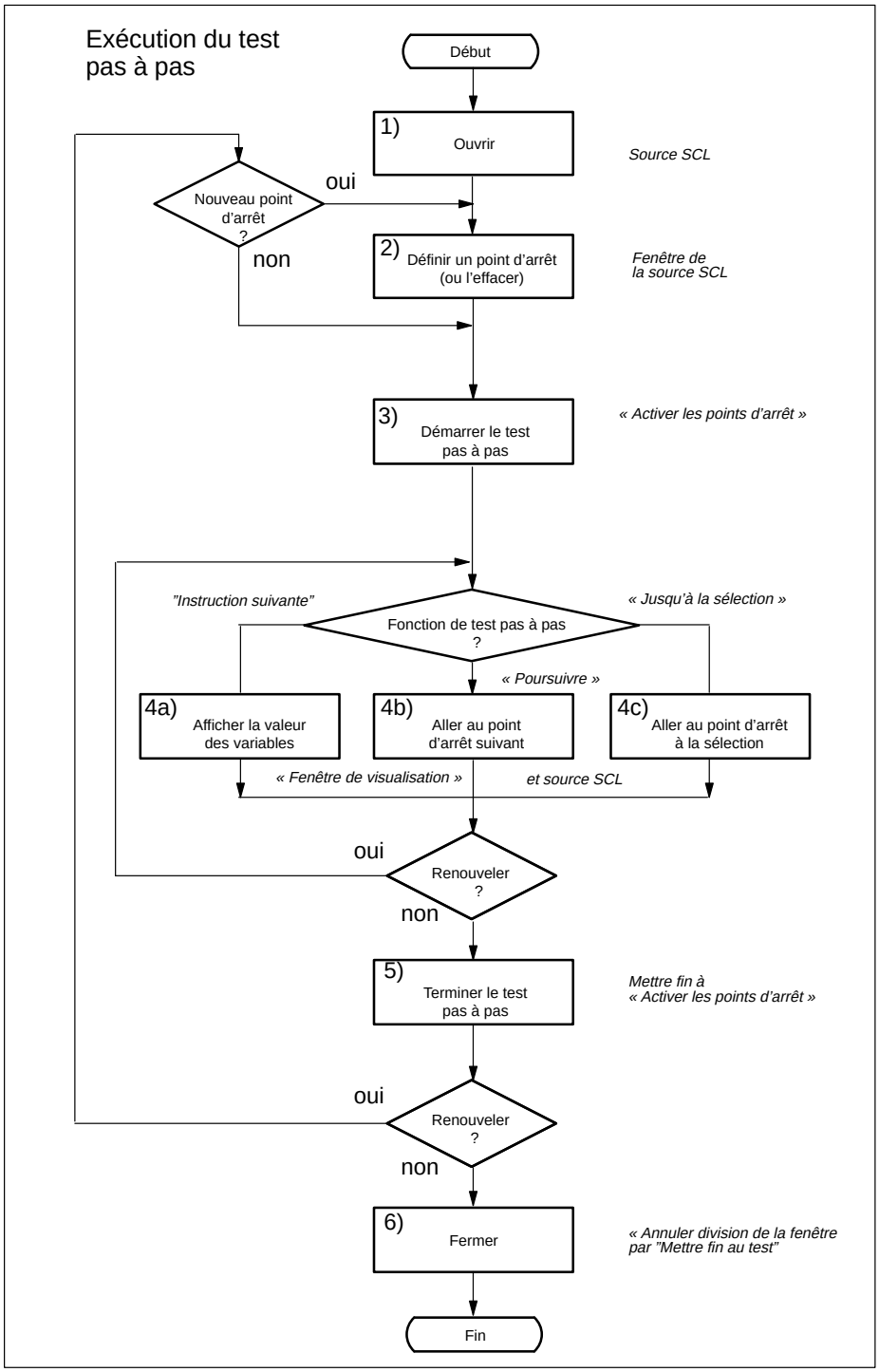


Figure 6-1 *Algorithme d'exécution du test*

6.4 Fonction de test « Visualiser/forcer des variables »

Présentation

Le test avec la fonction « Visualiser/forcer des variables » vous permet :

- d'afficher (observer) les valeurs actuelles des données globales de votre programme utilisateur
- d'attribuer des valeurs aux variables d'un programme utilisateur (les forcer).

Visualiser et forcer des variables

En choisissant la commande **Système cible ▸ Visualiser/forcer des variables**, vous pouvez :

- définir des points et conditions de déclenchement,
- attribuer des valeurs aux variables d'un programme utilisateur.

Vous devez dans les deux cas établir une table des variables dans laquelle vous indiquez quelles variables vous allez éditer. Dans le cas du « forçage », vous devez en plus donner les valeurs souhaitées.

Une description détaillée de la fonction de test est faite dans le guide de l'utilisateur STEP 7, **/231/**.

6.5 Fonction de test « Données de référence »

Présentation

Pour vous faciliter le test et la modification de votre programme utilisateur, vous pouvez générer et exploiter des données de référence.

Les données de référence englobent : la structure de programme, la liste des références croisées, le tableau d’affectation, la liste des opérandes libres ainsi que la liste des mnémoniques manquants.

Vous pouvez utiliser les données de référence :

- pour obtenir un aperçu de l’ensemble de votre programme utilisateur,
- comme base pour effectuer des modifications et tests,
- pour compléter la documentation de votre programme.

Générer des données de référence

Pour générer des données de référence, vous avez plusieurs possibilités :

- Vous pouvez générer, actualiser et afficher les données de référence si cela est nécessaire, en choisissant la commande **Outils ▸ Données de référence**.
- En choisissant la commande **Outils ▸ Paramètres**, vous pouvez définir que les données de référence soient générées automatiquement lors de la compilation d’une source. Pour ce faire, sélectionnez l’option « Génération des données de référence » dans la page d’onglet « Création de blocs ».

  La génération automatique des données de référence rallonge cependant la durée du processus de compilation.

Une description détaillée de la fonction de test est faite dans le guide de l’utilisateur STEP 7, **/231/**.

6.6 Utilisation des fonctions de test de STEP 7

Editeur LIST

Dans STEP 7, vous avez la possibilité d'ouvrir en LIST des blocs compilés avec SCL afin de les tester dans le mode de représentation LIST.

Consulter et modifier l'état de fonctionnement de la CPU

Pour visualiser ou modifier l'état de fonctionnement actuel de la CPU, choisissez la commande **Système cible ▸ Etat de fonctionnement**.

Afficher les propriétés de la CPU

Dans la boîte de dialogue qui s'ouvre lorsque vous choisissez la commande **Système cible ▸ Etat du module**, vous pouvez :

- constater la cause du passage à l'état d'arrêt par lecture de la mémoire de diagnostic,
- interroger le contenu des piles de la CPU. La pile des interruptions constitue une aide précieuse dans la détermination des causes d'erreur,
- consulter les caractéristiques techniques de la CPU,
- afficher la date et l'heure réglées pour la CPU,
- afficher le temps de cycle de la CPU,
- obtenir des informations sur les blocs contenus dans la CPU,
- obtenir des informations sur la communication avec la CPU.

Pour que vous puissiez exécuter toutes ces fonctions, il est impératif que la CPU soit en ligne.

Troisième partie : Description du langage

Notions fondamentales dans SCL

7

Présentation

Le présent chapitre décrit les divers éléments de langage que SCL met à votre disposition ainsi que la manière de les utiliser. Il constitue une simple introduction aux concepts de base de SCL et donne uniquement les définitions fondamentales qui seront approfondies dans les chapitres suivants.

Structure du chapitre

7.1 Aide relative à la description du langage

Description du langage SCL

Les diagrammes syntaxiques servent de base à la description du langage faite dans les chapitres suivants. Ils permettent d'illustrer la structure syntaxique (c'est-à-dire grammaticale) dans SCL. Ces diagrammes sont regroupés au complet avec les éléments de langage correspondants en annexe B du présent manuel.

Définition d'un diagramme syntaxique

Il s'agit d'une représentation graphique de la structure du langage. Cette structure est constituée d'une suite de règles, pouvant elles-mêmes se fonder sur des règles déjà énoncées.

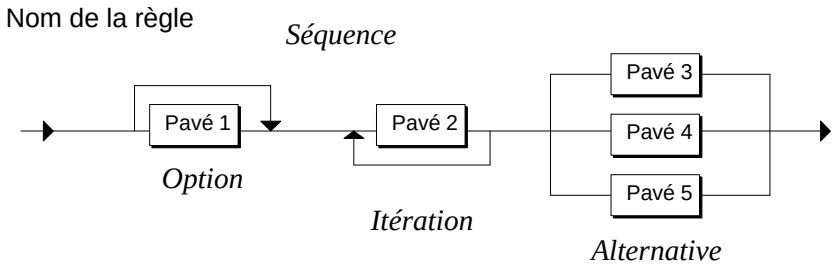


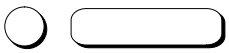
Figure 7-1 *Diagramme syntaxique*

Le diagramme syntaxique se lit de gauche à droite. En voici la structure :

- séquence : suite de pavés
- option : branchement facultatif
- itération : répétition d'un branchement
- alternative : branchement

Types de pavés

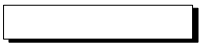
Un pavé peut être un élément de base ou alors être lui-même composé d'autres éléments de base. La figure suivante illustre les types de symboles représentant les pavés :



Elément de base qui ne nécessite pas de description supplémentaire.

Il s'agit de caractères imprimables et de caractères spéciaux, de mots-clés et d'identificateurs prédéfinis.

Les données relatives à ces pavés doivent être reprises telles quelles.



Elément composé qui est décrit par d'autres diagrammes syntaxiques.

Signification du format libre

Lorsque vous saisissez un texte source, vous devez tenir compte aussi bien des **règles syntaxiques** que des **règles lexicales**. Ces deux types de règles sont représentés en annexe B et C.

Le format libre signifie qu'entre les pavés, vous pouvez insérer des caractères de mise en forme, comme par exemple des caractères d'espacement, des tabulateurs, des caractères de changement de page ainsi que des commentaires.

Règle lexicale

Pour les règles lexicales, comme par exemple à la figure 7-2, le format n'est **pas** libre. Lorsque vous appliquez une règle lexicale, vous devez reprendre les données telles quelles.

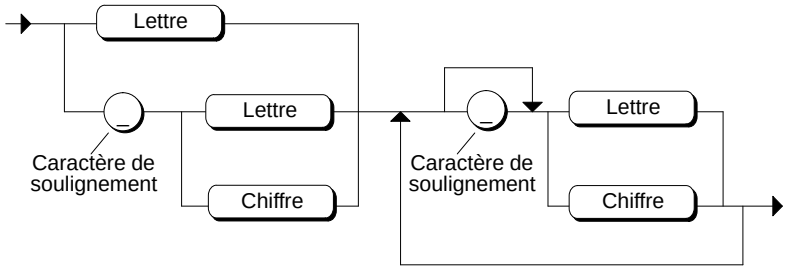


Figure 7-2 *Exemple de règle lexicale*

Exemples dans lesquels la règle représentée est correctement appliquée :

```
R_REGULATEUR3
_A_TABLEAU
_100_3_3_10
```

Exemples dans lesquels la règle n'est pas correctement appliquée :

```
1_1AB
RR__20
*#AB
```

Règle syntaxique

Pour les règles syntaxiques (voir, par exemple, figure 7-3), le format est libre.

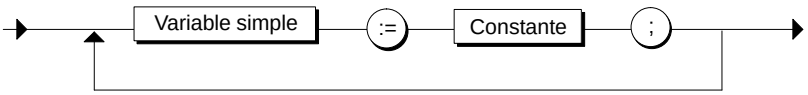


Figure 7-3 *Exemple de règle syntaxique*

Exemples dans lesquels la règle représentée est correctement appliquée :

```
VARIABLE_1  := 100;     COMMUTATEUR:=FALSE;
VARIABLE_2  := 3.2;
```

7.2 Jeu de caractères de SCL

Caractères, chiffres

Dans le jeu de caractères ASCII partiel, SCL utilise :

- les lettres (minuscules et majuscules) de A à Z,
- les chiffres arabes de 0 à 9,
- le caractère d'espacement (code ASCII 32) ainsi que tous les caractères de commande (ASCII 0-31), y compris le caractère de fin de ligne (ASCII 13).

Autres caractères

Les caractères suivants ont une signification bien précise dans SCL :

+	–	*	/	=	<	>	[	]	(	)
.	,	:	;	$	#	”	’	{	}	

Informations supplémentaires

En annexe A, vous trouverez une liste détaillée de tous les caractères utilisables, précisant comment SCL les interprète.

7.3 Mots réservés

Définition

Il s'agit de mots-clés dont l'usage est prédéfini. Il n'y a pas de distinction entre les minuscules et les majuscules.

Mots-clés

AND
ANY
ARRAY
BEGIN
BLOCK_DB
BLOCK_FB
BLOCK_FC
BLOCK_SDB
BLOCK_SFB
BLOCK_SFC
BOOL
BY
BYTE

CASE
CHAR
CONST
CONTINUE
COUNTER
DATA_BLOCK
DATE
DATE_AND_TIME
DINT
DIV
DO
DT
DWORD
ELSE
ELSIF
END_CASE
END_CONST
END_DATA_BLOCK
END_FOR
END_FUNCTION
END_FUNCTION_BLOCK
END_STRUCT
END_VAR
END_WHILE
EXIT
FOR
FUNCTION
FUNCTION_BLOCK
GOTO
IF
INT
LABEL
MOD
NIL
NOT
OF
OR
ORGANIZATION_BLOCK
POINTER
REAL
REPEAT
RETURN
S5TIME
STRING
STRUCT
THEN
TIME
TIMER
TIME_OF_DAY
TO
TOD
TYPE
VAR
VAR_TEMP
UNTIL
VAR_INPUT

Mots-clés, suite

END_IF
END_LABEL
END_TYPE
END_ORGANIZATION_BLOCK
END_REPEAT
VOID
VAR_IN_OUT
VAR_OUTPUT
WHILE
WORD
XOR

Autres noms réservés

EN
ENO
OK
TRUE
FALSE
Nom des fonctions standard [1]

7.4 Identificateurs dans SCL

Définition

Un identificateur est un nom que vous pouvez attribuer à un objet du langage de SCL, à savoir une constante, une variable, une fonction ou un bloc.

Règles

Il peut s'agir d'une combinaison quelconque de lettres et de chiffres, le premier caractère devant obligatoirement être une lettre ou le caractère de soulignement. Aussi bien les minuscules que les majuscules sont permises. La distinction entre les majuscules et les minuscules n'est pas faite (Anna et AnNa sont par exemple identiques).

Diagramme syntaxique formel permettant de représenter un identificateur :

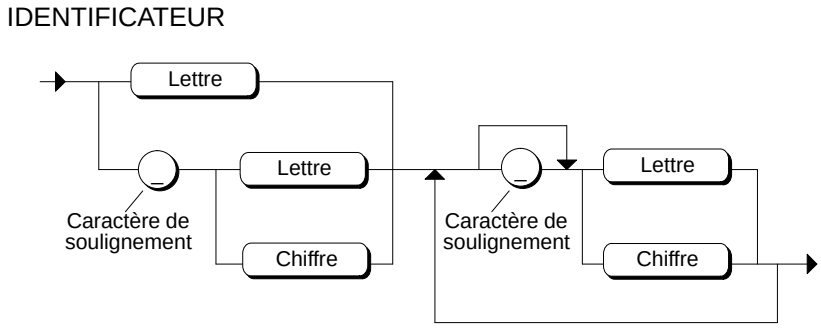


Figure 7-4 *Syntaxe : identificateur*

Tenez compte des remarques suivantes :

- Nous vous recommandons de choisir des noms univoques et évocateurs qui contribuent à l'intelligibilité du programme.
- Assurez-vous que le nom n'appartient ni à un mot-clé (comme, par exemple, ceux du tableau 7-1) ni à un identificateur standard.
- La longueur maximale d'un identificateur est de 24 caractères.
- Vous devez définir les identificateurs de blocs (identificateur autres que ceux du tableau 7-1) dans la table des mnémoniques de STEP 7 (des informations détaillées à ce sujet sont données dans le guide de l'utilisateur /**231**/)

Exemples

Les noms d'identificateurs suivants sont autorisés :

```
x          y12        Somme      Température
Nom        Surface    Régulateur Tableau
```

Les noms d'identificateurs suivants sont **interdits** pour les raisons indiquées :

`4ème`	Le premier caractère doit être une lettre ou le caractère de soulignement.
`Array`	ARRAY est un mot-clé et n'est pas autorisé.
`S Valeur`	Les caractères d'espacement comptent comme des caractères et ne sont pas autorisés.

7.5 Identificateurs standard

Définition

Une série d'identificateurs sont déjà prédéfinis dans SCL. Il s'agit des *identificateurs standard* suivants :

- mots-clés de blocs et
- identificateurs d'opérandes pour adresser les zones de mémoire de la CPU.

Mots-clés de blocs

Ces identificateurs standard sont utilisés pour l'adressage absolu de blocs.

Le tableau 7-1 donne les mots-clés de blocs classés d'après les abréviations SIMATIC, avec les abréviations internationales CEI correspondantes.

Tableau 7-1 Mots-clés de blocs

Abréviations SIMATIC	Abréviations CEI	Désignation
DBx	DBx	Bloc de données (Data- Block)
FBx	FBx	Bloc fonctionnel (Function- Block)
FCx	FCx	Fonction (Function)
OBx	OBx	Bloc d'organisation (Organization- Block)
SDBx	SDBx	Bloc de données système (System- Data- Block)
SFCx	SFCx	Fonction système (System- Function)
SFBx	SFBx	Bloc fonctionnel système (System- Function- Block)
Tx	Tx	Temporisation (Timer)
UDTx	UDTx	Type de données global ou utilisateur (Userdefined Data-Type)
Zx	Cx	Compteur (Counter)
x nombre compris entre 0 et 32767 DB0 est réservé		

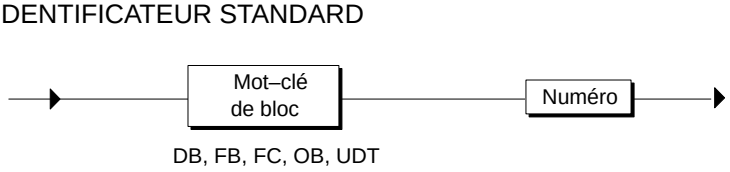


Figure 7-5 *Syntaxe : identificateur standard*

Voici des désignations autorisées :

```
FB10
DB100
T141
```

Identificateurs d'opérandes

Un identificateur d'opérande permet d'adresser la zone de mémoire correspondante de la CPU, à partir d'un endroit quelconque du programme.

Le tableau suivant donne les identificateurs d'opérandes classés d'après les abréviations SIMATIC, avec les abréviations internationales CEI correspondantes.

Abréviations SIMATIC	Abréviations CEI	Adressage	Type de données
Ax,y	Qx,y	Sortie (par la mémoire image)	Bit
ABx	QBx	Sortie (par la mémoire image)	Octet
ADx	QDx	Sortie (par la mémoire image)	Double mot
AWx	QWx	Sortie (par la mémoire image)	Mot
AXx.y	QXx.y	Sortie (par la mémoire image)	Bit
Dx.y [1]	Dx.y [1]	Bloc de données	Bit
DBx [1]	DBx [1]	Bloc de données	Octet
DDx [1]	DDx [1]	Bloc de données	Double mot
DWx [1]	DWx [1]	Bloc de données	Mot
DXx	DXx	Bloc de données	Bit
Ex.y	Ix.y	Entrée (par la mémoire image)	Bit
EBx	IBx	Entrée (par la mémoire image)	Octet
EDx	IDx	Entrée (par la mémoire image)	Double mot
EWx	IWx	Entrée (par la mémoire image)	Mot
EXx.y	IXx.y	Entrée (par la mémoire image)	Bit
Mx.y	Mx.y	Mémento	Bit
MBx	MBx	Mémento	Octet
MDx	MDx	Mémento	Double mot
MWx	MWx	Mémento	Mot
MXx.y	MXx.y	Mémento	Bit
PABx	PQBx	Sortie (périphérie directe)	Octet
PADx	PQDx	Sortie (périphérie directe)	Double mot
PAWx	PQWx	Sortie (périphérie directe)	Mot
PEBx	PIBx	Entrée (périphérie directe)	Octet
PEDx	PIDx	Entrée (périphérie directe)	Double mot
PEWx	PIWx	Entrée (périphérie directe)	Mot
x = nombre compris entre 0 et 65535 (adresse d'octet absolue) y = nombre compris entre 0 et 7 (numéro de bit)			

Voici des exemples autorisés :

```
E1.0        MW10        PAW5        DB20.DW3
```

1 *Ces identificateurs d'opérandes ne sont valables que s'ils sont indiqués avec le bloc de données correspondant*

7.6 Nombres

Présentation

Il y a plusieurs manières d'écrire un nombre dans SCL. Il peut comporter un signe, une virgule décimale (représentée par un point) ou un exposant. Les règles suivantes s'appliquent à tous les nombres :

1. Un nombre ne doit comporter ni virgule, ni caractère d'espacement.
2. Le caractère de soulignement (_) peut être utilisé comme séparateur optique.
3. Un nombre peut être précédé soit d'un plus (+), soit d'un moins (–). En l'absence de signe le nombre est considéré comme étant positif.
4. Un nombre doit être compris entre des valeurs minimales et maximales données.

Nombres entiers

Un nombre entier ne comporte ni virgule décimale, ni exposant. Il s'agit simplement d'une simple suite de chiffres, pouvant être signée. Dans SCL, il existe deux types d'entiers avec des plages de valeurs différentes, `INT` et `DINT` (voir chapitre 9).

Voici des nombres entiers autorisés :

```
0          1          +1         -1
743        -5280      600_00     -32_211
```

Les nombres suivants ne sont **pas** autorisés pour les raisons indiquées :

123,456	Les virgules sont interdites.
36.	La virgule décimale (représentée par un point) est interdite dans un nombre entier.
10 20 30	Les caractères d'espacement ne sont pas autorisés.

Nombres entiers sous forme de nombres binaires, octaux ou hexadécimaux

Dans SCL, vous pouvez représenter un nombre entier dans différents systèmes de numération. Il faut alors le faire précéder d'un mot-clé pour désigner le système dont il s'agit. 2# désigne le système binaire, 8# désigne le système octal et 16# désigne le système hexadécimal.

Voici le nombre décimal 15 correctement représenté dans d'autres systèmes de numération :

```
2#1111     8#17       16#F
```

Nombres réels

Un nombre réel doit comporter une virgule décimale ou un exposant (ou les deux). La virgule décimale doit figurer entre deux chiffres. Un nombre réel ne peut donc ni commencer, ni se terminer par une virgule décimale.

Voici des nombres réels autorisés :

```
0.0        1.0        -0.2       827.602
50000.0    -0.000743  12.3       -315.0066
```

Les nombres réels suivants ne sont **pas corrects** pour les raisons indiquées :

`1.`	La virgule décimale (point) doit figurer entre deux chiffres.
`1,000.0`	Les virgules ne sont pas autorisées.
`.3333`	La virgule décimale doit figurer entre deux chiffres.

Un exposant sert à indiquer la position de la virgule décimale. En l'absence de cette dernière, l'on considère qu'elle se trouve à droite du chiffre. Quant à l'exposant, il doit s'agir d'un nombre entier positif ou négatif. La base 10 est représentée par la lettre E.

Dans SCL, la grandeur 3 x 10^{10} équivaut aux nombres réels suivants :

```
3.0E+10   3.0E10        3e+10       3E10
0.3E+11   0.3e11        30.0E+9     30e9
```

Les nombres réels suivants ne sont **pas corrects** pour les raisons indiquées :

`3.E+10`	La virgule décimale doit figurer entre deux chiffres.
`8e2.3`	L'exposant doit être un nombre entier.
`.333e-3`	La virgule décimale doit figurer entre deux chiffres.
`30 E10`	Les caractères d'espacement ne sont pas autorisés.

Chaînes de caractères

Une chaîne de caractères est une suite de caractères (lettres, chiffres et caractères spéciaux) entourés d'apostrophes. Les majuscules et les minuscules sont autorisées.

Voici des chaînes de caractères autorisées :

```
'ROUGE'   '75 000 Paris'            '270-32-3456'
'Fr19.95' 'Le résultat correct est :'
```

Pour saisir un caractère de mise en forme spécial, une apostrophe (') ou le caractère $, faites le précéder du caractère $.

Texte source	après compilation
`'SIGNAL$'ROUGE$''`	SIGNAL'ROUGE'
`'50.0$$'`	50.0$
`'VALEUR$P'`	VALEUR *nouvelle page*
`'REG-$L'`	REG- *changement de ligne*
`'REGULATEUR$R`	REGULATEUR *retour chariot*
`'ETAPE$T'`	ETAPE *tabulation*

Pour indiquer un caractère non imprimable, entrez sa représentation en code hexadécimal $*hh*, où *hh* correspond à la valeur hexadécimale du caractère ASCII.

Pour interrompre une chaîne de caractères (pour des commentaires qui ne doivent être ni imprimés, ni affichés), SCL met à votre disposition les caractères $> et $< .

7.7 Types de données

Généralités

Dans toute déclaration de variable, vous devez indiquer son type de données. Il détermine la plage des valeurs de la variable et définit les opérations qu'elle permet de réaliser.
Un type de données comprend :

- le type et la signification de l'élément de données,
- la plage autorisée de l'élément de données,
- le nombre d'opérations que vous pouvez réaliser avec un opérande de ce type de données et
- la notation des données de ce type.

Types de données

On distingue les types de données suivants.

Tableau 7-2 Types de données

Type de données	Signification
simple	Mis à votre disposition de manière standard par SCL.
complexe	Vous pouvez le créer en combinant des types de données simples.
utilisateur	Vous pouvez le définir spécialement pour votre application et lui attribuer un nom quelconque.
type de paramètre	Ils servent uniquement à déclarer des paramètres.

Type de données simple

Le type de données simple définit une structure de données que l'on ne peut pas décomposer en unités plus petites. Il répond à la norme DIN EN 1131-3.

Dans SCL, douze types de données simples sont prédéfinis :

```
BOOL        CHAR        INT         TIME
BYTE                    DINT        DATE
WORD                    REAL        TIME_OF_DAY
DWORD                               S5TIME
```

Type de données complexe

Le type de données complexe définit une structure de données qui peut être décomposée en d'autres types de données. SCL autorise les types de données complexes suivants :

```
DATE_AND_TIME
STRING
ARRAY
STRUCT
```

Type de données utilisateur

Il s'agit d'un type de données global (UDT) que vous pouvez créer dans SCL pour votre application. Vous pouvez l'utiliser dans la section de déclaration d'un bloc ou bloc de données sous sa désignation UDTx (x correspond à son numéro) ou sous le mnémonique que vous lui avez attribué.

Types de paramètres

Outre les types de données simples, complexes et utilisateur, vous pouvez également utiliser des types de paramètres afin de définir des paramètres. SCL propose les types de paramètres suivants :

```
TIMER       BLOCK_FB     POINTER      ANY
COUNTER     BLOCK_FC
            BLOCK_DB
            BLOCK_SDB
```

7.8 Variables

Déclaration de variables

On appelle *variable*, un identificateur dont la valeur peut être modifiée durant l'exécution du programme. Toute variable doit être déclarée individuellement avant de pouvoir être utilisée dans un bloc de code ou de données. La déclaration de la variable permet de définir un identificateur comme variable (et non pas comme constante etc.) et d'en spécifier le type en l'affectant au type de données.

On distingue les variables suivantes :

- données locales,
- données utilisateur globales et
- variables prédéfinies (zones de mémoire d'une CPU).

Données locales

Il s'agit de données qui sont déclarées dans un bloc de code (FC, FB, OB) et qui ne sont valables que pour celui-ci.

Tableau 7-3 Données locales d'un bloc

Variable	Signification
Variable statique	Une variable statique est une variable locale dont la valeur reste inchangée tout au long de l'exécution du bloc (mémoire du bloc). Elle sert à enregistrer les valeurs d'un **bloc fonctionnel**.
Variable temporaire	Une variable temporaire appartient localement à un bloc de code et n'occupe **aucune** zone de mémoire statique. Sa valeur ne reste conservée que durant une exécution du bloc. Il n'est **pas** possible d'y accéder en dehors du bloc dans lequel elle a été déclarée.
Paramètre de bloc	Il s'agit d'un paramètre formel de bloc fonctionnel ou de fonction. C'est une variable locale qui sert à transmettre les paramètres actuels indiqués à l'appel du bloc.

Données utilisateur globales

Les données utilisateur globales sont des données ou zones de données que vous pouvez utiliser à partir d'un endroit quelconque du programme. Vous devez à cet effet créer des blocs de données (DB).

Lorsque vous créez un DB, vous définissez sa structure en la déclarant. Vous pouvez cependant également utiliser un type de données utilisateur (UDT) au lieu de déclarer la structure du DB. L'ordre dans lequel vous indiquez la composante de la structure détermine celui des données dans le DB.

Zones de mémoire d'une CPU

Vous pouvez adresser les zones de mémoire d'une CPU directement à partir d'un endroit quelconque du programme, en utilisant des identificateurs d'opérandes (voir paragraphe 7.5) qu'il est inutile de déclarer.

Vous avez en outre toujours la possibilité d'adresser ces zones de données de façon symbolique. Dans ce cas, vous leur affectez des mnémoniques dans STEP 7, avec une table des mnémoniques. Vous trouverez des informations complémentaires à ce sujet dans le guide de l'utilisateur **/231/**.

7.9 Expressions

Présentation

Une expression représente une valeur qui sera calculée durant la compilation ou l'exécution du programme. Elle comporte un ou plusieurs opérandes combinés par des opérateurs, dont l'ordre d'exécution dépend de leur priorité. L'utilisation de parenthèses permet de modifier la priorité. SCL distingue :

- des expressions arithmétiques,
- des expressions logiques,
- des expressions de comparaison.

Expression arithmétique

En voici un exemple typique :

```
(b*b-4*a*c)/(2*a)
```

Les identificateurs a et b ainsi que les nombres 4 et 2 sont les opérandes ; les symboles *, - et / (multiplication, soustraction et division) sont les opérateurs. L'ensemble de l'expression représente un nombre.

Expression de comparaison

Il s'agit d'une expression logique dont le résultat peut être vrai ou faux. En voici un exemple :

```
Valeur_consigne < 100.0
```

Dans cet exemple, `Valeur_consigne` est une variable réelle et le symbole < un opérateur de comparaison. La valeur de l'expression est vraie si `Valeur_consigne` est inférieure à 100.0, sinon elle est fausse.

Expression logique

En voici un exemple typique :

```
a AND NOT b
```

Les identificateurs a et b sont les opérandes ; les mots-clés AND et NOT sont les opérateurs logiques. L'ensemble de l'expression représente un élément binaire.

7.10 Instructions

Présentation

Dans SCL, une instruction désigne une action exécutable dans la section des instructions d'un bloc de code. Il en existe trois types principaux :

1. Affectations de valeurs (affectation d'une expression à une variable).
2. Instructions de contrôle (itération ou branchement d'instructions).
3. Traitements de sous-programmes (appels ou branchements de blocs de code).

Affectation de valeur

Voici un exemple typique d'affectation de valeur :

```
Valeur_consigne := 0.99*Ancienne_valeur_consigne
```

Dans cet exemple, on a comme hypothèse que `Valeur_consigne` et `Ancienne_valeur_consigne` sont des variables réelles. L'instruction d'affectation multiplie la valeur `Ancienne_valeur_consigne` par 0.99 et en affecte le produit à la variable `Valeur_consigne`. Le symbole d'affectation est " :=".

Instruction de contrôle

En voici un exemple typique :

```
FOR Compteur :=1 TO 20 DO
            LISTE[Compteur]  := VALEUR+Compteur;

END_FOR;
```

Dans cet exemple, l'affectation est réalisée 20 fois. A chaque fois, la nouvelle valeur calculée est inscrite dans la cellule immédiatement au-dessus du tableau `LISTE`.

Exécution de sous-programmes

Lorsque vous indiquez une désignation de fonction (FC) ou de bloc fonctionnel (FB), le bloc de code ayant été déclaré avec cet identificateur est appelé[1]. Si des paramètres formels ont été déclarés dans la déclaration du bloc de code, vous pouvez leur affecter des opérandes effectifs lorsque vous les appelez.

Tous les paramètres qui figurent dans les blocs de déclaration

`VAR_INPUT, VAR_OUTPUT et VAR_IN_OUT`

d'un bloc de code sont appelés paramètres formels. On appelle par contre paramètres effectifs, les paramètres correspondants appelés dans la section des instructions.

La transmission des paramètres effectifs aux paramètres formels fait partie de l'appel.

Voici un exemple typique de traitement de sous-programme :

```
FC31(X:=5, S1:=Somme_chiffres);
```

1 Pour les FC, l'affectation des paramètres effectifs est indispensable, pour les FB elle est optionnelle.

7.11 Blocs SCL

Présentation

Dans une source SCL, vous pouvez programmer de 1 à n blocs comme texte source.

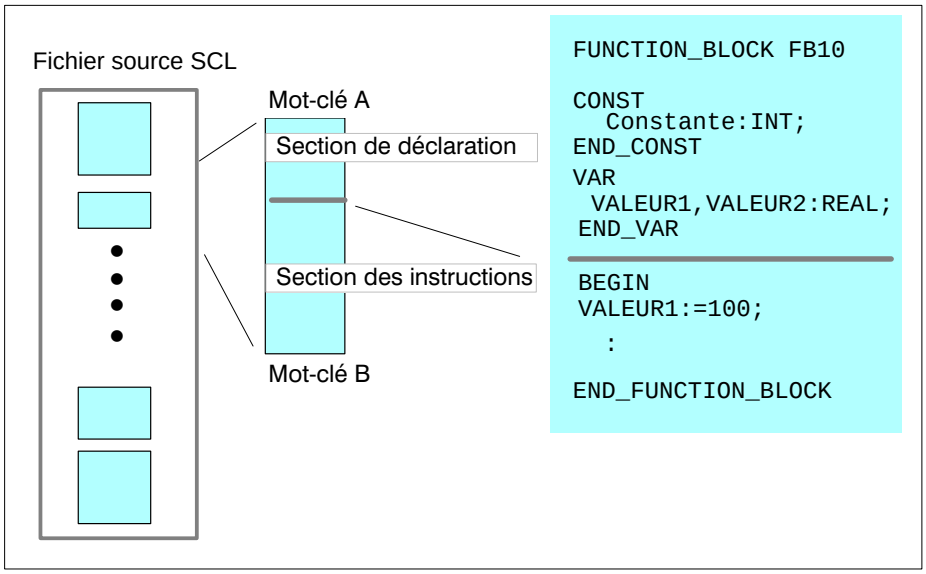
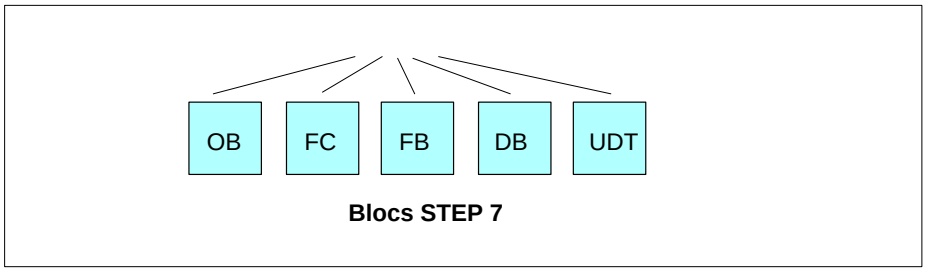


Figure 7-6 *Structure d'un fichier source SCL*

Types de blocs

De part leur fonction, leur structure ou leur utilisation, les blocs STEP 7 sont des sections limitées d'un programme utilisateur. SCL vous permet de programmer les blocs suivants :



Blocs prédéfinis

Vous n'êtes pas obligés de programmer vous-même chaque fonction. Il en existe également qui sont prédéfinies. Elles se trouvent dans le système d'exploitation de l'unité centrale ou dans des bibliothèques *(S7lib)* du logiciel de base STEP 7. Vous pouvez les utiliser pour programmer des fonctions de communication, par exemple.

Structure d'un bloc SCL

Tout bloc comporte les sections suivantes :

- en-tête/fin de bloc (mot-clé correspond au type de bloc),
- section de déclaration,
- section des instructions (section d'affectation pour les blocs de données).

Section de déclaration

La section de déclaration sert à énoncer les définitions qui constitueront la base de la section des instructions : définition des constantes, déclaration des variables et des paramètres, par exemple.

Section des instructions

La section des instructions est introduite par le mot-clé `BEGIN` est se termine par un identificateur standard de fin de bloc `END_xxx` (voir paragraphe 8.2).

Chaque instruction se termine par un point-virgule (;) et peut être précédée d'un repère de saut (étiquette) optionnel. La syntaxe de la section des instructions et d'une instruction individuelle est donnée au chapitre 13.

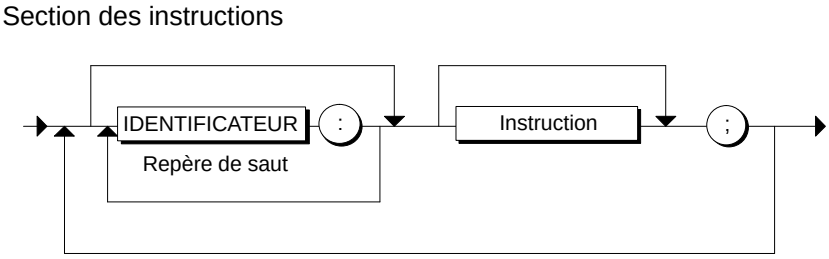


Figure 7-7 *Syntaxe : section des instruction*

Voici un exemple de section des instructions de FB :

```
:                 //Fin de la section de déclaration
:
BEGIN             //Début de la section des instructions
            X := X+1;
REPERE1     Y := Y+10;
            Z := X*Y;
            :
            GOTO REPERE1
REPEREn;          //Fin section des instructions
END_FUNCTION_BLOCK
```

Dans la section des instructions d'un bloc de données, vous pouvez affecter des valeurs par défaut aux données du DB. C'est la raison pour laquelle, la section des instructions d'un DB sera appelée **section d'affectation** dans les chapitres suivants.

Programme S7

Une fois compilés, les blocs créés sont enregistrés dans le classeur des blocs du programme S7 correspondant, d'où vous devez les charger dans la CPU. Vous trouverez des informations à ce sujet dans le guide de l'utilisateur **/231/** .

7.12 Commentaires

Présentation

Les commentaires permettent de documenter un bloc SCL afin de le rendre plus intelligible. Une fois compilés, ils n'interviennent d'aucune façon dans l'exécution du programme. Il existe deux types de commentaires :

- la ligne de commentaire et
- le bloc de commentaire

Ligne de commentaire

Il s'agit d'un commentaire introduit par // qui s'étend jusqu'à la fin de la ligne. Sa longueur est limitée à 253 caractères au maximum, y compris les caractères d'introduction de commentaire //. Il peut être représenté par le diagramme syntaxique suivant :

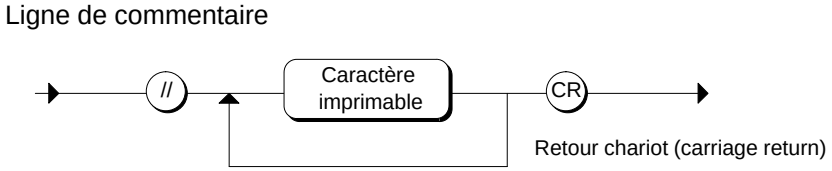


Figure 7-8 *Syntaxe : ligne de commentaire*

La liste de tous les caractères imprimables figure dans le tableau A-2 à l'annexe A. La paire de caractères '(*' et '*)' est insignifiante dans une ligne de commentaire.

Bloc de commentaire

Commentaire pouvant s'étendre sur plusieurs lignes, introduit par '(*' et se terminant par '*)'. Le chevauchement des blocs de commantaire est autorisé de manière standard. Toutefois, vous avez la possibilité de modifier ce paramètre et d'interdire le chevauchement des blocs de commantaire.

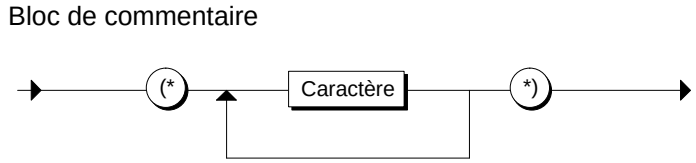


Figure 7-9 *Syntaxe : bloc de commentaire*

La liste des caractères autorisés figure dans le tableau A-2 à l'annexe A.

Règles

Règles de notation :

- Les blocs de commentaire doivent être introduits dans les **blocs de données** comme des lignes de commentaire. Il faut les faire précéder également de deux barres obliques //.
- De manière standard, l'imbrication de commentaires est permise. Ce paramètre du compilateur peut toutefois être modifié par l'option "Commentaires imbriqués". Choisissez à cet effet la commande **Outils ▸ Paramètres**, puis désactivez cette option dans l'onglet "Compilateur" de la boîte de dialogue suivante.
- Un commentaire ne doit interrompre ni un mnémonique, ni une constante. L'interruption de chaînes de caractère est cependant permise.

Voici un exemple de commentaire **erroné** :

```
FUNCTION_(* adaptation *)BLOCK FB10
```

Exemple d'intégration de commentaires

L'exemple suivant comporte deux blocs de commentaire et une ligne de commentaire :

```
FUNCTION_BLOCK FB 15
(* Ceci est un bloc de commentaire pouvant s'étendre
sur plusieurs lignes*)
VAR
    COMMUTATEUR: INT; // Ligne de commentaire
END_VAR;
BEGIN
    (* Affecter une valeur à la variable COMMUTATEUR *)
    COMMUTATEUR:= 3;
END_FUNCTION_BLOCK
```

Figure 7-10 *Exemple de commentaire*

Nota

Les lignes de commentaire suivant immédiatement la section de déclaration d'un bloc de données sont conservées lors d'une décompilation en programme LIST.

Ces commentaires se trouvent dans LIST dans la partie de l'interface, c'est-à-dire dans la partie supérieure de la fenêtre (voir également **/231/**).

Dans l'exemple de la figure 7-10, la première ligne de commentaire est donc conservée.

Structure d’un fichier source SCL

8

Présentation

Un fichier source SCL contient principalement du texte au kilomètre. Vous pouvez y programmer plusieurs blocs, à savoir des OB, FB, FC, DB ou UDT.

Le présent chapitre décrit la structure externe des blocs, leur structure interne étant décrite dans les chapitres suivants.

Structure du chapitre

8.1 Structure

Présentation

Un fichier source SCL contient le texte source de 1 à n blocs (FB, FC, OB, DB ou UDT).

Afin de pouvoir compiler votre fichier source SCL en blocs individuels, vous devez tenir compte de la structure et des règles syntaxiques propres à ces blocs.

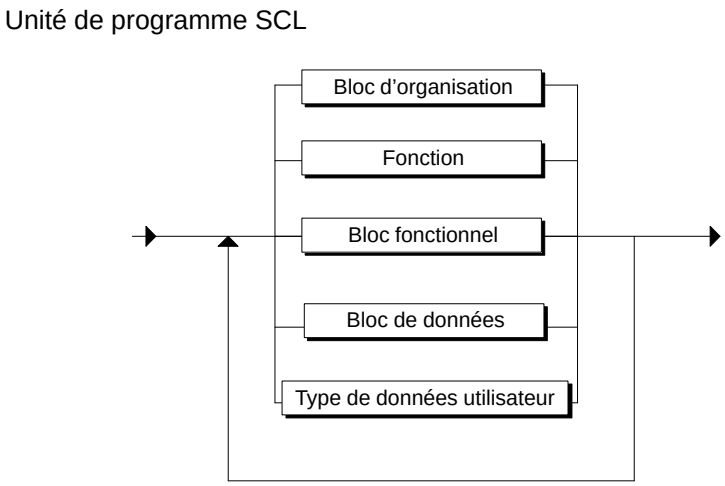


Figure 8-1 *Syntaxe : unité de programme SCL*

Ordre des blocs

Lorsque vous écrivez le fichier source, vous devez tenir compte des règles suivantes quant à l'ordre des blocs :

Un bloc appelé doit précéder le bloc appelant.

Ceci signifie que :

- Un type de données utilisateur (UDT) précède les blocs qui l'utilisent.
- Un bloc de données avec un type de données utilisateur (UDT) affecté suit cet UDT.
- Un bloc de données auquel tout bloc de code peut accéder précède le bloc qui l'appelle.
- Un bloc de données avec un bloc fonctionnel affecté suit ce bloc fonctionnel.
- L'OB1 qui appelle d'autres blocs doit être le dernier bloc. Les blocs appelés par d'autres blocs, eux-mêmes appelés par l'OB1 doivent les précéder.

Les blocs que vous appelez dans un fichier source sans pour autant les programmer dans le même fichier source doivent se trouver dans le programme utilisateur correspondant au moment de la compilation du fichier.

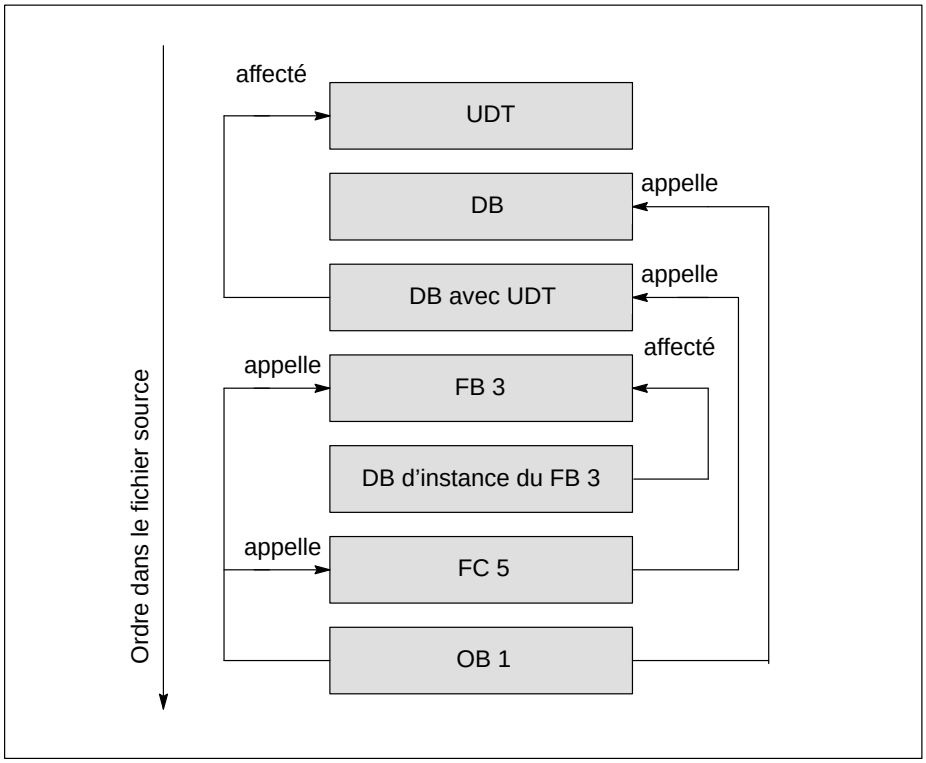


Figure 8-2 Structure de bloc d'un fichier source (exemple)

Structure générale des blocs

Le code source d'un bloc comprend en principe les sections suivantes :

- début de bloc avec désignation du bloc (absolue ou symbolique),
- attributs de bloc (facultatif),
- section de déclaration (diffère selon le type de bloc),
- section d'instructions dans les blocs de code ou
  affectation de valeurs actuelles dans les blocs de données (facultatif).
- fin de bloc.

8.2 Début et fin de bloc

Présentation

Le texte source de chaque bloc est introduit selon le type de bloc par un identificateur standard de début de bloc et la désignation de bloc ; il est bouclé par un identificateur standard de fin de bloc (voir tableau 8-1).

Tableau 8-1 Identificateurs standard pour le début et la fin de bloc

Syntaxe

Syntaxe	Type de bloc	Désignation

`ORGANIZATION_BLOCK` *`nom_ob`*
`:`
`END_ORGANIZATION_BLOCK`
	OB	Bloc d'organisation

`FUNCTION` *`nom_fc:type_fonction`*
`:`
`END_FUNCTION`
	FC	Fonction

`FUNCTION_BLOCK` *`nom_fb`*
`:`
`END_FUNCTION_BLOCK`
	FB	Bloc fonctionnel

`DATA_BLOCK` *`nom_db`*
`:`
`END_DATA_BLOCK`
	DB	Bloc de données

`TYPE` *`nom_udt`*
`:`
`END_TYPE`
	UDT	Type de données utilisateur

Désignation d'un bloc

Dans le tableau 8-1, *nom_xx* correspond à la désignation des blocs. Elle répond aux règles syntaxiques suivantes :

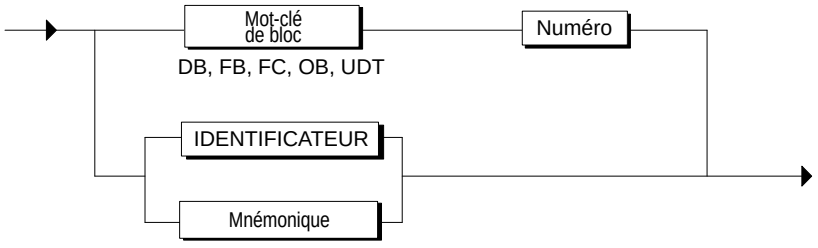


Figure 8-3 *Syntaxe : désignation d'un bloc*

Vous trouverez des informations supplémentaires au paragraphe 7.5. Vous devez définir les identificateurs ou mnémoniques dans la table des mnémoniques de STEP 7 (voir /**231**/).

Exemple

```
FUNCTION_BLOCK FB10
FUNCTION_BLOCK Bloc_régulateur
FUNCTION_BLOCK "Régulateur.B1&U2"
```

8.3 Attributs de bloc

Définition

Il peut s'agir :

- d'attributs de blocs,
- d'attributs système de blocs.

Attributs de blocs

Vous pouvez indiquer le titre d'un bloc, sa version, sa protection, son auteur, son nom et sa famille en utilisant des mots-clés.

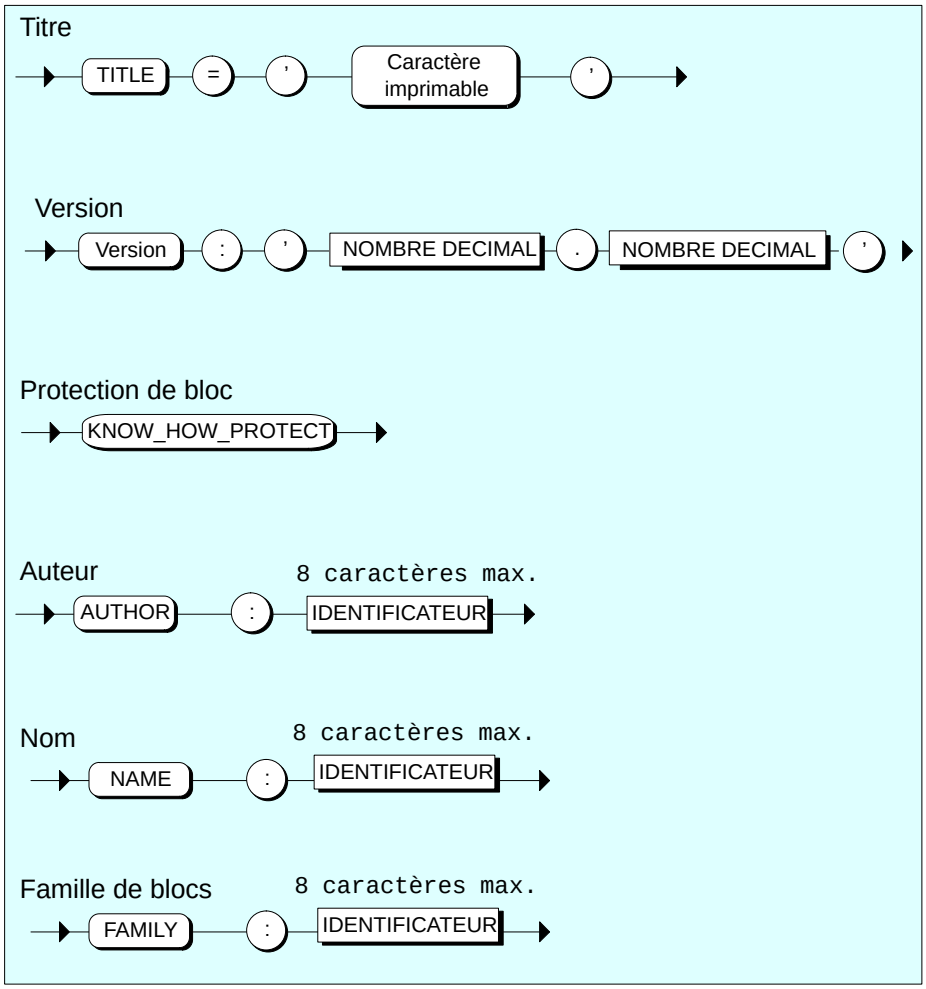


Figure 8-4 *Syntyxe : attributs de bloc*

Attributs système de blocs

Vous pouvez en outre affecter des attributs système aux blocs, par exemple pour la configuration de systèmes de conduite.

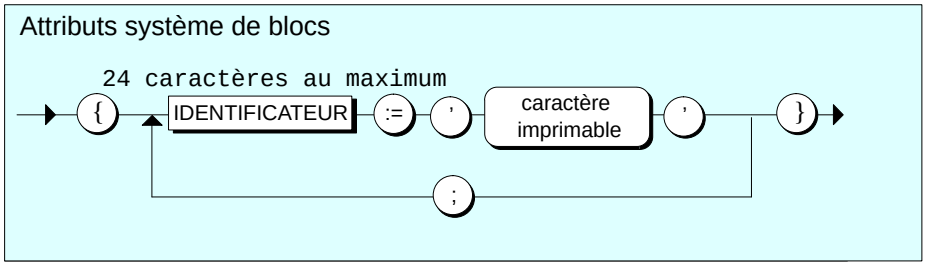


Figure 8-5 *Syntyxe : attributs système de bloc*

Le tableau 8-2 indique les attributs système que vous pouvez affecter aux blocs dans SCL.

Tableau 8-2 Attributs système de blocs

Attribut	Valeur	Attribut affecté lorsque	Type de bloc autorisé
S7_m_c	true, false	le bloc doit être commandé et contrôlé depuis un dispositif de contrôle-commande.	FB
S7_tasklist	taskname1, taskname2, etc.	le bloc doit être appelé non seulement dans des blocs d'organisation cycliques, mais aussi dans d'autres OB (par exemple OB d'erreur ou de démarrage).	FB, FC
S7_block-view	big, small	le bloc doit être représenté en grand ou petit format sur un dispositif de contrôle-commande.	FB, FC

Affectation d'attributs

Vous affectez les attributs **après** la désignation du bloc et **avant** la section de déclaration.

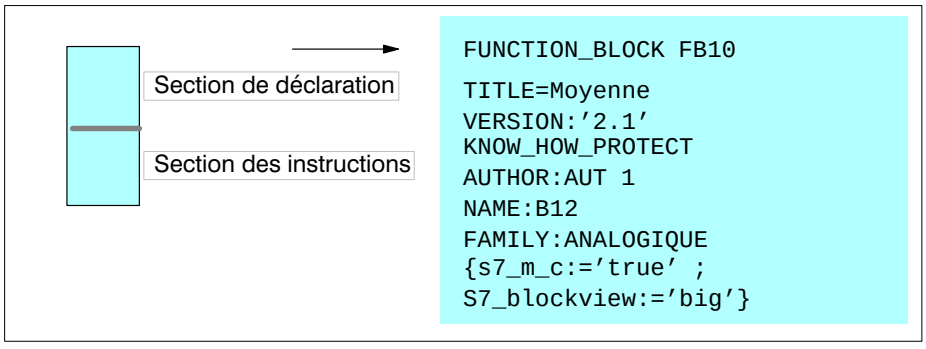


Figure 8-6 *Affectation d'attributs*

8.4 Section de déclaration

Présentation

La section de déclaration sert à définir les variables locales et globales, les paramètres, les constantes et les repères de saut.

- Dans la section de déclaration d'un bloc de code, vous définissez les variables locales, les paramètres, les constantes et les repères de saut qui ne doivent être valables que pour ce bloc de code.
- Dans la section de déclaration d'un DB, vous définissez les données globales auxquelles tout bloc de code doit pouvoir accéder.
- Dans la section de déclaration d'un UDT, vous définissez un type de données utilisateur.

Structure

La section de déclaration se divise elle-même en plusieurs sections de déclaration caractérisées chacune par une paire de mots-clés. Elles contiennent une liste de déclaration pour des données de même type, comme par exemple des constantes, des repères de saut, des données statiques et des données temporaires. Chacune d'entre elles doit être unique et ne peut pas être utilisée dans chaque type de bloc, comme le montre le tableau 8-3. L'ordre des blocs est quelconque.

Sections de déclaration

Tableau 8-3 Sections de déclaration possibles

Données	Syntaxe	FB	FC	OB	DB	UDT

Constantes	`CONST`
	Liste de déclaration
	`END_CONST`
		X	X	X		

Repères de saut	`LABEL`
	Liste de déclaration
	`END_LABEL`
		X	X	X		

Variables temporaires	`VAR_TEMP`
	Liste de déclaration
	`END_VAR`
		X	X	X		

Variables statiques	`VAR`
	Liste de déclaration
	`END_VAR`
		X	X[2]		X[1]	X[1]

Paramètres d'entrée	`VAR_INPUT`
	Liste de déclaration
	`END_VAR`
		X	X			

Paramètres de sortie	`VAR_OUTPUT`
	Liste de déclaration
	`END_VAR`
		X	X			

Paramètres d'entrée/sortie	`VAR_IN_OUT`
	Liste de déclaration
	`END_VAR`
		X	X			

Liste de déclaration :	liste des identificateurs du type à déclarer.					

1 *Pour les DB et UDT, les mots-clés VAR et END_VAR sont remplacés par STRUCT et END_STRUCT.*

2 La déclaration de variables entre les mots-clés VAR et END_VAR est autorisée pour les fonctions, cependant lors de la compilation, les déclarations sont décalées dans la zone temporaire.

Attributs système de paramètres

Vous pouvez en outre affecter des attributs système aux paramètres d'entrée, de sortie ou d'entrée/sortie, par exemple pour la configuration de messages ou liaisons.

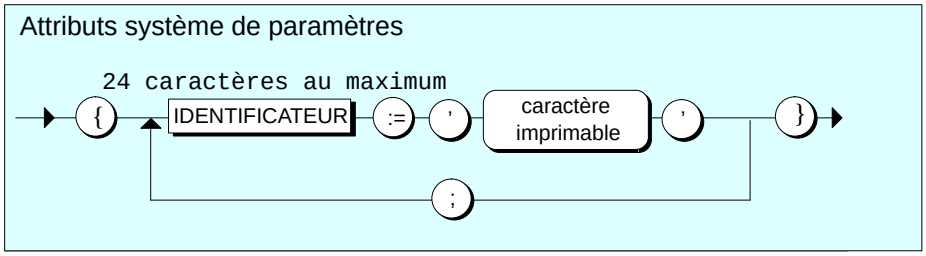


Figure 8-7 *Syntyxe : attributs système de paramètres*

Le tableau 8-4 indique les attributs système que vous pouvez affecter aux paramètres.

Tableau 8-4 Attributs système de paramètres

Attribut	Valeur	Attribut affecté lorsque	Type de déclaration autorisé
S7_server	connection, alarm_archiv	le paramètre est significatif pour la configuration de liaisons ou de messages. Ce paramètre contient le numéro de liaison ou de message.	IN
S7_a_type	alarm, alarm_8, alarm_8p, alarm_s, notify, ar_send	vous définissez le type du bloc de message appelé dans la section d'instructions (condition : l'attribut S7_server:=alarm_archiv est également attribué).	IN, uniquement dans FB
S7_co	pbkl, pbk, ptpl, obkl, fdl, iso, pbks, obkv	vous définissez le type de la liaison à configurer (condition : l'attribut S7_server:=connection est également attribué).	IN
S7_m_c	true, false	le paramètre doit être commandé et contrôlé depuis un dispositif de contrôle-commande.	IN/OUT/ IN_OUT, uniquement dans FB
S7_shortcut	2 beliebige Zeichen, z. B. W, Y	des raccourcis doivent être affectés au paramètre pour l'exploitation de valeurs analogiques.	IN/OUT/ IN_OUT, uniquement dans FB
S7_unit	Einheit, z. B. Liter	des unités doivent être affectées au paramètre pour l'exploitation de valeurs analogiques.	IN/OUT/ IN_OUT, uniquement dans FB
S7_string_0	16 beliebige Zeichen, z. B. AUF	du texte doit être affecté au paramètre pour l'exploitation de valeurs binaires.	IN/OUT/ IN_OUT, uniquement dans FB, FC
S7_string_1	16 beliebige Zeichen, z. B. ZU	du texte doit être affecté au paramètre pour l'exploitation de valeurs binaires.	IN/OUT/ IN_OUT, uniquement dans FB, FC
S7_visible	true, false	le paramètre doit être affiché dans CFC ou pas.	IN/OUT/IN_OUT, uniquement dans FB, FC
S7_link	true, false	le paramètre doit pouvoir être connecté dans CFC ou pas.	IN/OUT/IN_OUT, uniquement dans FB, FC

Tableau 8-5 Attributs système de paramètres

Attribut	Valeur	Attribut affecté lorsque	Type de déclaration autorisé
S7_dyna-mic	true, false	le paramètre doit pouvoir être dynamisé dans CFC lors du test ou pas	IN/IN_OUT, uniquement dans FB, FC
S7_param	true, false	le paramètre doit pouvoir être paramétré dans CFC ou pas.	IN/IN_OUT, uniquement dans FB, FC

Affectation d’attributs

Vous affectez les attributs système de paramètres dans les blocs de déclaration paramètres d’entrée, paramètres de sortie ou paramètres d’entrée/sortie.

Exemple :

```
VAR_INPUT
   in1  {S7_server:=’alarm_archiv’;
        S7_a_type:=’ar_send’}:DWORD;
END_VAR
```

8.5 Section des instructions

Présentation

La section des instructions contient les **instructions**[1] :

- qui doivent être exécutées après l'appel d'un bloc de code. Ces instructions servent à traiter des données et des adresses.
- qui servent à définir des valeurs par défaut individuelles dans des blocs de données.

La figure 8-8 illustre la syntaxe de la section des instructions. Elle consiste en une suite des instructions individuelles, chacune pouvant être précédée d'un repère de saut optionnel (voir paragraphe 11.6) qui correspond au branchement d'une instruction de saut.

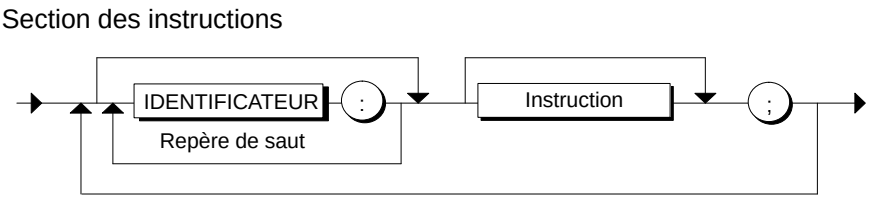


Figure 8-8 *Syntaxe : section des instructions*

Voici des exemples d'instructions autorisées :

```
BEGIN
            VALEUR_INIT :=0;
            VALEUR_FIN  :=200;
:
ENREGISTRER: RESULTAT   :=VALEUR_CONSIGNE;
:
```

Important

Lorsque vous formulez une instruction, assurez-vous que :

- la section des instructions commence facultativement par le mot-clé `BEGIN,`
- la section des instructions se termine par le mot-clé de fin de bloc,
- chaque instruction se termine par un point-virgule,
- tous les identificateurs utilisés dans la section des instructions ont été déclarés.

1 *Le terme « instruction » désigne une structure de déclaration de fonction exécutable.*

8.6 Instruction

Présentation

Chaque instruction comporte des :

- **affectations de valeurs** permettant d’attribuer à une variable, soit une valeur, soit le résultat d’une expression ou encore la valeur d’une autre variable.
- **instructions de contrôle** servant à répéter des instructions ou groupes d’instructions dans un programme ou encore à y réaliser des branchements.
- **exécutions de sous-programmes** destinés à appeler des fonctions et blocs fonctionnels.

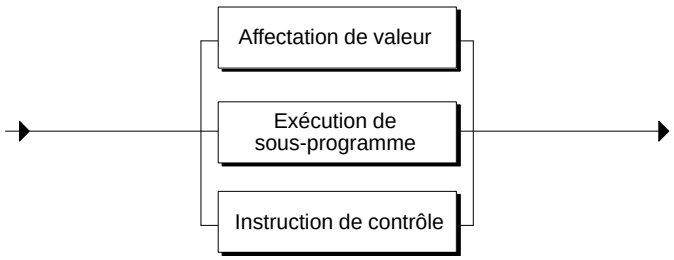


Figure 8-9 *Syntaxe : instruction*

Les éléments nécessaires à la formulation de ces instructions sont des expressions, des opérateurs et des opérandes. Ils seront traités dans les chapitres suivants.

Exemples

Les exemples suivants illustrent les divers types d’instructions :

```
// Exemple d’affectation de valeur
      VAL_MES:= 0 ;

// Exemple d’exécution d’un sous-programme
      FB1.DB11(VALIDATION:= 10) ;

// Exemple d’instruction de contrôle
      WHILE COMPTEUR < 10 DO... ;
      :
      END_WHILE
```

Exemple 8-1 *Instructions*

8.7 Structure d'un bloc fonctionnel (FB)

Présentation

Un bloc fonctionnel (FB) est un bloc de code qui contient une partie d'un programme et qui dispose d'une zone de mémoire qui lui est affectée. A chaque fois que vous appelez un FB, vous devez lui affecter un DB d'instance (voir chapitre 10) dont vous définissez la structure dans la section de déclaration du FB.

Bloc fonctionnel

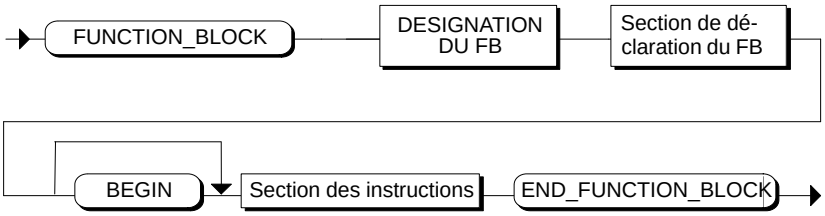


Figure 8-10 *Syntaxe : bloc fonctionnel (FB)*

Désignation du FB

Comme désignation du FB, indiquez après le mot-clé

```
FUNCTION_BLOCK
```

le mot-clé FB, suivi du numéro de bloc ou du mnémonique du FB.

Exemples :

```
FUNCTION_BLOCK FB10
FUNCTION_BLOCK MOTEUR_1
```

Section de déclaration du FB

La section de déclaration du FB permet de déclarer les données spécifiques au bloc. Les sections de déclaration autorisées sont indiquées dans le paragraphe 8.4. Dans la section de déclaration, vous devez également définir la structure du DB d'instance affecté au FB.

Exemples :

```
CONST
   CONSTANTE:=5;
END_CONST

VAR
   VALEUR1,VALEUR2,VALEUR3:INT;
END_VAR
```

Exemple

L'exemple 8-2 représente le code source d'un bloc fonctionnel. Des valeurs initiales ont été affectées aux paramètres d'entrée et de sortie (V1, V2).

```
FUNCTION_BLOCK FB11
      VAR_INPUT
                  V1: INT:= 7;
      END_VAR
      VAR_OUTPUT
                  V2: REAL;
      END_VAR
      VAR
                  PASSAGE_1:INT;
      END_VAR
      BEGIN
      IF V1 = 7 THEN
      PASSAGE_1:= V1;
      V2:= FC2 (VALEURTEST:= PASSAGE_1);
      //Appel de la fonction FC2 avec
      //affectation des paramètres par la
      //variable statique PASSAGE_1
      END_IF;
END_FUNCTION_BLOCK
```

Exemple 8-2 *Exemple d'un bloc fonctionnel (FB)*

8.8 Structure d'une fonction (FC)

Présentation

Une fonction (FC) est un bloc de code auquel aucune zone de mémoire individuelle n'est affectée. Elle n'a donc pas besoin de DB d'instance. Contrairement à un FB, une fonction peut fournir un résultat (**valeur en retour**) au point appelant. Elle peut de ce fait être utilisée comme une variable dans une expression. Les fonctions de type VOID ne donnent pas de valeur en retour.

Fonction

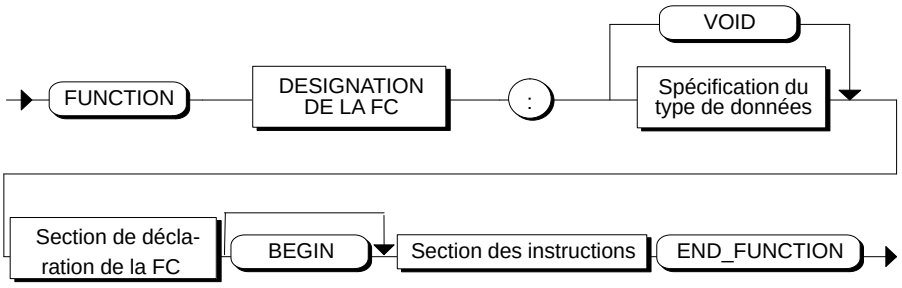


Figure 8-11 *Syntaxe : fonction (FC)*

Désignation de la FC

Comme désignation de la FC, indiquez après le mot-clé

```
FUNCTION
```

le mot-clé FC, suivi du numéro de bloc ou du mnémonique de la FC.

Exemples :

```
FUNCTION FC100
```

```
FUNCTION REGIME
```

Spécification du type de données

Indiquez le type de données de la valeur en retour. Tous les types de données décrits au chapitre 9 sont autorisés, à l'exception des types de données STRUCT et ARRAY. L'indication du type de données s'avère inutile lorsque vous renoncez à la valeur en retour en utilisant l'attribut **VOID**.

Section de déclaration de la FC

Les sections de déclaration autorisées sont indiquées au paragraphe 8.4.

Section des instructions

Dans la section des instructions, vous devez affecter le **résultat de la fonction** au nom de la fonction. Voici un exemple d'instruction correcte pour la fonction FC31 :

```
FC31:= VALEUR;
```

Exemple

L'exemple représente une fonction avec les paramètres d'entrée formels x1, x2, y1, y2, un paramètre de sortie formel Q2 et une valeur en retour FC11.

La signification des paramètres formels est décrite au chapitre 10.

```
FUNCTION FC11: REAL
      VAR_INPUT
                  x1: REAL;
                  x2: REAL;
                  y1: REAL;
                  y2: REAL;
      END_VAR
      VAR_OUTPUT
                  Q2: REAL;
      END_VAR
      BEGIN          // Section des instructions
      FC11:= SQRT    // Affectation de la
                     // valeur de la fonction
      ( (x2 - x1)**2 + (y2 - y1) **2 );
      Q2:= x1;
END_FUNCTION
```

Exemple 8-3 *Exemple d'une fonction (FC)*

8.9 Structure d’un bloc d’organisation (OB)

Présentation

Un bloc d’organisation (OB) fait partie du programme utilisateur, comme un FB ou une FC. Le système d’exploitation l’appelle cycliquement ou lors d’événements donnés. Il constitue l’interface entre le programme utilisateur et le système d’exploitation.

Bloc d’organisation

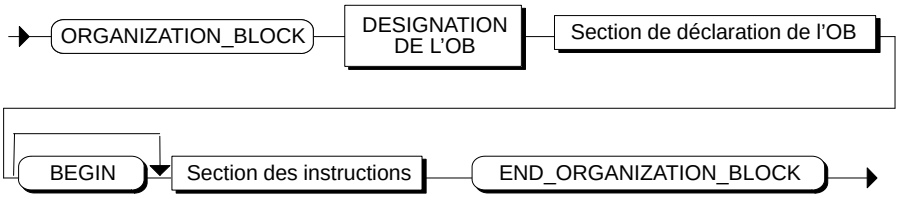


Figure 8-12 *Syntaxe : bloc d’organisation (OB)*

Désignation de l’OB

Comme désignation de l’OB, indiquez après le mot-clé

```
ORANIZATION_BLOCK
```

le mot-clé OB, suivi du numéro de bloc ou du mnémon

Exemples :

ique de l’OB.

```
ORGANIZATION_BLOCK OB14
ORGANIZATION_BLOCK ALARME_HORAIRE
```

Section de déclaration de l’OB

L’exécution de chaque OB requiert en principe **20 octets de données locales** pour une information de départ. Vous pouvez déclarer des variables temporaires supplémentaire dans l’OB, si c’est nécessaire pour le programme. La description des 20 octets de données locales est faite dans le manuel **/235/**.

Exemple

```
ORGANIZATION_BLOCK OB14

//ALARME HORAIRE
VAR_TEMP
HEADER:ARRAY [1..20] OF BYTE;// 20 octets pour
                             // l’information de départ
:
:
END_VAR
```

Les autres sections de déclaration autorisées pour les OB sont indiquées au paragraphe 8.4.

8.10 Structure d'un bloc de données (DB)

Description

Un bloc de données (DB) contient des données utilisateur globales, auxquelles **tous** les blocs d'un programme peuvent accéder. Tout(e) FB, FC ou OB peut lire ou écrire dans ces DB. La structure des blocs de données (DB d'instance) qui ne sont affectés qu'à certains FB est donnée au chapitre 12.

Bloc de données

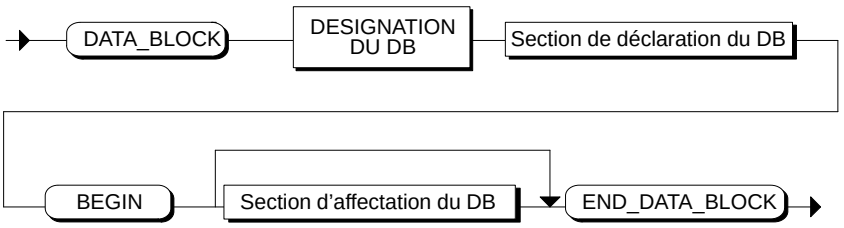


Figure 8-13 *Syntaxe : bloc de données (DB)*

Désignation du DB

Comme désignation du DB, indiquez après le mot-clé

```
DATA_BLOCK
```

le mot-clé DB, suivi du numéro de bloc ou du mnémonique du DB.

Exemple

```
DATA_BLOCK DB20
```

```
DATA_BLOCK ETENDUE_MESURE
```

Section de déclaration du DB

Dans la section de déclaration du DB, vous définissez la structure des données du DB. Vous pouvez affecter le type de données structure (STRUCT) ou le type de données utilisateur (UDT) à une variable du DB.

Section de déclaration du DB

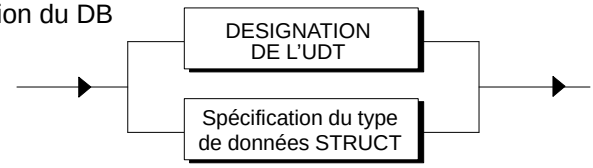


Figure 8-14 *Syntaxe : section de déclaration du DB*

Exemple

```
DATA_BLOCK DB20
       STRUCT      // Section de déclaration
         VALEUR:ARRAY [1..100] OF INT;
       END_STRUCT

BEGIN              //Début de la section des instructions
:
END_DATA_BLOCK     //Fin du bloc de données
```

Section d'affectation du DB

Vous pouvez affecter des valeurs spécifiques pour votre application particulière aux données que vous avez déclarées dans la section de déclaration. La section d'affectation commence par le mot-clé :

```
BEGIN
```

puis comprend une suite d'affectations de valeurs avec la syntaxe suivante :

Section d'affectation du DB

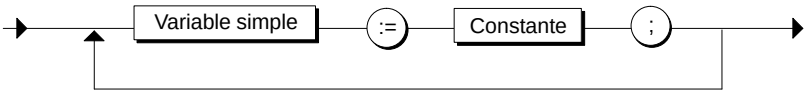


Figure 8-15 *Syntaxe : section d'affectation d'un DB*

Nota

Il convient de respecter la syntaxe de LIST lors de l'attribution des valeurs initiales (initialisation) et de l'indication d'attributs et de commentaires au sein du DB. Pour plus d'informations sur les différentes notations de constantes, attributs et commentaires, reportez-vous aux manuels **/231/** ou **/232/**.

Exemple

L'exemple suivant montre comment formuler la section d'affectation si les composantes [1] et [5] du tableau doivent prendre les valeurs entières 5 et -1 à la place de la valeur par défaut 1.

```
DATA_BLOCK        DB20
SRTUCT               //Déclaration de données
                     //avec valeurs par défaut
      VALEUR      :  ARRAY [ 1..100] OF INT := 100 (1);
      MEMENTO     :  BOOL   := TRUE;
      S_MOT       :  WORD   := W#16#FFAA;
      S_OCTET     :  BYTE   := B#16#FF
      S_TIME      :  S5TIME := S5T#1h30m30s
END_STRUCT

BEGIN                //Section d'affectation
      //Affectation de valeurs
      //à diverses composantes du tableau
      VALEUR [1]  := 5;
      VALEUR [5]  :=-1;
END_DATA_BLOCK
```

Exemple 8-4 Section d'affectation d'un DB

8.11 Structure d’un type de données utilisateur (UDT)

Présentation

Un type de données utilisateur (UDT) est une structure de données particulière que vous créez vous-même. Puisqu’il possède un nom, il peut être utilisé plusieurs fois. Il peut par définition être utilisé dans l’ensemble du programme de la CPU et correspond donc à un type de données globales. Vous pouvez donc utiliser ce type de données :

- comme type de données simple ou complexe dans un bloc ou
- comme modèle pour créer des blocs de structure de données identique.

Type de données utilisateur

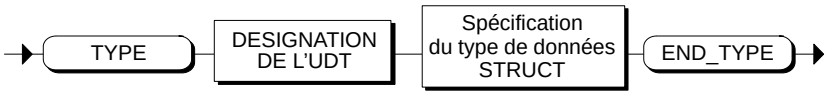


Figure 8-16 *Syntaxe : type de données utilisateur (UDT)*

Désignation de l’UDT

Comme désignation de l’UDT, indiquez après le mot-clé

```
TYPE
```

le mot-clé UDT, suivi du numéro de l’UDT ou de son mnémonique.

Exemples

```
TYPE UDT 10
TYPE SECTION_AFFECTATION
```

Spécification des types de données

Elle est toujours réalisée à l’aide de la **spécification du type de données STRUCT**. Vous pouvez utiliser le type de données UDT dans les sections de déclaration de blocs de code ou de blocs de données, ou alors l’affecter à des DB. Les sections de déclaration autorisées et de plus amples informations à ce sujet sont données au chapitre 9.

Types de données

9

Présentation

Un type de données regroupe les plages de valeurs et les opérations en une unité. Comme la plupart des langages de programmation, SCL possède des types de données prédéfinis (intégrés au langage). Le programmeur a en outre la possibilité de créer des types de données complexes et des types de données utilisateur.

Structure du chapitre

9.1 Présentation

Généralités

Le tableau 9-1 indique les types de données utilisés dans SCL.

Tableau 9-1 Type de données dans SCL

Types de données simples			
BOOL BYTE WORD DWORD	CHAR	INT DINT REAL	TIME DATE TIME_OF_DAY S5TIME
Types de données complexes			
DATE_AND_TIME	STRING	ARRAY	STRUCT
Types de données utilisateur			
UDT			
Types de paramètres			
TIMER COUNTER	BLOCK_FB BLOCK_FC BLOCK_DB BLOCK_SDB	POINTER	ANY

Ces types de données définissent :

- le type et la signification des éléments de données,
- les plages autorisées des éléments de données,
- le nombre d'opérations pouvant être exécutées avec un opérande d'un type de données et
- la notation des données de ce type.

9.2 Types de données simples

Présentation

Un type de données simple définit la structure de données qui ne peuvent pas être divisées en éléments plus petits. Il répond à la définition de la norme DIN EN 1131-3. Un type de données simple décrit une zone de mémoire de longueur fixe et correspond à une grandeur exprimée sous forme binaire, entière, réelle, sous forme de durée, d'heure ou de caractère. Tous ces types de données sont prédéfinis dans SCL.

Tableau 9-2 Longueur en bits et plage des valeurs des types de données simples

Type	Mot-clé	Longueur en bits	Plage des valeurs
Type de données binaire	Les données de ce type occupent 1 bit (type de données BOOL), 8 bits, 16 bits ou 32 bits.		
Bit	BOOL	1	`0, 1 ou FALSE (FAUX), TRUE (VRAI)`
Octet	BYTE	8	Il n'est pas possible d'indiquer une plage de valeurs numérique. Il s'agit ici de combinaisons binaires avec lesquelles il est impossible de former des expressions numériques.
Mot	WORD	16	
Double mot	DWORD	32	
Type de données caractère	Les données de ce type occupent exactement 1 caractère du jeu de caractères ASCII		
Caractère unique	CHAR	8	Jeu de caractères ASCII étendu
Types de données numériques	Ils permettent de traiter des valeurs numériques.		
Entier (nombre entier)	INT	16	`-32_768` à `32_767`
Entier double	DINT	32	`-2_147_483_648` à `2_147_483_647`
Virgule flottante (nombre à virgule flottante IEE)	REAL	32	-3.402822E+38 à -1.175495E-38, 0.0, +1.175495E-38 à 3.402822E+38
Types de données temps	Les données de ce type représentent les diverses valeurs de temps et de date dans STEP 7.		
S5TIME	S5TIME	16	T#0H_0M_0S_10MS à T#2H_46M_30S
Durée : Durée CEI par pas de 1 ms	TIME (=DURATION)	32	`–T#0D_0H_0M_0S_0MS` à `T#24D_20H_31M_23S_647MS`
Date : Date CEI par pas de 1 jour	DATE	16	`D#1990-01-01` à `D#2168-12-31`
Heure du jour : Heure par pas de 1 ms	TIME_OF_DAY (=TOD)	32	TOD#0:0:0 à TOD#23:59:59.999

Remarque relative à S5TIME : selon la base de temps choisie – 0.01S, 0.1S, 1S ou 10S – la résolution de la valeur de temps est limitée. Le compilateur arrondit les valeurs en conséquence.

9.3 Types de données complexes

Présentation

SCL autorise les types de données complexes suivants :

Tableau 9-3 Types de données complexes

Type de données	Description
DATE_AND_TIME DT	Définit une zone de 64 bits (8 octets). Ce type de données prédéfini dans SCL enregistre la date et l'heure (en format décimal codé binaire).
STRING	Définit une zone pour une chaîne de 254 caractères au maximum (type de données (CHAR).
ARRAY	Définit un tableau formé de composantes d'un même type de données (simples ou complexes).
STRUCT	Définit un groupe formé d'une combinaison quelconque de types de données. Vous pouvez définir un tableau composé de structures ou une structure composée d'autres structures et de tableaux.

9.3.1 Type de données DATE_AND_TIME

Présentation

Le type de données DATE_AND_TIME est composé des types de données DATE et TIME et définit une zone de 64 bits (8 octets) pour l'indication de la date et de l'heure. Cette zone de données enregistre les informations suivantes :
année-mois-jour-heures:minutes:secondes.millisecondes

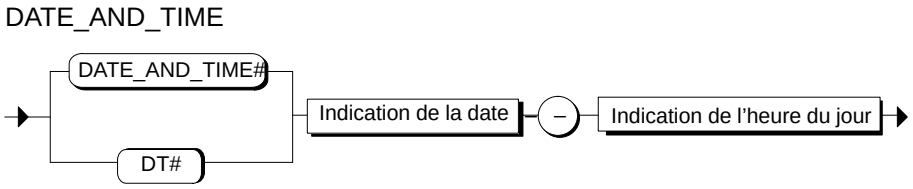


Figure 9-1 *Syntaxe : DATE_AND_TIME*

Plage des valeurs

Tableau 9-4 Longueur en bits et plage des valeurs

Type	Mot-clé	Longueur en bits	Plage des valeurs
Date et heure	DATE_AND_TIME (=DT)	64	`DT#1990-01-01:0:0:0.0` à `DT#2089-12-31:23:59:59.999`

La syntaxe exacte pour indiquer la date et l'heure du jour est donnée au chapitre 11 du présent manuel. Voici la définition correcte du 20.10.1995 12 heures 20 min. 30 s et 10 ms :

DATE_AND_TIME#1995-10-20-12:20:30.10

DT#1995-10-20-12:20:30.10

Nota

Des fonctions standard sont à votre disposition pour vous permettre d'accéder précisément à la composante DATE ou TIME.

9.3.2 Type de données STRING

Présentation

Le type de données STRING définit une chaîne de caractères, dont la longueur maximale est de 254 caractères uniques.

La zone de mémoire standard réservée à une chaîne de caractères est de 256 octets, où vont être enregistrés les 254 caractères ainsi qu'un en-tête de 2 octets.

Vous avez la possibilité de diminuer cette zone de mémoire réservée à une chaîne de caractères, en définissant le nombre maximal de caractères à enregistrer dans la chaîne. Une chaîne de longueur nulle, c'est-à-dire une chaîne sans contenu représente la plus petite valeur possible.

Spécification du type de données STRING

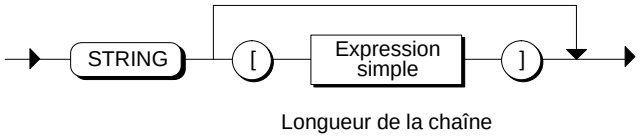


Figure 9-2 *Syntaxe : spécification du type de données STRING*

L'expression simple (longueur de la chaîne) représente le nombre maximal de caractères dans la chaîne.

Voici des types de chaînes de caractères autorisés :

```
STRING[10]
STRING[3+4]
STRING[3+4*5]
```

`STRING` Plage des valeurs max. (par défaut ≙ 254 caractères)

Plage des valeurs

Tous les caractères du code ASCII sont autorisés dans une chaîne de caractères. Le chapitre 11 décrit comment utiliser des caractères de commande et des caractères non imprimables.

Nota

Afin d'optimiser les ressources de votre CPU, vous avez la possibilité de réduire à un nombre quelconque de caractères la zone de 254 caractères réservée de manière standard. Choisissez la commande **Paramètres** dans le menu **Outils**, puis l'onglet "Compilateur" dans la boîte de dialogue suivante. Indiquez ensuite le nombre de caractères voulus pour l'option "Long. de chaîne max.".

9.3.3 Type de données ARRAY

Présentation

Un tableau (type de données ARRAY) comporte un nombre défini de composantes d'un même type de données. Le type de données ARRAY est spécifié plus précisément dans le diagramme de syntaxe 9-3, après le mot réservé `OF`. SCL distingue :

- le type de données ARRAY unidimensionnel.
  Il s'agit d'une liste d'éléments de données disposés dans l'ordre croissant.
- le type de données ARRAY bidimensionnel.
  Il s'agit d'un tableau de données comportant des lignes et des colonnes. La première dimension correspond au numéro de ligne, la seconde au numéro de colonne.
- le type de données ARRAY de dimension supérieure.
  Il s'agit de l'extension du tableau bidimensionnel à d'autres dimensions. Le nombre maximal de dimensions autorisé est de 6.

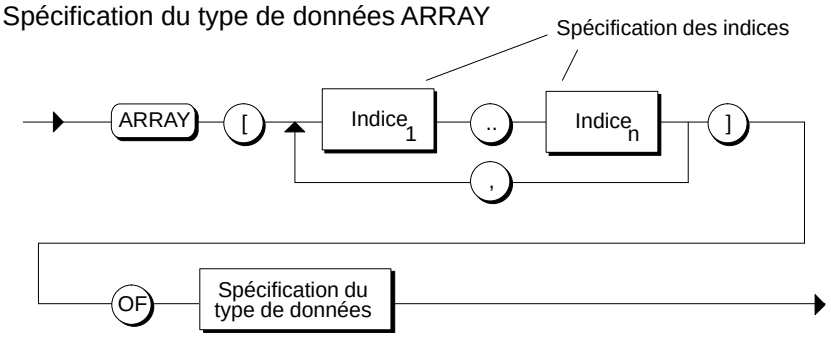


Figure 9-3 *Syntaxe : spécification du type de données ARRAY*

Spécification des indices

Elle décrit les dimensions du tableau :

- indice le plus petit et le plus grand possible (zone d'indexage) pour chaque dimension. Il peut s'agir d'une valeur entière quelconque (-32768 à 32767).
- Les limites doivent être indiquées séparées par deux points.
- Chaque zone d'indexage est séparée par une virgule et l'ensemble de la spécification des indices figure entre crochets.

Spécification du type de données

Elle consiste à déclarer le type de données des composantes du tableau. Tous les types de données décrits dans ce chapitre sont autorisés. Le type de données d'un tableau (ARRAY) peut également être une structure (STRUCT).

Voici des spécifications de tableaux autorisées :

```
ARRAY[1..10] OF REAL
ARRAY[1..10] OF STRUCT..END_STRUCT
ARRAY[1..100, 1..10] OF REAL
```

9.3.4 Type de données STRUCT

Présentation

Le type de données STRUCT décrit une zone comportant un nombre fixe de composantes dont les types de données peuvent être différents. Dans le diagramme syntaxique de la figure 9-4, ces éléments de données sont indiqués dans la déclaration des composantes, immédiatement après le mot-clé STRUCT.

Un élément de données du type de données STRUCT peut lui-même être une structure, rendant ainsi leur imbrication possible.

Au chapitre 10, vous apprendrez à adresser les données d'une structure.

Spécification du type de données STRUCT

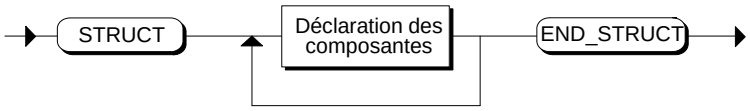


Figure 9-4 *Syntaxe : déclaration du type de données STRUCT*

Déclaration des composantes

Elle correspond à la liste des diverses composantes d'une structure. Comme le montre le diagramme syntaxique 9-5, elle comporte :

- 1 à n identificateurs avec
- le type de données qui leur est affecté et
- des valeurs initiales par défaut optionelles.

Déclaration des composantes

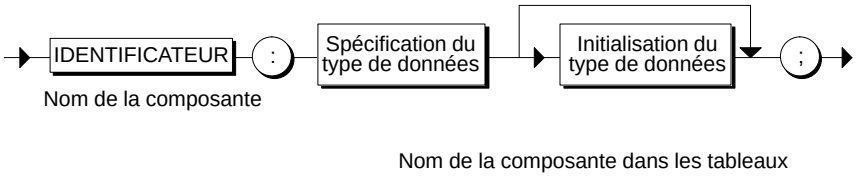


Figure 9-5 *Syntaxe : déclaration des composantes*

Identificateur

Il correspond au nom de l'une des composante de la structure, à laquelle s'applique la spécification du type de données suivante.

Initialisation du type de données

Après la spécification du type de données, vous pouvez optionnellement affecter une valeur initiale à une composante individuelle de la structure. Cette affectation de valeur est décrite au chapitre 10.

Exemple

L'exemple 9-1 montre une définition possible pour un type de données `STRUCT`.

```
STRUCT
//DEBUT de la déclaration des composantes
  A1              :INT;
  A2              :STRING[254];
  A3              :ARRAY [1..12] OF REAL;
   |Nom de la composante  |Spécification du type de données

//FIN de la déclaration des composantes
END_STRUCT
```

Exemple 9-1 Définition d'un type de données `STRUCT`

9.4 Type de données utilisateur (UDT)

Présentation

Vous définissez un type de données utilisateur (UDT) sous forme de bloc, comme décrit au chapitre 8. Une fois défini, un tel type de données peut être utilisé dans l'ensemble du programme utilisateur. Il s'agit donc d'un type de données global. Vous pouvez l'utiliser dans la section de déclaration d'un bloc ou bloc de données en indiquant sa désignation UDTx (x correspond au numéro) ou le mnémonique qui lui est affecté.

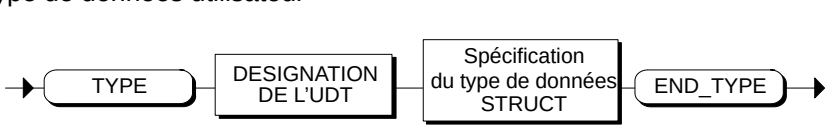


Figure 9-6 *Syntaxe : Type de données utilisateur (UDT)*

Désignation de l'UDT

La déclaration d'un UDT est introduite par le mot-clé TYPE, suivi du nom de l'UDT (identificateur de l'UDT). Comme décrit au chapitre 8, le nom de l'UDT peut être attribué de manière absolue comme nom standard de la forme UDTx (x correspond à un numéro) ou de manière symbolique avec un mnémonique (voir aussi chapitre 8).

Spécification du type de données

La désignation de l'UDT est suivie de la spécification du type de données. Seule la spécification du type de données STRUCT est autorisée (voir chapitre 9.3.4) :

```
STRUCT
:
END_STRUCT
```

Vous devez ensuite clore l'ensemble de la déclaration d'un UDT avec le mot-clé

```
END_TYPE.
```

Utilisation d'un UDT

Vous pouvez vous servir du type de données ainsi défini pour utiliser des variables ou des paramètres, ainsi que pour déclarer des DB. L'UDT vous permet également de déclarer des composantes de structures ou de tableaux, même dans d'autres UDT.

Nota

Il convient de respecter la syntaxe de LIST lors de l'attribution des valeurs initiales (initialisation) au sein de l'UDT. Pour plus d'informations sur les différentes notations de constantes, reportez-vous aux manuels **/231/** ou **/232/**.

Exemple

Cet exemple vous montre comment définir un UDT et utiliser ce type de données dans la déclaration d'une variable. On suppose que le mnémonique « VALEURS_MESURE » a été déclaré pour l'UDT50 dans la table des mnémoniques.

```
TYPE VALEUR_MESURE // Définition de l'UDT
 STRUCT
  BIPOL_1 : INT;
  BIPOL_2 : WORD   := W#16#AFA1;
  BIPOL_3 : BYTE   := B#16#FF;
  BIPOL_4 : WORD   := B#(25,25);
  BIPOL_5 : INT    := 25;
  S_TIME  : S5TIME := S5T#1h20m10s;
  MESURE:  STRUCT
            BIPOLAIRE_10V      : REAL;
            UNIPOLAIRE_4_20MA  : REAL;
           END_STRUCT;
 END_STRUCT
END_TYPE
```

```
FUNCTION_BLOCK
VAR
      ETENDUE_MESURE: VALEURS_MESURE;
END_VAR
BEGIN
...
      ETENDUE_MESURE.BIPOL:= -4;
      ETENDUE_MESURE.MESURE.UNIPOLAIRE_4_20MA:= 2.7;
...
END_FUNCTION_BLOCK;
```

Exemple 9-2 Déclaration de types de données utilisateur

9.5 Types de paramètres

Présentation

Outre les types de données simples, complexes et définis par l'utilisateur, vous pouvez également utiliser ce que l'on appelle des **types de paramètres** pour définir les paramètres de bloc formels des FB et FC. Ces types de données servent à :

- déclarer des fonctions de comptage et de temporisation (`TIMER ou COUNTER`),
- déclarer des FC, FB, DB et SDB comme paramètres (BLOCK_xx),
- autoriser l'utilisation d'un opérande d'un type de données quelconque comme paramètre (`ANY`),
- autoriser l'utilisation d'une zone de mémoire comme paramètre (`POINTER`).

Tableau 9-5 Types de paramètres

Paramètre	Longueur	Description
TIMER	2 octets	Caractérise une temporisation particulière qui doit être utilisée par le programme dans le bloc de code appelé. Paramètre effectif : `p. ex. T1`
COUNTER	2 octets	Caractérise un compteur particulier qui doit être utilisé par le programme dans le bloc de code appelé. Paramètre effectif : `p. ex. Z10`
BLOCK_FB BLOCK_FC BLOCK_DB BLOCK_SDB	2 octets	Caractérise un bloc particulier qui doit être utilisé par le programme dans le bloc de code appelé. Paramètre effectif : `p. ex. FC101` `DB42`
ANY	10 octets	S'utilise lorsqu'un type de paramètres quelconque, à l'exception de ANY, doit être autorisé pour le paramètre effectif.
POINTER	6 octets	Caractérise une zone de mémoire particulière qui doit être utilisée par le programme. Paramètre effectif : `p. ex. M50.0`

TIMER et COUNTER

Vous définissez une temporisation ou un compteur particuliers qui vont être utilisés lors de l'exécution d'un bloc. Les types de paramètres `TIMER` et `COUNTER` ne sont autorisés que pour les paramètres d'entrée (`VAR_INPUT`).

Types de paramètres BLOCK

Vous définissez un bloc particulier qui doit être utilisé comme paramètre d'entrée. C'est la déclaration du paramètre d'entrée qui détermine le type de bloc (FB, FC ou DB). Pour l'affectation du paramètre, vous pouvez indiquer l'identificateur de bloc de manière absolue (par exemple FB20) ou de manière symbolique, avec un mnémonique.

SCL ne propose pas d'opérations sur ces types de paramètres. Eux seuls peuvent être affectés lors d'appels de blocs. Pour les FC, l'indication d'un paramètre d'entrée n'est pas possible.

Vous pouvez affecter dans SCL des paramètres effectifs aux opérandes des types de données suivants :

- blocs fonctionnels sans paramètres formels,
- fonctions sans paramètres formels ni valeur en retour (VOID),
- blocs de données et blocs de données système.

ANY

SCL offre la possibilité de déclarer des paramètres de blocs du type ANY. Lorsque vous appelez un tel bloc, vous pouvez affecter des opérandes de type quelconque (à l'exception de ANY) à ces paramètres. SCL ne propose toutefois qu'une possibilité d'utilisation du type de paramètre `ANY` : la transmission à des blocs de niveau hiérarchique inférieur.

SCL vous permet d'affecter des opérandes avec les types de données suivants comme paramètres effectifs :

- types de données simples :
  vous indiquez l'adresse absolue ou le mnémonique du paramètre effectif.
- types de données complexes :
  vous indiquez le mnémonique des données avec le type de données complexes (par exemple tableaux et structures).
- type de données `ANY` :
  ceci n'est possible que lorsque l'opérande est un type de paramètre compatible avec des paramètres formels.
- type de données `NIL` :
  vous indiquez un pointeur zéro.
- temporisations, compteurs et blocs ; vous indiquez leur numéro (par exemple T1, Z20 ou FB6).

Le type de paramètre `ANY` est autorisé pour les paramètres formels d'entrée, d'entrée/sorties de FB et de FC de même que pour les paramètres de sortie de FC.

Nota

Lorsque lors de l'appel d'un FB ou d'une FC, vous affectez une variable temporaire à un paramètre formel de type ANY, vous ne pouvez pas transmettre ce paramètre à un autre bloc dans le bloc appelé. En effet, les adresses des variables temporaires perdent leur validité lors de la transmission.

POINTER

SCL offre la possisbilité de déclarer des paramètres de bloc du type de données `POINTER`. Lorsque vous appelez un tel bloc, vous pouvez affecter des opérandes d'un type de données quelconque à ces paramètres. SCL ne propose toutefois qu'une instruction de traitement du type de données `POINTER` : la transmission à des blocs de niveau hiérarchique inférieur.

Comme paramètres effectifs, vous pouvez affecter des opérandes des types de données suivants dans SCL :

- types de données simples :
  vous indiquez l'adresse absolue ou le mnémonique du paramètre effectif.
- types de données complexes :
  vous indiquez le mnémonique des données avec le type de données complexes (par exemple tableaux et structures).
- type de données `POINTER` :
  ceci n'est possible que lorsque l'opérande est un type de paramètre compatible avec des paramètres formels.
- type de données `NIL` :
  vous indiquez un pointeur zéro.

Le type de paramètre `POINTER` est autorisé pour les paramètres formels d'entrée, d'entrée/sorties de FB et de FC de même que pour les paramètres de sortie de FC.

Nota

Lorsque lors de l'appel d'un FB ou d'une FC, vous affectez une variable temporaire à un paramètre formel de type POINTER, vous ne pouvez pas transmettre ce paramètre à un autre bloc dans le bloc appelé. En effet, les adresses des variables temporaires perdent leur validité lors de la transmission.

Exemples

```
FUNCTION INTERVALLE: REAL
      VAR_INPUT
                  MonDB:BLOCK_DB;
                  TEMPS: TIMER;
      END_VAR
      VAR
                  INDICE: INTEGER;
      END_VAR
      BEGIN
      MonDB.DB5:=5;
      INTERVALLE:=..../Valeur en retour

END_FUNCTION
```

Exemple 9-3 Type de paramètre BLOCK_DB et TIMER

```
FUNCTION FC100: VOID
      VAR_IN_OUT
            in, out:ANY;
      END_VAR
      VAR_TEMP
            ret: INT;
      END_VAR
      BEGIN
      //...
      ret:=SFC20(DSTBLK:=out,SCRBLK:=in);
      //...
END_FUNCTION

FUNCTION_BLOCK FB100
      VAR
            ii:INT;
            aa, bb:ARRAY[1..1000] OF REAL;
      END_VAR
      BEGIN
      //...
      FC100(in:=aa, out:=bb);
      //...
END_FUNCTION_BLOCK
```

Exemple 9-4 Type de données ANY

Déclaration de variables locales et de paramètres de blocs

10

Présentation

Les variables locales et les paramètres de bloc sont des données que vous déclarez dans un bloc de code (FC, FB, OB) et qui ne sont valables que pour ce bloc. Le présent chapitre traite de la déclaration et de l'initialisation de ces données.

Structure du chapitre

10.1 Présentation

Diverses variables

Le tableau 10-1 décrit les catégories dans lesquelles entrent les variables locales :

Tableau 10-1 Variables locales

Variable	Signification
Variables statiques	Les variables statiques sont des variables locales, dont la valeur reste inchangée durant plusieurs exécutions de blocs (mémoire du bloc). Elles servent à stocker les valeurs d'un bloc fonctionnel et sont enregistrées dans son bloc de données d'instance.
Variables temporaires	Les variables temporaires appartiennent localement à un bloc de code et n'occupent **aucune** zone de mémoire statique, puisqu'elles sont enregistrées dans la pile de la CPU. Leur valeur ne reste conservée que durant une exécution du bloc. Il n'est **pas** possible d'adresser les variables temporaires en dehors du bloc dans lequel elles ont été déclarées.

Divers paramètres de blocs

Les paramètres de bloc servent à réserver de la place et ne sont définis que lors de l'utilisation concrète du bloc (appel). Les paramètres de bloc déclarés dans le bloc sont les paramètres formels, les valeurs qui leur sont affectées à l'appel du bloc sont les paramètres effectifs. Les paramètres formels d'un bloc peuvent être considérés comme variables locales.

Le tableau 10-2 décrit les catégories dans lesquelles entrent les paramètres de bloc :

Tableau 10-2 Paramètres de bloc

Paramètres de bloc	Signification
Paramètres d'entrée	A l'appel du bloc, les paramètres d'entrée prennent les valeurs d'entrée actuelles. Vous pouvez uniquement les lire.
Paramètres de sortie	Les paramètres de sortie transmettent les valeurs de sortie actuelles au bloc appelant. Vous pouvez les décrire et également les lire.
Paramètres d'entrée/sortie	A l'appel du bloc, les paramètres d'entrée/sortie prennent la valeur actuelle d'une variable, la traitent puis inscrivent les résultats dans la même variable.

Drapeaux (drapeau OK)

Le compilateur de SCL dispose d'un drapeau servant à déceler des erreurs durant l'exécution de programmes dans la CPU. Il s'agit d'une variable locale de type BOOL, dont le nom prédéfini est « OK ».

Déclaration de variables et de paramètres

Une section de déclaration différente et une paire de mots-clés sont affectées à chaque catégorie de variables locales ou de paramètres, comme le montre le tableau 10-3.

Chaque section contient les déclaration qui y sont autorisées. Elle ne doit figurer qu'une seule fois dans la section de déclaration d'un bloc, l'ordre des sections étant quelconque.

Les sections de déclaration autorisées dans chaque bloc sont marqués « x ».

Tableau 10-3 Sections de déclaration des variables locales et des paramètres

Données	Syntaxe	FB	FC	OB
Variable statique	VAR : END_VAR	X	X[1]	
Variable temporaire	VAR_TEMP : END_VAR	X	X	X
Paramètres de bloc comme : Paramètre d'entrée	VAR_INPUT : END_VAR	X	X	
Paramètre de sortie	VAR_OUTPUT : END_VAR	X	X	
Paramètre d'entrée/sortie	VAR_IN_OUT : END_VAR	X	X	

1 La déclaration de variables entre les mots-clés VAR et END_VAR est autorisée pour les fonctions, cependant lors de la compilation, les déclarations sont décalées dans la zone temporaire.

Initialisation

Lorsque vous déclarez des variables et des paramètres, vous devez leur affecter un type de données qui définit leur structure et par conséquent aussi la place mémoire requise. Vous pouvez en outre affecter des valeurs initiales à des variables statiques et aux paramètres d'un bloc fonctionnel. Le tableau 10-4 précise quand l'initialisation est possible ou ne l'est pas.

Tableau 10-4 Initialisation de données locales

Catégorie de données	Initialisation
Variables statiques	possible
Variables temporaires	impossible
Paramètres de bloc	uniquement possible pour les paramètres d'entrée et de sortie d'un bloc fonctionnel

10.2 Déclaration de variables et de paramètres

Présentation

La déclaration de variables ou de paramètres consiste à choisir un identificateur quelconque comme nom de la variable et à en indiquer le type de données. Le diagramme syntaxique suivant en montre la forme générale. L'affectation d'attributs système de paramètres est décrite en détails au paragraphe 8.4.

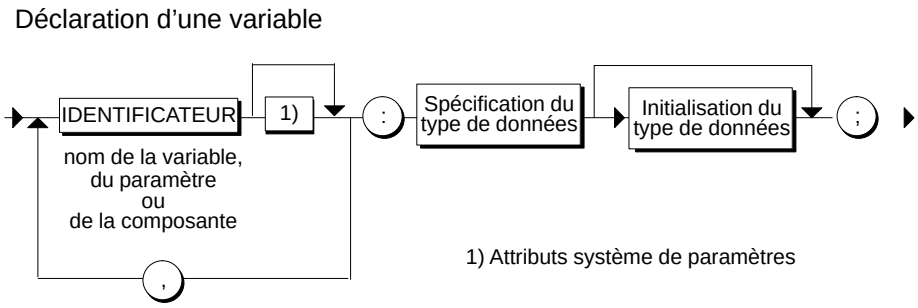


Figure 10-1 *Syntaxe : déclaration d'une variable*

Exemples de déclarations correctes :

```
VALEUR1   : REAL;
```

ou si plusieurs variables possèdent le même type :

```
VALEUR1, VALEUR2, VALEUR4,....: INT;

TABLEAU   : ARRAY[1..100, 1..10] OF REAL;

BLOC      : STRUCT
              MESURE:ARRAY[1..20] OF REAL;
              COMMUTATEUR:BOOL;
            END_STRUCT
```

Spécification du type de données

Tous les types de données décrits au chapitre 9 sont autorisés.

Nota

Vous pouvez déclarer les mots réservés, valables uniquement dans SCL, comme identificateurs en les faisant précéder par le signe # (par exemple, #FOR). Ceci est utile lorsque vous voulez transmettre des paramètres effectifs à des blocs programmés dans un langage différent (par exemple LIST).

10.3 Initialisation

Principe

Lorsque vous déclarez des variables statiques, des paramètres d'entrée et des paramètres de sortie d'un FB vous pouvez leur affecter une valeur par défaut. Pour ce faire, vous affectez une valeur (:=) après l'indication du type de données. Comme le montre la figure 10-2, vous pouvez affecter :

- une constante à une variable simple ou
- une liste d'initialisation à un tableau.

Initialisation

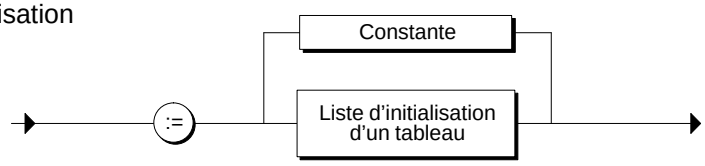


Figure 10-2 *Syntaxe : initialisation*

Exemple :

```
VALEUR      :REAL       := 20.25;
```

Sachez que l'initialisation d'une liste de variables (A1, A2, A3,...: INT:=...) n'est pas possible. Vous devez dans ce cas initialiser les variables séparément. Vous pouvez affecter des valeurs initiales à des tableaux, comme le montre la figure 10-3.

Liste d'initialisation de tableaux

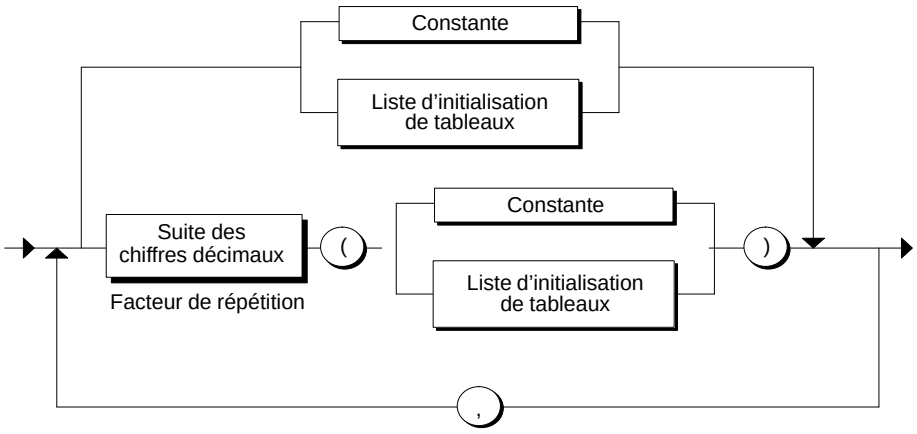


Figure 10-3 *Syntaxe : initialisation de tableaux*

```
TABLEAU : ARRAY[1..10, 1..100] OF INT:=10(100(0));
```

Facteur de répétition (nombre de colonnes)

Valeur

Facteur de répétition (nombre de lignes)

Exemples

L’exemple 10-1 illustre l’initialisation d’une variable statique :

```
VAR
      INDICE1: INT:= 3;
END_VAR
```

Exemple 10-1 Initialisation de variables statiques

L’exemple 10-2 montre l’initialisation d’un tableau bidimensionnel. Si dans SCL, vous souhaitez déclarer la structure de données suivantes sous le nom REGULATEUR vous devez écrire :

-54	736	-83	77
-1289	10362	385	2
60	-37	-7	103
60	60	60	60

```
VAR
REGULATEUR:
ARRAY [1..4, 1..4] OF INT:= -54, 736, -83, 77,
                            -1289, 10362, 385, 2,
                            60, -37, -7, 103,
                            4(60);

END_VAR
```

Exemple 10-2 Initialisation d’un tableau

L’exemple 10-3 représente l’initialisation d’une structure :

```
VAR
GENERATEUR:STRUCT
      DONNEES: REAL  := 100.5;
      A1:        INT := 10;
      A2:        STRING[6]:= ’FACTEUR’;
      A3:        ARRAY[1..12] OF REAL:= 12(100.0);
END_STRUCT;
END_VAR
```

Exemple 10-3 Initialisation d’une structure

10.4 Déclaration d'instances

Présentation

Dans la section de déclaration de blocs fonctionnels, vous pouvez non seulement déclarer des variables de types de données simples, complexes ou définis par l'utilisateur, mais également des variables de type FB ou SFB. Il s'agit des **instances locales** du FB ou du SFB.

Les instances locales sont enregistrées dans le bloc de données d'instance du bloc fonctionnel appelant.

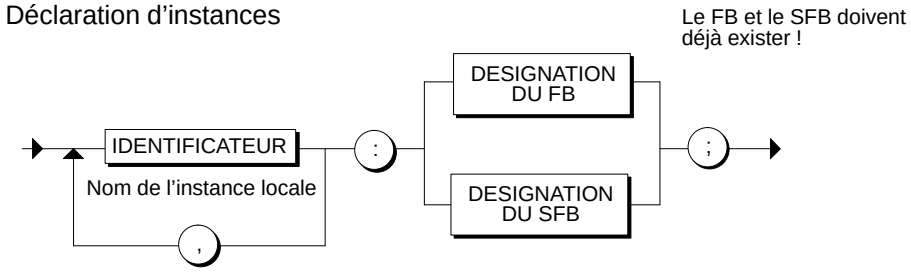


Figure 10-4 *Syntaxe : déclaration d'instances locales*

Exemples : voici des exemples dans lesquels la syntaxe de la figure 10-4 est correcte :

```
Alimentation1           : FB10;
Alimentation2,Alimentation3,Alimentation4  : FB100;
Moteur1     : Moteur ;
```

// Moteur est un mnémonique défini dans la table des mnémoniques.

Mnémonique	Adresse	Type de données
MOTEUR	FB20	FB20

Figure 10-5 *Table de mnémoniques correspondante dans STEP 7*

Initialisation

Une initialisation locale spécifique à l'instance n'est pas possible.

10.5 Variables statiques

Présentation

Une variable statique est une variable locale, dont la valeur reste inchangée durant toutes les exécutions du bloc (mémoire du bloc). Elle sert à stocker les valeurs d'un bloc fonctionnel et est enregistrée sous le bloc de données d'instance.

Section de variables statiques

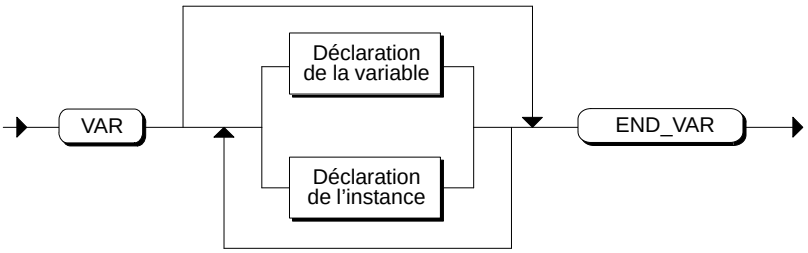


Figure 10-6 *Syntaxe : section de variables statiques*

Section de déclaration
VAR
END_VAR

Cette section de déclaration fait partie de la section de déclaration du FB. Vous pouvez y :

- déclarer les noms de variables et des types de données (l'initialisation est optionnelle) grâce à la déclaration de la variable, comme décrit au paragraphe 10.2 ou
- insérer d'autres déclarations existantes de variables grâce à la déclaration de l'instance, comme décrit au paragraphe 10.4.

Après la compilation, c'est cette section qui, avec les sections pour les paramètres de blocs déterminera la structure du bloc de données d'instance correspondant.

Exemple

L'exemple 10-4 représente la déclaration de variables statiques :

```
VAR
      PASSAGE        :INT;
      MESURE         :ARRAY[1..10] OF REAL;
      COMMUTATEUR :BOOL;
      MOTEUR_1,Moteur_2:FB100; // Déclaration de
                               // l'instance

END_VAR
```

Exemple 10-4 Déclaration de variables statiques

Adressage

Dans la section des instructions, l'adressage de ces variables s'effectue de la manière suivante :

- **par adressage interne**, c'est-à-dire dans la section des instructions du bloc fonctionnel dont la section de déclaration contient la déclaration de la variable. C'est ce qui est expliqué au chapitre 14, Affectation de valeurs.
- **par adressage externe, par l'intermédiaire du DB d'instance**, c'est-à-dire par l'intermédiaire de la variable indexée *DBx.variable*. DBx correspond à la désignation du bloc de données.

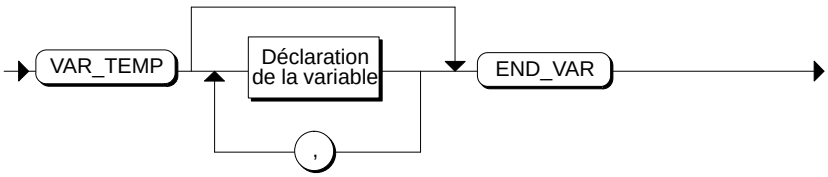
10.6 Variables temporaires

Présentation

Les variables temporaires appartiennent localement à un bloc de code et n'occupent **aucune** zone de mémoire statique. Leur valeur ne reste conservée que durant une exécution du bloc. Il n'est **pas** possible d'adresser les variables temporaires en dehors du bloc dans lequel elles ont été déclarées.

Vous ne devriez déclarer des données comme étant temporaires que si vous en avez uniquement besoin pour y enregistrer des résultats intermédiaires lors du traitement de votre OB, FB ou FC.

Section de variables temporaires



L'initialisarion n'est pas possible

Figure 10-7 *Syntaxe : section de variables temporaires*

Section de déclaration
VAR_TEMP
END_VAR

Cette section de déclaration fait partie d'un FB, d'une FC, ou d'un OB. Comme mentionné au paragraphe 10.2, le nom de la variable et le type de données doivent être spécifiés dans la déclaration de la variable.

Au début de l'exécution d'un OB, FB ou d'une FC, la valeur des données temporaires n'est pas définie. L'initialisation n'est pas possible.

Exemple

L'exemple 10-5 montre la déclaration de variables temporaires dans un bloc :

```
VAR_TEMP
      MEMOIRE_1   :ARRAY [1..10] OF INT;
      AUXIL1,AUXIL2:REAL;
END_VAR
```

Exemple 10-5 Déclaration de variables temporaires dans un bloc

Adressage

L'adressage de ces variables s'effectue toujours dans la section des instructions du bloc de code dont la section de déclaration contient la déclaration de la **variable** (**adressage interne**), voir chapitre 14, Affectation de valeurs.

10.7 Paramètres de bloc

Présentation

Les paramètres de bloc sont des paramètres formels d'un bloc fonctionnel ou d'une fonction. A l'appel du bloc fonctionnel ou de la fonction, les paramètres effectifs remplacent les paramètres formels, ce qui constitue un moyen d'échange d'informations entre les blocs appelés et les blocs appelants.

- Les paramètres d'entrée formels prennent les valeurs d'entrée en cours (flux de données de l'extérieur vers l'intérieur).
- Les paramètres de sortie formels servent à transmettre des valeurs de sortie (flux de données de l'intérieur vers l'extérieur).
- Les paramètres d'entrée/sortie ont à la fois la fonction d'un paramètre d'entrée et celle d'un paramètre de sortie.

De plus ample informations sur l'utilisation de paramètres et l'échange d'informations sont fournies au chapitre 16.

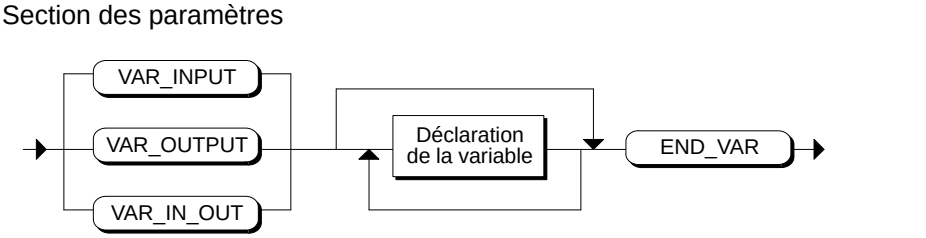


Figure 10-8 *Syntaxe : section des paramètres*

Section de déclaration
`VAR_INPUT`
`VAR_OUTPUT`
`VAR_IN_OUT`

Cette section de déclaration fait partie d'un FB, ou d'une FC. Comme mentionné au paragraphe 10.2, le nom de la variable et le type de données doivent être spécifiés dans la déclaration de la variable.

Après la compilation d'un FB, ce sont ces sections qui, avec la section `VAR` et `END_VAR` détermineront la structure du bloc de données d'instance correspondant.

Exemple

L’exemple 10-6 représente la déclaration d’un paramètre :

```
VAR_INPUT           //Paramètre d’entrée
      REGULATEUR  :DWORD;
      HEURE       :TIME_OF_DAY;
END_VAR
```

```
VAR_OUTPUT          //Paramètre de sortie
      VALEUR_CONSIGNE : ARRAY [1..10] of INT;
END_VAR
```

```
VAR_IN_OUT          //Paramètre d’entrée/sortie
      SELECTION: INT;
END_VAR
```

Exemple 10-6 Déclaration de paramètres

Adressage

Dans la section des instructions d’un bloc de code, l’adressage des paramètres de bloc s’effectue :

- **par adressage interne**, c’est-à-dire dans la section des instructions du bloc dont la section de déclaration contient la déclaration des **paramètres**. C’est ce qui est expliqué au chapitre 14, Affectation de valeurs et au chapitre 13, Expressions, opérateurs et opérandes.
- **par adressage externe par l’intermédiaire du DB d’instance**
  Vous pouvez accéder aux paramètres de bloc d’un bloc fonctionnel par son DB d’instance (voir paragraphe 14.8).

10.8 Drapeaux (drapeau OK)

Description

Le drapeau OK sert à marquer l'exécution correcte ou incorrecte d'un bloc. Il s'agit d'une variable globale de type BOOL avec le mot-clé « OK ».

Si une erreur se produit durant l'exécution d'une instruction de bloc (par exemple un débordement haut pour une multiplication), alors le drapeau OK prend la valeur FALSE. Lorsque vous quittez le bloc, la valeur du drapeau OK est enregistrée dans le paramètre de sortie ENO défini automatiquement (paragraphe 16.4) d'où il peut être exploité par le bloc appelant.

Au début de l'exécution d'un bloc, le drapeau OK a la valeur TRUE. Il peut être interrogé à un endroit quelconque du bloc par des instructions de SCL ou prendre la valeur TRUE/FALSE.

Déclaration

Le drapeau OK est une variable déclarée dans le système. Il s'avère donc inutile de la déclarer. Si vous souhaitez l'utiliser dans votre programme utilisateur, vous devez cependant choisir l'option de compilation « Drapeau OK » avant d'effectuer la compilation.

Exemple

L'exemple 10-7 illustre l'utilisation du drapeau OK :

```
      // Attribuer la valeur TRUE à la variable OK
      // pour permettre de vérifier
      // si l'action suivante
      // se déroule correctement.
OK: = TRUE;
SUM: = SUM + IN;
IF OK THEN
      // L'addition s'est correctement déroulée.
      :
      :
ELSE  // L'addition n'a pas été réalisée
      // correctement
      :
END_IF;
```

Exemple 10-7 Utilisation du drapeau OK

Déclaration de constantes et de repères de saut

11

Présentation

Les constantes sont des données possédant une valeur fixe qui ne varie pas durant l'exécution du programme. Une **constante littérale** est une constante dont la valeur s'exprime par la notation.

Il n'est pas nécessaire de déclarer les constantes. Vous pouvez toutefois leur attribuer des mnémoniques dans la section de déclaration.

Les repères de saut correspondent à des branchements dans la section des instructions du bloc de code.

La déclaration des mnémoniques de constantes ainsi que des repères de saut s'effectue dans des sections de déclaration distinctes.

Structure du chapitre

11.1 Constantes

Utilisation de constantes

Outre les variables et les paramètres de bloc, vous pouvez également utiliser des constantes dans les affectations de valeurs et les expressions. Il peut s'agir de constantes sous forme de constantes littérales ou de constantes désignées par un mnémonique.

Déclaration de mnémoniques

La déclaration des mnémoniques des constantes s'effectue dans la section de déclaration `CONST` de votre bloc de code (voir paragraphe 8.4)

Section des constantes

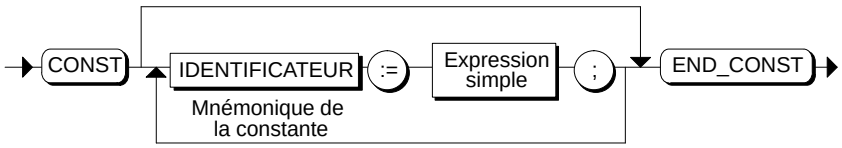


Figure 11-1 *Syntaxe : section des constantes*

L'expression simple désigne une expression arithmétique, dans laquelle vous pouvez utiliser les opérations de base +, -, *, /, DIV et MOD.

Exemple

L'exemple 11-1 représente la déclaration des mnémoniques :

```
CONST
 Nombre           := 10 ;
 HEURE1           := TIME#1D_1H_10M_22S.2MS ;
 NOM              := 'SIEMENS' ;
 NOMBRE2          := 2 * 5 + 10 * 4 ;
 NOMBRE3          := 3 + NOMBRE2 ;
 END_CONST
```

Exemple 11-1 Déclaration de constantes symboliques

Notations

SCL met à votre disposition diverses notations (formats) pour saisir ou afficher des constantes, que l'on désignera alors par constantes littérales. C'est d'elles que nous allons parler dans la suite de ce chapitre.

11.2 Constantes littérales

Définition

Une constante littérale est caractérisée par une notation formelle qui définit sa valeur et son type. Il existe les classes suivantes de constantes littérales :

- constantes littérales numériques,
- constantes littérales de type caractère,
- constantes de temporisation.

La notation de la valeur de la constante diffère selon le type et le format de données choisis :

```
15      Valeur 15  sous forme de nombre entier en représentation décimale
2#1111  Valeur 15  sous forme de nombre entier en représentation binaire
16#F    Valeur 15 sous forme de nombre entier en représentation hexadécimale
```

Plusieurs notations pour la constante littérale de valeur 15

Affectation de types de données à des constantes littérales

Le type de données affecté à une constante est celui dont la plage de valeurs suffit juste encore pour y inscrire la constante sans perdre de valeur. Ce type de données doit être compatible avec celui qui a été choisi pour utiliser des constantes dans une affectation à une variable ou comme type de données cible dans une expression :

Si, par exemple, vous définissez une constante littérale entière dont la valeur est supérieure à la plage des entiers autorisée, elle sera considérée comme un entier double. Si vous affectez cette valeur à une variable de type entier, le compilateur émet un message d’erreur.

11.3 Notations des constantes littérales entières et réelles

Présentation

Pour les notations de valeurs numériques, SCL met à votre disposition :

- des constantes littérales entières pour les nombres entiers et
- des constantes littérales réelles pour les nombres à virgule flottante.

Chacun de ces deux types de constantes littérales correspond à une suite de chiffres, dont la structure est représentée à la figure 11-2. Dans les diagrammes syntaxiques suivants, on parlera de suite de chiffres décimaux.

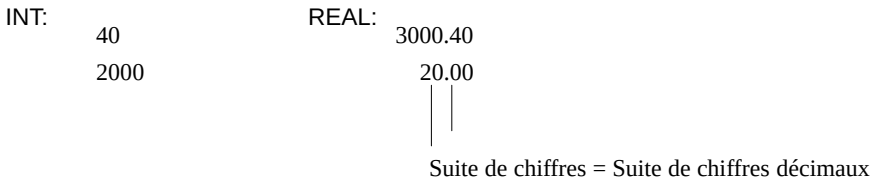


Figure 11-2 *Suite de chiffres dans une constante littérale*

Dans une constante littérale, une suite de chiffres décimaux peut optionnellement être séparée par des caractères de soulignement pour améliorer la lisibilité de grands nombres.

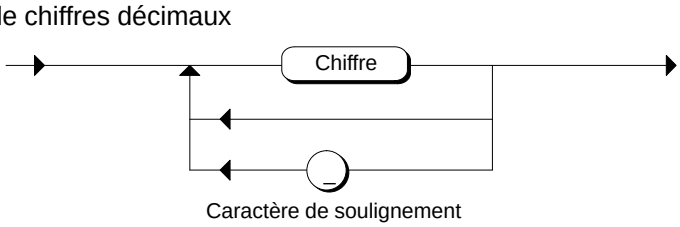


Figure 11-3 *Syntaxe : suite de chiffres décimaux*

Voici des notations correctes de suites de chiffres décimaux dans les constantes littérales :

```
1000
1_120_200
666_999_400_311
```

Constante littérale entière

Les constantes littérales entières sont des valeurs entières que vous affectez en fonction de leur longueur à des variables du type de données :

`BOOL, BYTE, INT, DINT, WORD` ou `DWORD`

La figure 11-4 représente la syntaxe d'une constante littérale entière :

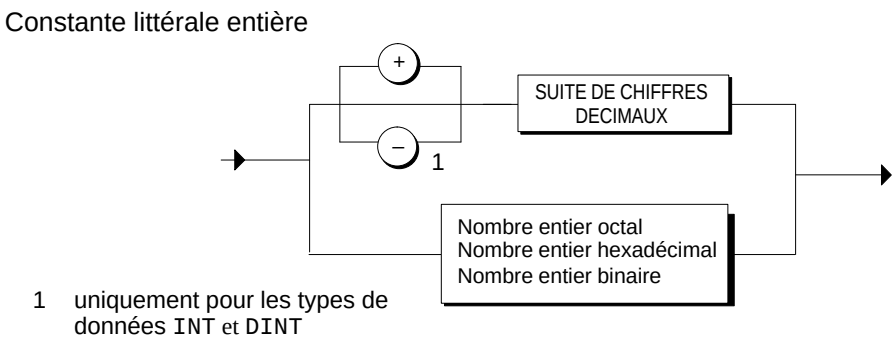


Figure 11-4 *Syntaxe : constante littérale entière*

Voici des notations correctes de suites de chiffres décimaux dans des constantes littérales entières :

```
1000
+1_120_200
-666_999_400_311
```

Valeurs binaires, octales et hexadécimales

Vous avez la possibilité d'entrer une constante littérale entière dans un autre système de numération que le système décimal en entrant d'abord le préfixe **2#, 8# ou 16#**, suivi du nombre dans la représentation du système respectif. L'usage du caractère de soulignement à des fins de meilleure lisibilité de grands nombres est optionnel.

La figure 11-5 illustre la notation générale d'une constante littérale entière, en prenant pour exemple une suite de chiffres formant un nombre octal :

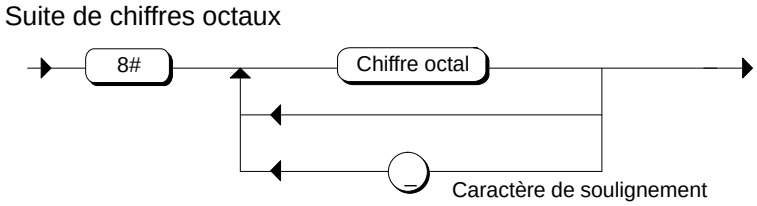


Figure 11-5 *Syntaxe : suite de chiffres octaux*

Les exemples suivants représentent des notations correctes de constantes littérales entières

```
Valeur_2:=2#0101;// nombre binaire, valeur décimale 5
Valeur_3:=8#17;  // nombre octal, valeur décimale 15
Valeur_4:=16#F;  // nombre hexadécimal, valeur décimale 15
```

Constante littérale réelle

Les constantes littérales réelles sont des valeurs avec virgule que vous pouvez affecter aux variables de type REAL. L'indication du signe est optionnelle. En l'absence de signe, le nombre est interprété comme un nombre positif.
La figure 11-6 donne la syntaxe d'un nombre réel :

Constante littérale réelle

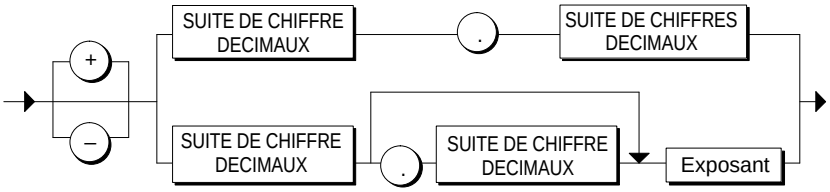


Figure 11-6 *Syntaxe : constante littérale entière*

La notation exponentielle vous permet de représenter un nombre à virgule flottante par un exposant, que vous écrivez en entrant la lettre « E » ou « e » suivie d'un nombre décimal. La syntaxe en est donnée à la figure 11-7.

Exposant

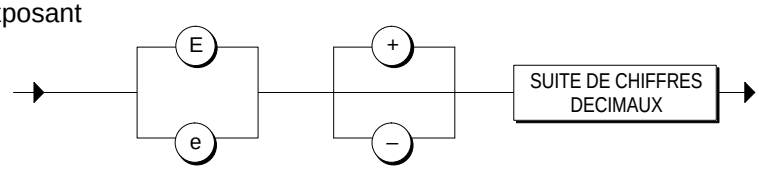


Figure 11-7 *Syntaxe : exposant*

Exemple :

Dans SCL vous pouvez représenter la grandeur 3 x 10 ^{10} par les nombres réels suivants :

```
3.0E+10     3.0E10      3e+10       3E10
0.3E+11     0.3e11      30.0E+9     30e9
```

Exemples

La figure 11-2 résume les diverses possibilités :

```
// Constantes littérales entières
NOMBRE1:= 10 ;
NOMBRE2:= 2#1010 ;
NOMBRE3:= 16#1A2B ;
// Constantes littérales réelles
NOMBRE4:= -3.4 ;
NOMBRE5:= 4e2 ;
NOMBRE6:= 40_123E10;
```

Exemple 11-2 Constantes littérales numériques

11.4 Notations des constantes littérales de type caractère ou chaîne de caractères

Présentation

SCL offre aussi la possibilité de saisir et de traiter des informations textuelles, par exemple une chaîne de caractères qui doit être affichée comme message.

Les constantes littérales de type caractère ne permettent pas d'effectuer de calculs et ne peuvent de ce fait **pas** être utilisées dans des expressions. Il existe :

- les constantes littérales de type caractère unique,
- les constantes littérales de type chaîne de 254 caractères au maximum.

Constante littérale de type caractère unique

Comme le montre la figure 11-8, une constante littérale de type caractère comporte exactement un caractère, représenté entre deux apostrophes (').

Constante littérale de type caractère

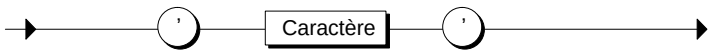


Figure 11-8 *Syntaxe : constante littérale de type caractère*

Exemple :

```
Caractère_1:='B';       // Caractère B
```

Constante littérale de type chaîne de caractères

Une constante littérale de type chaîne de caractères est une chaîne de 254 caractères au maximum (lettres, chiffres et caractères spéciaux), représentée entre deux apostrophes ('). Aussi bien les majuscules que les minuscules y sont autoriées.

Constante littérale de type chaîne de caractères

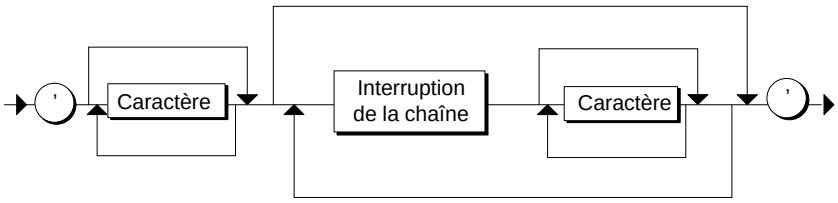


Figure 11-9 *Syntaxe : constante littérale de type chaîne de caractères*

Voici des constantes littérales de type chaîne de caractères autorisées :

```
'ROUGE'     '75000 Paris'          '270-32-3456'
'FR19.95'   'La réponse correcte est :'
```

Tenez compte du fait que lorsque vous affectez une constante littérale de type chaîne de caractères à une variable de type STRING, le nombre de caractères peut être limité à moins de 254.

L'affectation :

```
TEXT:STRING[20]:='SIEMENS KARLSRUHE Rheinbrückenstr.'
```

génère un message d'erreur et inscrit `'SIEMENS KARLSRUHE Rh'`

dans la variable 'TEXT'.

Pour entrer des caractères de mise en forme, l'apostrophe (') ou le caractère dollar ' $ ', faites les précéder du caractère d'alignement $. Vous avez la possibilité d'**interrompre** à plusieurs reprises une constante littérale de type chaîne de caractères puis de la reprendre.

Interruption d'une chaîne de caractères

Une chaîne de caractères peut se trouver sur une même ligne d'un bloc SCL ou être répartie sur plusieurs lignes, grâce à des codes spéciaux. Le code `'$>'` permet d'interrompre une chaîne, le code `'$<'` de la poursuivre dans la ligne suivante.

```
TEXT:STRING[20]:='Le FB$>//Version provisoire
$<convertit';
```

L'écart entre le code d'interruption et le code de poursuite peut s'étendre sur plusieurs lignes et ne peut contenir, hormis des caractères d'espacement, que des commentaires. Comme le montre la figure 11-10, vous pouvez interrompre et reprendre plusieurs fois une constante littérale de type chaîne de caractères.

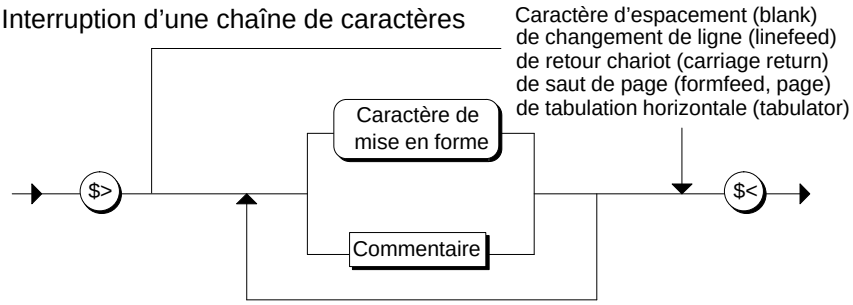


Figure 11-10 *Syntaxe : interruption d'une chaîne de caractères*

Caractères imprimables

Les caractères que vous pouvez utiliser dans les constantes littérales de type caractère ou chaîne de caractères sont ceux du jeu de caratères ASCII étendu complet. Pour entrer des caractères de mise en forme spéciaux et des caractères qui ne peuvent être représentés directement (' et $), utilisez le caractère d'alignement $.

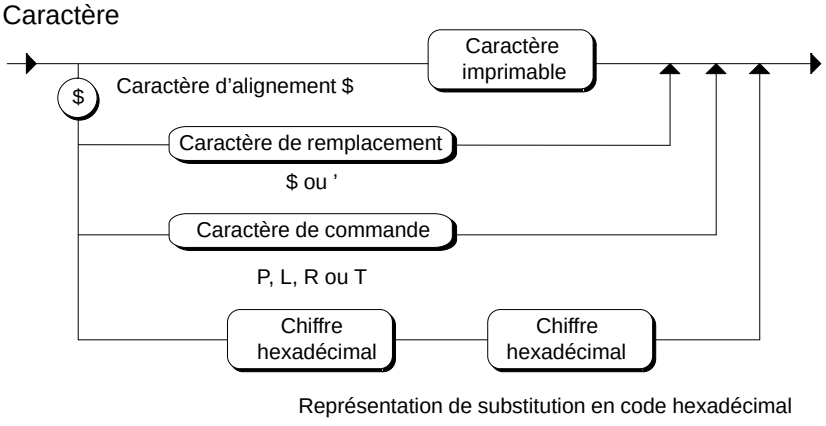


Figure 11-11 *Syntaxe : caractère*

Caractères non imprimables

Vous avez également la possibilité d'indiquer des caractères non imprimables du jeu de caractères ASCII étendu complet dans une constante littérale de type caractère ou chaîne de caractères. Il vous suffit pour cela d'indiquer leur représentation en **code hexadécimal**.

Pour indiquer un caractère ASCII, vous entrez **$hh**, hh correspondant à sa valeur hexadécimale.

Exemple :

```
CARACTERE_A :='$41'; //correspond à la lettre 'A'
Blank       :='$20'; //correspond au caractère ␣
```

De plus amples informations concernant les caractères de remplacement et les caractères de commande sont fournies en annexe A.

Exemples

Les exemples suivants illustrent la formulation des constantes littérales de type caractère :

```
// Constante littérale de type caractère
Caractère:= 'S' ;

// Constante littérale de type chaîne de caractères
NOM:= 'SIEMENS' ;

// Interruption d'une constante de type chaîne de
// caractère
MESSAGE1:= 'Commande $>
$<MOTEUR' ;

 // Chaîne sous forme de représentation hexadécimale
 MESSAGE1:= '$41$4E' (*chaîne de caractères AN*);
```

Exemple 11-3 Constantes littérales de type caractère

11.5 Notations des constantes de temporisation

Notation des types de temporisations

SCL met à votre disposition divers formats pour entrer des valeurs de date et d'heure. Voici les données de temporisation possibles :

```
Date
Durée
Heure du jour
Date et heure
```

Date

Comme le montre la figure 11-12, une date est introduite par le préfixe DATE# ou D#.

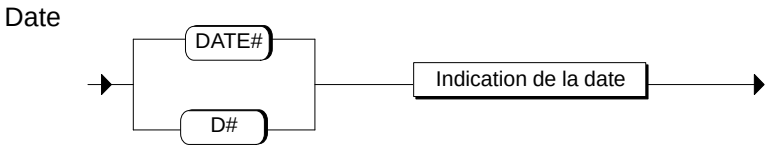


Figure 11-12 *Syntaxe : date*

Pour **indiquer la date**, vous entrez des nombres entiers pour l'année (4 places), le mois et le jour, séparés par des tirets.

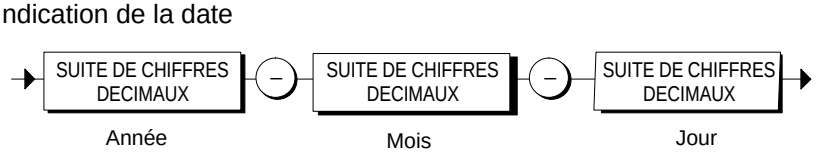


Figure 11-13 *Syntaxe : indication de la date*

Voici des indications de date correctes :

```
// Indication de la date
VARIABLE_TEMPS1:= DATE#1995-11-11;
VARIABLE_TEMPS2:= D#1995-05-05;
```

Durée

Comme le montre la figure 11-14, une durée est introduite par le préfixe TIME# ou T#. L'indication de la durée peut être réalisée de deux manières :

- par la représentation décimale,
- par la représentation par nivea.ux.

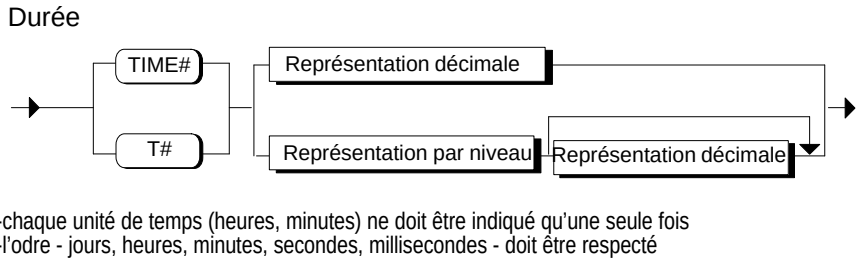


Figure 11-14 *Syntaxe : durée*

La **représentation décimale** vous permet d'exprimer la durée alternativement en jours, heures, minutes, secondes ou millisecondes :

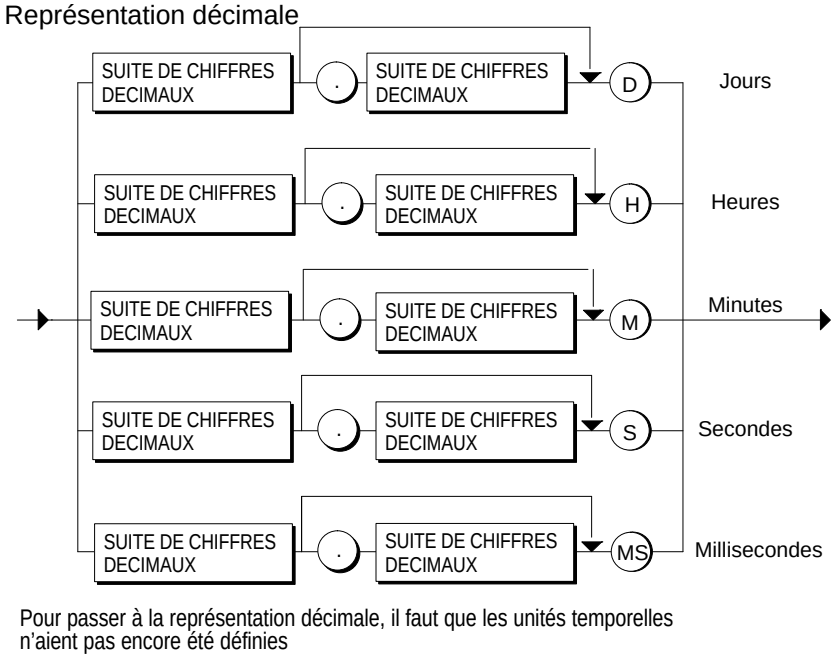


Figure 11-15 *Syntaxe : représentation décimale*

Exemples

Voici des indications correctes :

```
TIME#20.5D          correspond à 20,5 jours
TIME#45.12M         correspond à 45,12 minutes
T#300MS             correspond à 300 millisecondes
```

La **représentation par niveau**x vous permet d'exprimer la durée comme une séquence de jours, heures, minutes, secondes ou millisecondes. L'omission de composantes individuelles y est permise, comme le montre la figure 11-16. Vous devez cependant indiquer au minimum une composante.

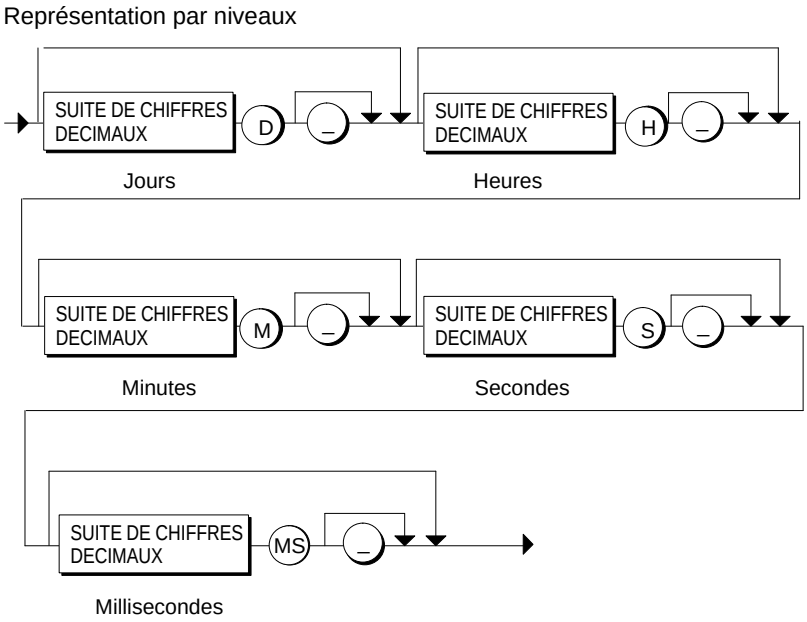


Figure 11-16 *Syntaxe : représentation par niveaux*

Voici des indications correctes :

```
TIME#20D_12H
TIME#20D_10H_25M_10s
TIME#200S_20MS
```

Heure du jour

Comme le montre la figure 11-17, l'heure du jour est introduite par le préfixe TIME_OF_DAY# ou TOD#.

Heure du jour

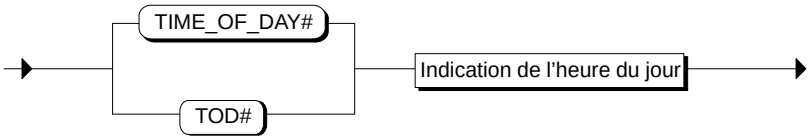


Figure 11-17 *Syntaxe : heure du jour*

Pour **indiquer l'heure du jour**, vous entrez les heures, minutes et secondes séparées par deux-points. L'indication des millisecondes est optionnelle. Elle est séparée des autres par un point. La figure 11-18 représente la syntaxe de l'indication de l'heure du jour :

Indication de l'heure du jour

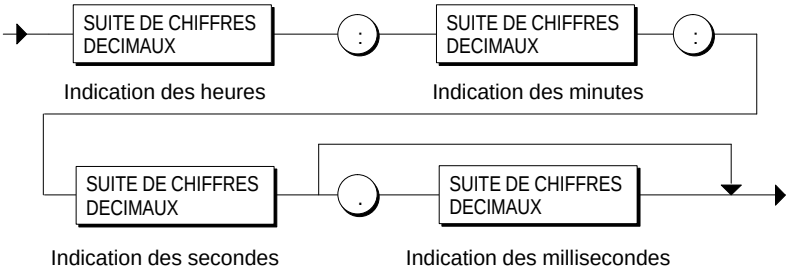


Figure 11-18 *Syntaxe : indication de l'heure du jour*

Voici des indications correctes :

```
//Indication de l'heure du jour
HEURE1:= TIME_OF_DAY#12:12:12.2;
HEURE2:= TOD#11:11:11.7.200;
```

Date et heure

Comme le montre la figure 11-19, l'indication de la date et de l'heure est introduite par le préfixe `DATE_AND_TIME#` ou `DT#`. Il s'agit d'une constante littérale formée de l'indication de la date et de celle de l'heure du jour.

Date et heure

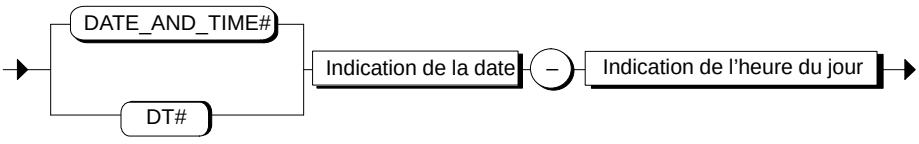


Figure 11-19 *Syntaxe : date et heure*

Cet exemple illustre l'indication de la date et de l'heure :

```
//Indication de la date et de l'heure
HEURE1:= DATE_AND_TIME#1995-01-01-12:12:12.2;
HEURE2:= DT#1995-02-02-11:11:11;
```

11.6 Repères de saut

Description

Les repères de saut (étiquettes) servent à définir le branchement d'une instruction GOTO (voir paragraphe 11.4).

Déclaration de repères

Les repères de saut doivent être déclarés avec leur mnémonique dans la section de déclaration d'un bloc de code (voir paragraphe 8.4) :

Section de déclaration des repères de saut

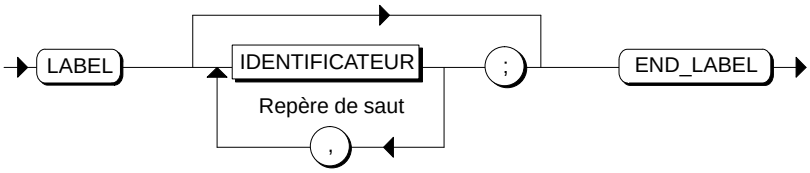


Figure 11-20 *Syntaxe : section de déclaration des repères de saut*

Exemple

L'exemple suivant illustre la déclaration des repères de saut :

```
LABEL
      REPERE1, REPERE2, REPERE3;
END_LABEL;
```

Exemple 11-4 Repères de saut

Déclaration de données globales

12

Présentation

Les données globales sont des données que tout bloc de code (FC, FB, OB) peut utiliser en les adressant de manière absolue ou symbolique. Le présent chapitre présente les diverses zones de mémoire et décrit comment adresser ces données.

Structure du chapitre

12.1 Présentation

Données globales

Dans SCL, vous avez la possibilité d'adresser des données globales. Il en existe deux sortes :

- **Les zones de mémoire de la CPU :**

  Il s'agit de données déclarées dans le système, comme par exemple les entrée, les sorties et les mémentos (voir paragraphe 7.5). Leur nombre dépend de votre CPU.

- **Les données utilisateur globales sous forme de blocs de données que vous pouvez charger :**

  Ces zones de données se trouvent dans des blocs de données. Pour pouvoir les utiliser, vous devez préalablement créer les blocs et y déclarer les données. Dans le cas de blocs de données d'instance, elles résultent de blocs fonctionnels et sont créées automatiquement.

Mode d'adressage

L' adressage des données globales peut être :

- **absolu :** avec un identificateur d'opérande et une adresse absolue,
- **symbolique :** avec un mnémonique que vous avez préalablement défini dans la table des mnémoniques (voir **/231/**),
- **indexé :** avec un identifiacteur d'opérande et un indice de tableau,
- **structuré :** avec une variable.

Tableau 12-1 Utilisation des modes d'adressage des données globales

Mode d'adressage	Zones de mémoire de la CPU	Données utilisateur globales
absolu	oui	oui
symbolique	oui	oui
indexé	oui	oui
structuré	non	oui

12.2 Zones de mémoire d'une CPU

Définition

Les zones de mémoire d'une CPU sont des zones déclarées dans le système, que vous n'avez par conséquent pas besoin de déclarer dans votre bloc de code.

Diverses zones de mémoire

Chaque CPU met à votre disposition les zones de mémoire suivantes possédant leur propre zone d'adressage :

- entrées/sorties de la mémoire image,
- entrées/sorties de la périphérie,
- mémentos,
- temporisations, compteurs (voir chapitre 17).

Syntaxe de l'adressage

Vous adressez les zones de mémoire de la CPU dans une affectation de valeur réalisée dans la section des instructions d'un bloc de code (voir paragraphe 14.3) par :

- adressage simple, absolu, symbolique, ou
- adressage indexé.

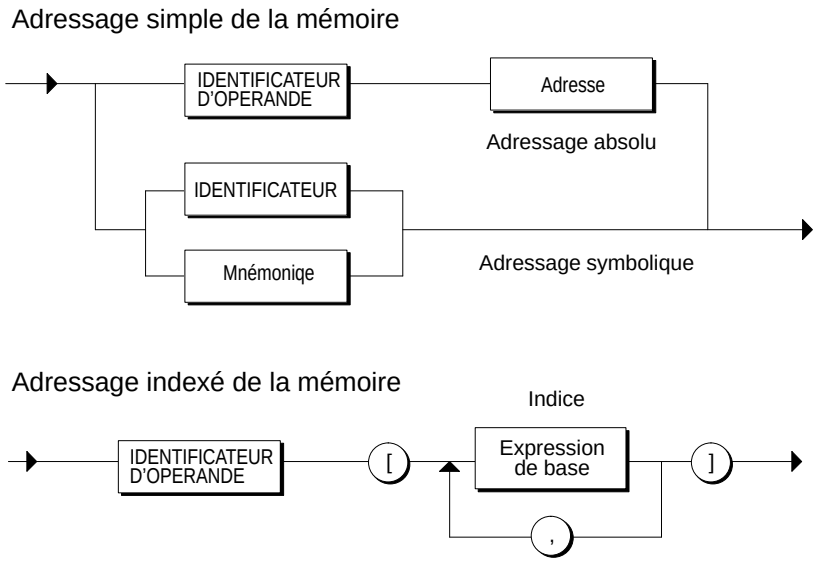


Figure 12-1 *Syntaxe : adressage simple et indexé de la mémoire*

12.3 Adressage absolu des zones de mémoire de la CPU

Principe

L'adressage absolu d'une zone de mémoire s'effectue par exemple en affectant une valeur à une variable de même type :

```
ETAT_2:= EB10;
```

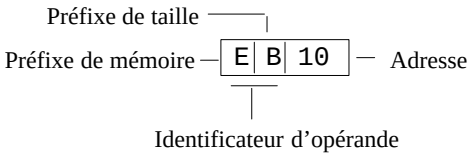
Adressage absolu

Variable de même type

L'identificateur absolu désigne une zone de mémoire dans la CPU, que vous spécifiez en indiquant l'identificateur d'opérande (EB dans notre exemple) suivi de l'adresse (ici `10`).

Identificateur absolu

L'identificateur absolu est formé d'un identificateur d'opérande, lui-même composé d'un préfixe de mémoire et d'un préfixe de taille, ainsi que d'une adresse.



Identificateur d'opérande

L'association d'un préfixe de mémoire et d'un préfixe de taille constitue un identificateur d'opérande.

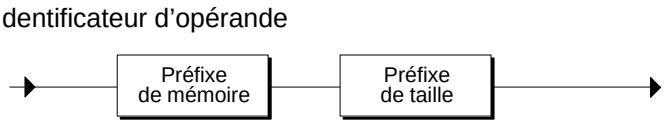


Figure 12-2 *Syntaxe : identificateur d'opérande*

Préfixe de mémoire

Le préfixe de mémoire permet de définir le type de zone de mémoire. Vous disposez des types représentés à la figure 12-3[1]:

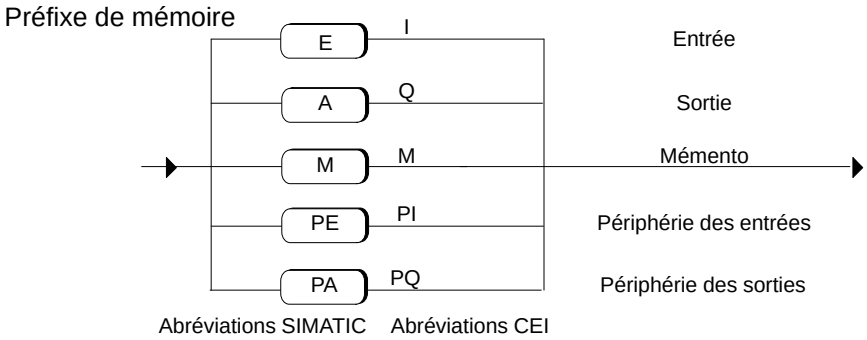


Figure 12-3 *Syntaxe : préfixe de mémoire*

1 *Selon le choix opéré dans le SIMATIC Manger (sélection du compilateur), ce sont les abréviations SIMATIC (allemandes) ou les abréviations CEI (internationales) dont la signification est réservée. Le choix des abréviations ne dépend pas de celui de la langue.*

Préfixe de taille

L'indication du préfixe de taille vous permet de spécifier la taille ou le type de la zone de mémoire qui doit être lue dans la périphérie, par exemple un octet ou un mot (voir figure 12-4). Si vous spécifiez un bit, cette indication est optionnelle. La syntaxe est indiquée par la figure 12-4 :

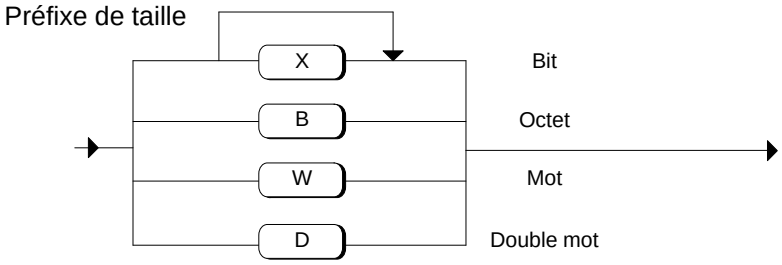


Figure 12-4 *Syntaxe : préfixe de taille*

Adresse

Pour l'adresse, vous indiquez une adresse absolue qui désigne un bit, un octet, un mot ou un double mot, selon le préfixe de taille que vous avez utilisé. Si vous avez spécifié un « bit », vous pouvez indiquer une adresse binaire supplémentaire, voir figure 12-5. Le premier numéro correspond à l'adresse d'octet et le second à l'adresse binaire.

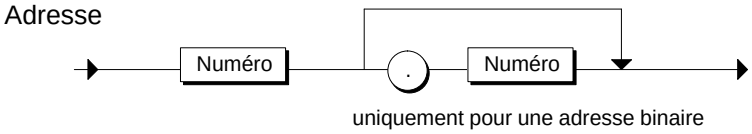


Figure 12-5 *Syntaxe : adresse*

Exemples

Voici des exemples d'adressage absolu :

```
OCTET_ETAT          := EB10;
ETAT_3              := E1.1;
Valeur_mesure     := EW20;
```

Exemple 12-1 Adressage absolu

12.4 Adressage symbolique des zones de mémoire de la CPU

Principe

La programmation symbolique consiste à adresser une zone de mémoire de la CPU en utilisant un mnémonique, à la place d'un identificateur d'opérande et d'une adresse :

Mnémonique	Opérande	Type de donées	Commentaire
Contact _moteur	E 1.7	BOOL	Commutateur de contact du moteur A 1
Entrée	EW 10	INT	Mot d'état d'entrée
Octet d'entrée1	EB1	BYTE	Octet d'entrée
"Entrée 1.1"	E 1.1	BOOL	Barrière photoélectrique
Voies de mesure	MW2	WORD	Mémoire des valeurs de mesure

Pour affecter les mnémoniques aux opérandes respectifs de votre programme utilisateur, vous devez créer une table de mnémoniques.

Tous les types de données simples sont autorisés, si tant est que leur longueur est suffisante pour la longueur d'adressage.

Adressage

L'adressage consiste par exemple à affecter une valeur à une variable de même type avec le mnémonique déclaré.

```
VALEUR_MESURE_1:= Contact_moteur;
```

Création d'une table de mnémoniques

La création de la table de mnémoniques et la saisie des valeurs doit être réalisée avec STEP 7.

Vous pouvez ouvrir la table de mnémoniques avec SIMATIC Manager, ou directement avec SCL en choisissant la commande **Outils ▸ Table de mnémoniques**.

Vous avez en outre la possibilité d'y importer d'autres tables de mnémoniques créées avec un éditeur de texte quelconque et enregistrées sous forme de fichiers de texte afin de les y traiter (pour plus d'informations reportez-vous au guide de l'utilisateur **/231/**.

Exemples

Voici des exemples d'adressage symbolique :

```
OCTET_ETAT        := Octet_entrée_bas;
ETAT_3            := "Entrée 1.1";
Valeur_mesure     := Canaux_mesure;
```

Exemple 12-2 Adressage symbolique

12.5 Adressage indexé des zones de mémoire de la CPU

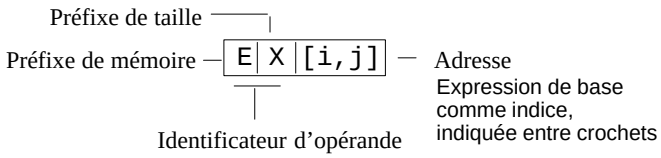
Principe

Vous avez également la possibilité d'effectuer un adressage indexé des zones de mémoire de la CPU. L'avantage par rapport à l'adressage absolu réside dans le fait que vous pouvez adresser les zones de mémoire de manière dynamique, en utilisant des indices variables. Ainsi, vous pouvez par exemple utiliser la variable de contrôle d'une boucle FOR comme indice.

L'adressage indexé d'une zone de mémoire est similaire à l'adressage absolu et se distingue que dans l'indication de l'adresse. A la place de l'adresse, vous devez spécifier un indice. Il peut s'agir d'une constante, d'une variable ou d'une expression arithmétique.

Identificateur absolu

Dans l'adressage indexé, l'identificateur absolu est formé de l'identificateur d'opérande ainsi que d'une expression de base comme indice (voir paragraphe 12.3).



Règles à appliquer à l'adressage indexé

L'indexation doit répondre aux règles suivantes :

- Dans le cas d'un adressage du type de données `BYTE`, `WORD` ou `DWORD`, vous devez utiliser exactement un indice. Celui-ci sera interprété comme adresse d'octet. La longueur d'adressage sera déterminée par le préfixe de taille.
- Dans le cas d'un adressage du type de données `BOOL`, vous devez utiliser deux indices. Le premier spécifie l'adresse d'octet, le second la position du bit dans cet octet.
- Chaque indice doit être une expression arithmétique du type de données `INT`.

```
VALEUR_MESURE_1   := EW[COMPTEUR];

MARQUE_SORTIE     := E[NO_OCT, NO_BIT];
```

Exemple 12-3 Adressage indexé

12.6 Blocs de données

Présentation

Dans les blocs de données, vous pouvez enregistrer et traiter toutes les données dont le domaine de validité s'étend sur l'ensemble du programme ou du projet de votre application. Tout bloc de code y a accès en lecture et en écriture.

Déclaration

La syntaxe de la structure des blocs de données est décrite au chapitre 8.
Il existe trois types de blocs de données :

- les blocs de données,
- les blocs de données d'instance.

Adressage des blocs de données

L'adressage des données de tout bloc de données peut être :

- simple ou absolu,
- indexé,
- structuré.

La figure 12-6 illustre ces divers modes d'adressage :

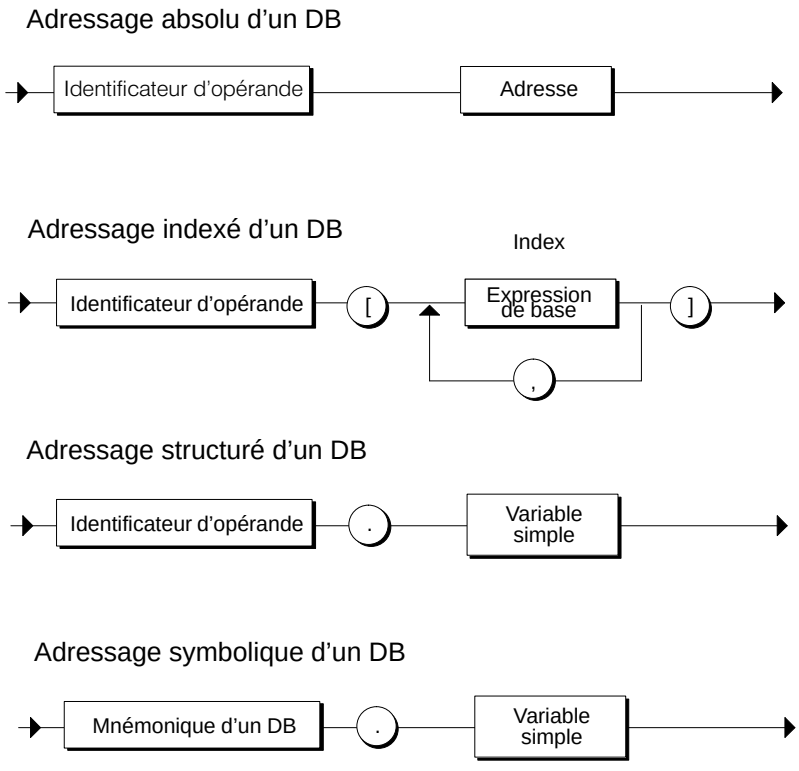
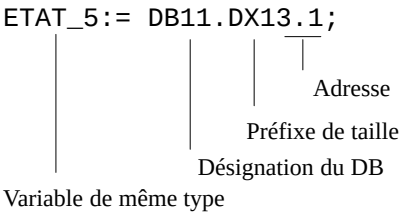


Figure 12-6 *Syntaxe : adressage absolu, indexé et structuré des blocs de données*

12.7 Adressage absolu des blocs de données

Principe

De même que pour les zones de mémoire de la CPU, l'adressage absolu à un bloc de données est réalisé par affectation d'une valeur à une variable de même type. Après la désignation du DB, vous indiquez le mot-clé « D », suivi du préfixe de taille (par exemple `X` pour `BIT`) et de l'adresse d'octet (par exemple `13.1`).



Adressage

Comme le montre la figure 12-7, vous spécifiez l'adressage en indiquant la désignation du DB, suivi du préfixe de taille et de l'adresse.

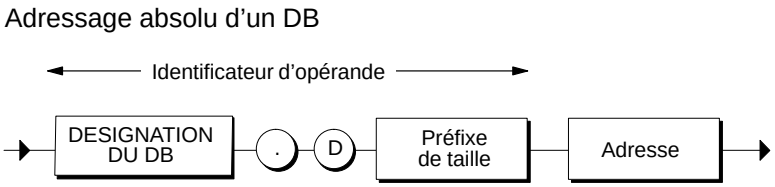


Figure 12-7 *Syntaxe : adressage absolu d'un bloc de données*

Préfixe de taille

Il précise la longueur de la zone de mémoire qui doit être adressée dans le bloc de données, par exemple un octet ou un mot. Si vous ne souhaitez spécifier qu'un bit, l'indication du préfixe de taille est optionnelle. La figure 12-8 représente la syntaxe du préfixe de taille.

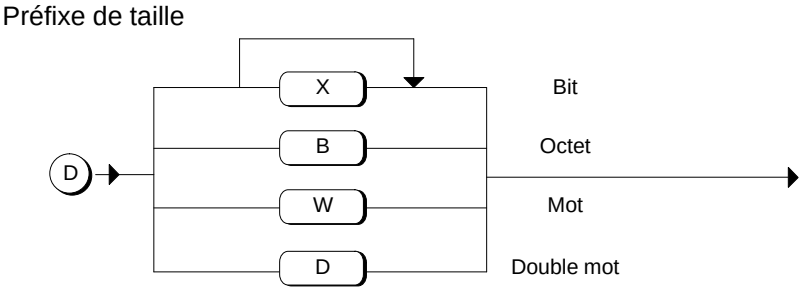


Figure 12-8 *Syntaxe : préfixe de taille*

Adresse

Selon le préfixe de taille utilisé, vous indiquez une adresse absolue désignant un bit, un octet, un mot ou un double mot, comme le montre la figure 12-9. Si vous n'avez spécifié qu'un « bit », vous avez la possibilité d'indiquer une adresse binaire supplémentaire. Le premier numéro désigne alors d'adresse d'octet, le second l'adresse du bit.

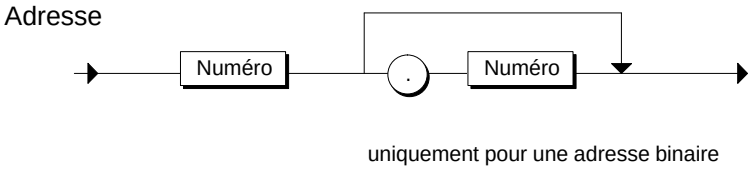


Figure 12-9 *Syntaxe : adresse*

Exemples

Voici quelques exemples d'adressage absolu d'un bloc de données. Dans la première partie, le bloc de données est indiqué de manière absolue, dans la seconde il l'est de manière symbolique :

```
OCTET_ETAT          := DB101.DB10;
ETAT_3              := DB30.D1.1;
Valeur_mesure       := DB25.DW20;

OCTET_ETAT          := Données_état.DB10;
ETAT_3              := ”Nouvelles données” D1.1;
Valeur_mesure       := Données_mesure.DW20;
ETAT_1              := WORD_TO_BLOCK_DB(INDEX).DW10;
```

Exemple 12-4 Adressage absolu

12.8 Adressage indexé des blocs de données

Adressage indexé

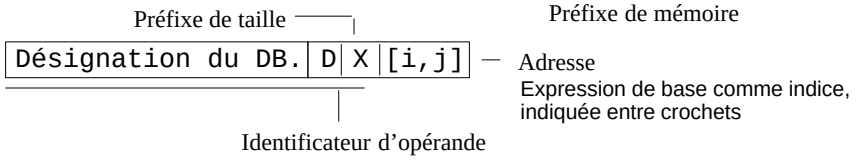
Vous avez également la possibilité d'adresser les blocs de données globaux de manière indexée. Par rapport à l'adressage absolu, l'avantage réside dans le fait que vous adressez le DB de manière dynamique, en utilisant des indices variables. Vous pouvez par exemple utiliser la variable de contrôle d'une boucle FOR comme indice.

L'adressage indexé d'un bloc de données est similaire à l'adressage absolu. Seule l'indication de l'adresse est différente.

A la place de l'adresse, vous indiquez un indice qui peut être une constante, une variable ou une expression arithmétique.

Identificateur absolu

Dans l'adressage indexé, l'identificateur absolu est formé de l'identificateur d'opérande (voir paragraphe 12.7) ainsi que d'une expression de base pour l'indexage.



Règles à appliquer à d'adressage indexé

L'indexation doit répondre aux règles suivantes :

- Chaque indice doit être une expression arithmétique du type de données `INT`.
- Dans le cas d'un adressage du type de données `BYTE`, `WORD` ou `DWORD`, vous devez utiliser exactement un indice. Celui-ci sera interprété comme adresse d'octet. La longueur d'adressage sera déterminée par le préfixe de taille.
- Dans le cas d'un adressage du type de données `BOOL`, vous devez utiliser deux indices. Le premier spécifie l'adresse d'octet, le second la position du bit dans cet octet.

```
ETAT_1:= DB11.DW[COMPTEUR];
ETAT_2:= DB12.DW[NO_W, NO_BIT];

ETAT_1:= Base_données1.DW[COMPTEUR];
ETAT_2:= Base_données2.DW[NO_W, NO_BIT];
ETAT_1:= WORD_TO_BLOCK_DB(INDEX).DW[COMPTEUR];
```

Exemple 12-5 Adressage indexé

12.9 Adressage structuré des blocs de données

Principe

Vous réalisez l'adressage structuré en affectant une valeur à une variable de même type.

```
HEURE_1:= DB11.HEURE_JOUR ;
```

Variable simple
Désignation du DB
Variable de même type

Pour référencer la variable dans le bloc de données, vous indiquez la désignation du DB, puis séparé par un point, le nom de la variable simple. La syntaxe est représentée à la figure 12-6.

Une variable simple est une variable à laquelle vous avez affecté un type de données simple ou complexe lorsque vous l'avez déclarée.

Exemples

```
Dans la section de déclaration du FB10 :
VAR
Résultat:         STRUCT ERG1 : INT;
                      ERG2 : WORD;
                  END_STRUCT
END_VAR
```

```
Type de données utilisateur UDT1 :
TYPE UDT1         STRUCT ERG1 : INT;
                      ERG2 : WORD;
                  END_STRUCT
```

```
DB20 avec type de données utilisateur :
DB20
UDT1
BEGIN ...
```

```
DB30 sans type de données utilisateur :
DB30              STRUCT ERG1 : INT;
                      ERG2 : WORD;
                  END_STRUCT
BEGIN ...
```

Exemple 12-6 Déclaration des données pour les blocs de données

```
Bloc fonctionnel avec les modes d'adressage :
..
FB10.DB10();
MOTERG_A          := DB10.Résultat.ERG2;
MOTERG_B          := DB20.ERG2;
MOTERG_C          := DB30.ERG2; Message d'erreur !
```

Exemple 12-7 Adressage des données des blocs de données

Expressions, opérateurs et opérandes

13

Présentation

Une expression représente une valeur qui sera calculée durant le processus de compilation ou l'exécution du programme. Elle comporte des opérandes (par exemple des constantes, des variables ou des valeurs de fonctions) et des opérateurs (par exemple *, /, +, -).

Le type d'expression est déterminé par le type de données des opérandes et les opérateurs utilisés. SCL distingue les :

- expressions arithmétiques,
- expressions de puissances,
- expressions de comparaison,
- expressions logiques.

Structure du chapitre

13.1 Opérateurs

Présentation

Les expressions sont formées d'opérateurs et d'opérandes. La plupart des opérateurs de SCL combinent deux opérandes et sont ce de fait désignés d'opérateurs *binaires.* Les autres opérateurs n'utilisent qu'un opérande et sont donc des opérateurs *unaires.*

Un opérateur binaire s'inscrit entre deux opérandes (par exemple A + B). Un opérateur unaire précède immédiatement son opérande (par exemple -B).

C'est la priorité des opérateurs, indiquée dans le tableau 13-1, qui détermine l'ordre de calcul. Le chiffre « 1 » correspond à la priorité la plus élevée.

Tableau 13-1 Liste des opérateurs

Classes d'opérateurs

Classe	Opérateur	Représentation	Priorité
Opérateur d'affectation *Cet opérateur inscrit une valeur dans une variable*	Affectation	: =	11
Opérateurs arithmétiques *Ces opérateurs permettent d'effectuer des calculs mathématiques*	Puissance	**	2
	Opérateurs unaires		
	Plus unaire	+	3
	Moins unaire	-	3
	Opérateurs arithmétiques de base		
	Multiplication	*	4
	Fonction Modulo	MOD	4
	Division d'entiers	DIV	4
	Addition	+	5
	Soustraction	-	5
Opérateurs de comparaison *Ces opérateurs permettent de formuler des conditions*	Infériorité	<	6
	Supériorité	>	6
	Infériorité ou égalité	<=	6
	Supériorité ou égalité	>=	6
	Egalité	=	7
	Différence	<>	7
Opérateurs logiques *Ces opérateurs sont utilisés dans des expressions logiques*	Négation (unaire)	NOT	3
	Opérations logiques de base		
	Et	AND ou &	8
	Ou exclusif	XOR	9
	Ou	OR	10
Parenthèses	(**expression**)	( )	1

13.2 Syntaxe d'une expression

Présentation

Les diverses expressions sont représentées dans le diagramme syntaxique de la figure 13-1. Puisque les expressions arithmétiques, expressions logiques, expressions de comparaison et expressions de puissances présentent quelques particularités, elles seront traitées séparément dans les paragraphes 13.3 à 13.6.

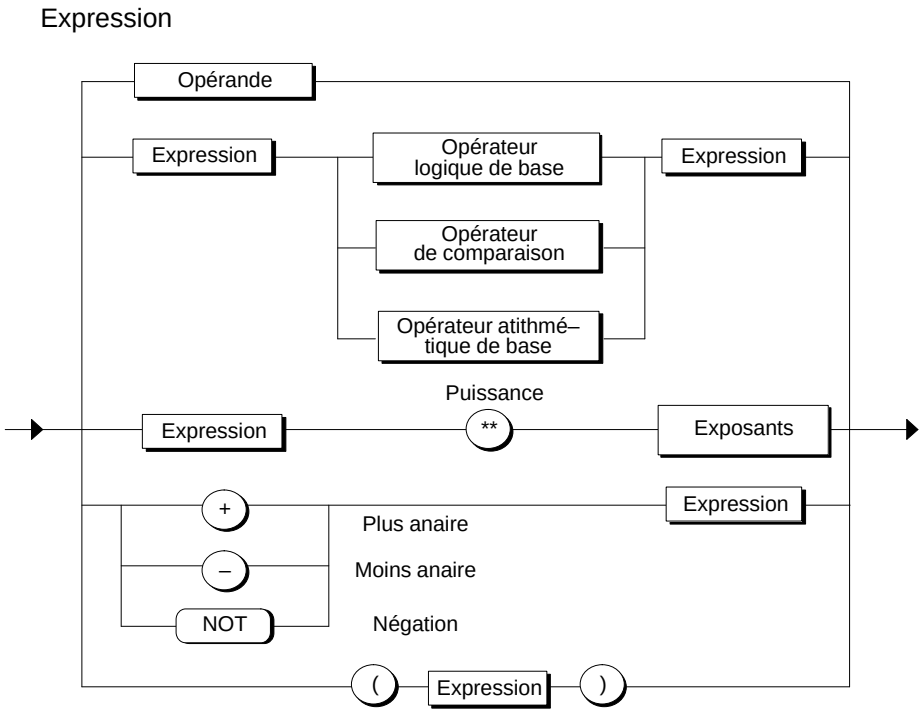


Figure 13-1 *Syntaxe : expression*

Résultat d'une expression

Le résultat d'une expression peut être utilisé :

- pour être affecté à une variable,
- comme condition pour une instruction de contrôle,
- comme paramètre pour l'appel d'une fonction ou d'un bloc fonctionnel.

Ordre de calcul

L'ordre de calcul de l'expression est déterminé par :

- la priorité des opérateurs présents,
- leur position de gauche à droite,
- les parenthèses utilisées dans le cas d'opérateurs de même priorité.

Règles

Le calcul d'expressions applique des règles précises :

- Les opérateurs sont traités dans leur ordre hiérarchique.
- Les opérateurs de même priorité sont traités de gauche à droite.
- Faire précéder un identificateur d'un signe moins équivaut à le multiplier par -1.
- Les opérateurs arithmétiques ne peuvent pas être immédiatement consécutifs. Ainsi, l'expression a * -b n'est pas autorisée, par contre a * (-b) l'est.
- L'utilisation de parenthèses peut annuler la priorité entre des opérateurs, car les parenthèses possèdent la priorité la plus élevée.
- Une expression entre parenthèses est considérée comme un opérateur unique et est toujours calculée en premier.
- Le nombre de parenthèses d'ouverture doit être équivalent à celui des parenthèses de fermeture.
- Les opérateurs arithmétiques ne peuvent s'appliquer à des caractères ou à des données logiques. Ainsi des expressions comme 'A'+'B' et (n<=0) + (n<0) sont erronées.

Exemples

Voici comment peuvent s'écrire diverses expressions :

```
EB10                          // Opérande
A1 AND (A2)                   // Expression logique
(A3) < (A4)                   // Expression de comparaison
3+3*4/2                       // Expression arithmétique

VALEUR_MESURE**2              // Expression de puissance
(DIFFERENCE)**DB10.EXPOSANT   // Expression de puissance
(SOMME)**FC100(..)            // Expression de puissance
```

Exemple 13-1 Diverses expressions

13.2.1 Opérandes

Présentation

Les opérandes sont des objets avec lesquels vous pouvez former des expressions. Ils sont représentés par le diagramme syntaxique de la figure 13-2.

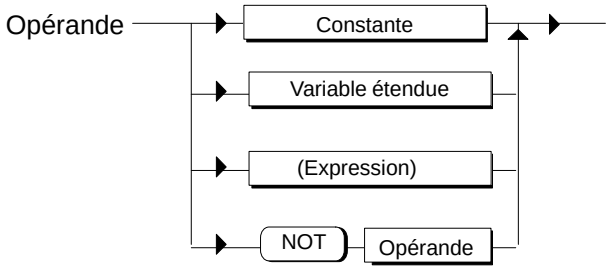


Figure 13-2 *Syntaxe : opérande*

Constante

Vous pouvez utiliser une constante sous forme de valeur numérique, de mnémonique ou de chaîne de caractères.

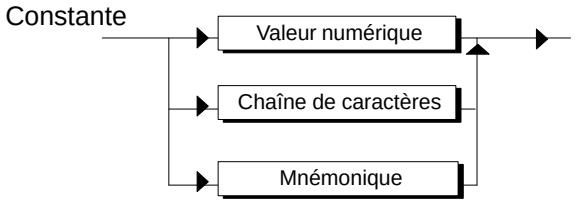


Figure 13-3 *Syntaxe : constantes*

Voici des exemples de constantes autorisées :

```
4_711
4711
30.0
'CARACTERE'
FACTEUR
```

Variable étendue

Une variable étendue est un concept général pour désigner une série de variables qui seront traitées plus en détails au chapitre 14.

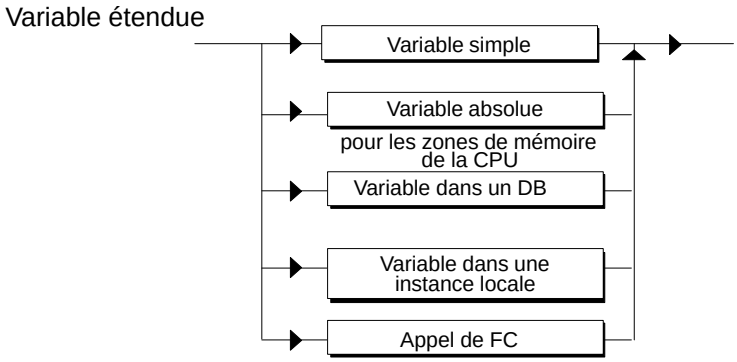


Figure 13-4 *Syntaxe : variable étendue*

Exemples de variables étendues

Voici des exemples de variables autorisées :

```
VALEUR_CONSIGNE      Variable simple
EW10                 Variable absolue
E100.5               Variable absolue
DB100.DW[INDICE]     Variable dans un DB
MOTEUR.REGIME        Variable dans une instance locale
SQR(20)              Fonction standard
FC192(VAL_CONSIGNE)  Appel de fonction
```

Exemple 13-2 Variables étendues dans des expressions

Nota

Lors d'un appel de fonction, le résultat du calcul, à savoir la valeur en retour remplace la désignation de la fonction dans l'expression. Les fonctions VOID qui ne possèdent pas de valeur en retour ne peuvent donc **pas** être utilisées comme opérandes dans des expressions.

13.3 Expressions arithmétiques

Définition

Une expression arithmétique est formée d'opérateurs arithmétiques et permet de traiter des types de données numériques.

Opérateurs arithmétiques de base

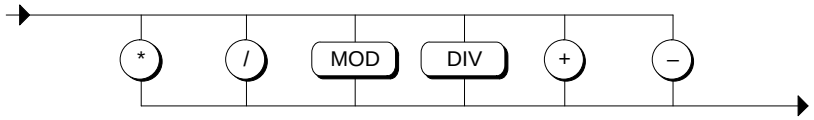


Figure 13-5 *Syntaxe : opérateurs arithmétiques de base*

Opérations arithmétiques

Le tableau 13-2 représente les opérations arithmétiques possibles et indique à quel type de données leur résultat est affecté, en fonction des opérandes utilisés.

`ANY_INT` représente les types de données `INT, DINT`

`ANY_NUM` représente les types de données `ANY_INT et REAL`

Tableau 13-2 Opérateurs arithmétiques

Opération	Opérateur	1er opérande	2nd opérande	Résultat [1]	Priorité
Puissance	**	ANY_NUM	INT	REAL	2
Plus unaire	+	ANY_NUM	-	ANY_NUM	3
		TIME	-	TIME	
Moins unaire	-	ANY_NUM	-	ANY_NUM	3
		TIME	-	TIME	
Multiplication	*	ANY_NUM	ANY_NUM	ANY_NUM	4
		TIME	ANY_INT	TIME	
Division	/	ANY_NUM	ANY_NUM	ANY_NUM	4
		TIME	ANY_INT	TIME	
Division d'entiers	DIV	ANY_INT	ANY_INT	ANY_INT	4
		TIME	ANY_INT	TIME	
Division Modulo	MOD	ANY_INT	ANY_INT	ANY_INT	4
Addition	+	ANY_NUM	ANY_NUM	ANY_NUM	5
		TIME	TIME	TIME	
		TOD	TIME	TOD	
		DT	TIME	DT	
Soustraction	-	ANY_NUM	ANY_NUM	ANY_NUM	5
		TIME	TIME	TIME	
		TOD	TIME	TOD	
		DATE	DATE	TIME	
		TOD	TOD	TIME	
		DT	TIME	DT	
		DT	DT	TIME	

[1] *Il est à noter que le type du résultat est déterminé par le type d'opérande le plus puissant.*

Règles

L'ordre de traitement des opérateurs dans une expression arithmétique est fonction de leur priorité (voir tableau 13-2).

- Pour une meilleure lisibilité, il est recommandé d'indiquer les nombres négatifs entre parenthèses, même lorsque cela n'est pas indispensable.
- Pour des divisions avec deux opérandes entiers de type INT, les opérateurs "DIV" et "/" fournissent le même résultat (cf. exemple 13-3).
- Avec les opérateurs de division "/", "MOD" et "DIV", le second opérande doit être différent de zéro.
- Lorsque l'un des opérandes est du type INT (entier) et le second du type REAL (nombre à virgule flottante), le résultat est toujours du type REAL.

Exemples

Les exemples suivants illustrent la formation d'expressions arithmétiques.

Soient i et j, deux variables de type entier, dont les valeurs respectives sont 11 et -3. La figure 13-3 montre quelques expressions avec des entiers et donne le résultat correspondant.

Expression	Valeur
i + j	8
i - j	14
i * j	-33
i DIV j	-3
i MOD j	2
i/j	-3

Exemple 13-3 Expressions arithmétiques

Soient i et j, deux variables entières dont les valeurs respectives sont 3 et -5. Le résultat de l'expression arithmétique représentée dans l'exemple 13-4, à savoir la valeur entière 7, est ensuite affecté à la variable VALEUR.

```
VALEUR:= i + i * 4 / 2 - (7+i) / (-j) ;
```

Exemple 13-4 Expression arithmétique

13.4 Exposants

Présentation

La figure 13-6 représente la formation d'un exposant d'une expression de puissance. (voir paragraphe 13.2). Il est à noter que vous pouvez en particulier utiliser des variables étendues pour la formation d'un exposant.

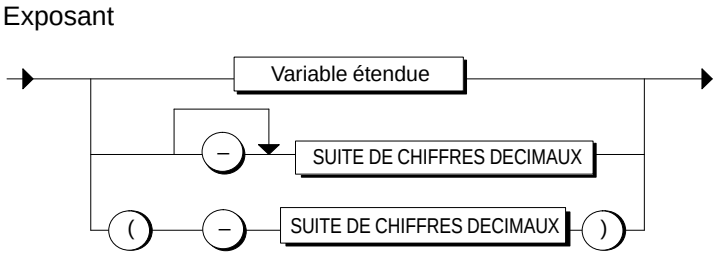


Figure 13-6 *Syntaxe : exposant*

```
VALEUR_MESURE**2             //Expression de puissance
(DIFFERENCE)**DB10.EXPOSANT  //Expression de puissance
(SOMME)**FC100               //Expression de puissance
```

Exemple 13-5 Expressions de puissance avec diverses exposants

13.5 Expressions de comparaison

Définition

Une expression de comparaison est une expression de type BOOL formée avec des opérateurs de comparaison. Il s'agit d'une combinaison d'opérandes du même type ou de la même classe de types avec les opérateurs représentés à la figure 13-7.

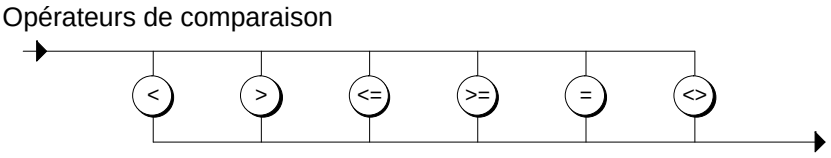


Figure 13-7 *Syntaxe : opérateurs de comparaison*

Opérateurs de comparaison

Les opérateurs de comparaison comparent la valeur numérique des deux opérandes.

`1er Opérande Opérateur 2nd Opérande ⇒ Valeur booléenne`

Le résultat de la comparaison est une valeur qui est soit vraie, soit fausse. La valeur est « vraie » (TRUE) si la comparaison est réalisée, « fausse » (FALSE) si elle ne l'est pas.

Règles

Pour former une expression de comparaison, vous devez appliquer les règles suivantes :

- Vous devriez indiquer les opérandes logiques entre parenthèses pour préciser l'ordre d'exécution des opérations logiques.
- Vous pouvez appliquer la logique booléenne pour combiner des expressions logiques et formuler des conditions du type « si a < b **et** b < c, alors... ». Comme expressions, vous pouvez utiliser des variables ou des constantes du type BOOL, de même que des expressions de comparaison.
- Pour les classes de types suivantes, toutes les variables peuvent être comparées :
  - `INT, DINT, REAL`
  - `BOOL, BYTE, WORD, DWORD`
  - `CHAR, STRING`
- Pour les types de temps suivants, seules les variables appartenant au même type peuvent être comparées :
  - `DATE, TIME, TOD, DT`
- La comparaison de caractères (type CHAR) s'effectue d'après la chaîne de caractères ASCII.
- Il n'est pas possible de comparer les variables S5TIME.
- Lorsque les deux opérandes sont du type `DT` ou `STRING`, vous devez les comparer avec la fonction CEI correspondante.

Exemples

Les exemples suivants illustrent la formation d'expressions de comparaison :

```
// Négation du résultat de l'expression de comparaison

      IF NOT (COMPTEUR > 5) THEN
            //...
            //...
      END_IF;

// Négation du résultat de la première expression de
// comparaison, puis combinaison avec le résultat
// de la seconde

      A:= NOT (COMPTEUR1 = 4) AND (COMPTEUR2 = 10) ;


// Combinaison OU de deux expressions de comparaison
      WHILE (A >= 9) OR (INTERROGATION <> 'n') DO.... ;
            //...
            //...
      END_WHILE;
```

Exemple 13-6 Expressions logiques

13.6 Expressions logiques

Définition

Une expression logique est formée d'opérateurs logiques (AND, &, XOR, OR) qui permettent de combiner des opérandes logiques (de type BOOL) ou des variables de type BYTE, WORD ou DWORD. La négation (c'est-à-dire l'inversion) de la valeur d'un opérande logique s'obtient avec l'opérateur NOT.

Opérateurs logiques de base

NOT n'est pas un opérateur logique de base ! L'opérateur agit comme un signe.

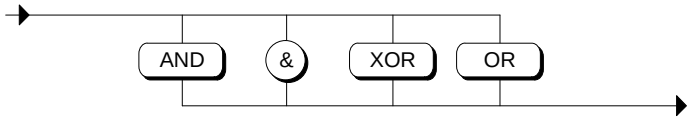


Figure 13-8 *Syntaxe : opérateurs logiques de base*

Opérateurs logiques

Le tableau 13-3 représente les expressions logiques disponibles et les types de données du résultat et des opérandes utilisés.

ANY_BIT représente les types de données BOOL, BYTE, WORD, DWORD

Tableau 13-3 Opérateurs logiques

Opération	Opérateur	1er opérande	2nd opérande	Résultat	Priorité
Négation	NOT	ANY_BIT	-	ANY_BIT	3
Combinaison ET	AND	ANY_BIT	ANY_BIT	ANY_BIT	8
Combinaison OU exclusif	XOR	ANY_BIT	ANY_BIT	ANY_BIT	9
Combinaison OU	OR	ANY_BIT	ANY_BIT	ANY_BIT	10

Résultats

Le résultat d'une expression logique peut être soit :

- 1 (*true*) ou 0 (*false*) dans le cas de la combinaison d'opérandes booléens

  soit

- un profil binaire après la combinaison binaire des deux opérandes.

Exemples

Soit `n` une variable entière avec la valeur 10 et `s` une variable de type caractère représentant la lettre A. Voici des expressions logiques combinant ces deux variables :

```
Expression                          Valeur
-------------------------------------------

( n>0 )   AND     ( n<20)           vrai
( n>0 )   AND     ( n<5 )           faux
( n>0 )   OR      ( n<5 )           vrai
( n>0 )   XOR     ( n<20)           faux
( n=10 )  AND     ( s='A')          vrai
( n<>5 )  OR      ( s>='A')         vrai
```

Exemple 13-7 Expressions logiques

Affectation de valeurs

14

Présentation

Une affectation de valeur permet d'affecter la valeur d'une expression à une variable, en écrasant la valeur précédente.

Structure du chapitre

Informations supplémentaires

SCL connaît les instructions simples et les instructions structurées. Les premières englobent, outre les affectations de valeurs, les instructions d'appel et l'instruction GOTO. Vous trouverez des informations à ce sujet dans les chapitres 15 et 16.

Les instructions de contrôle pour le branchement de programme et pour le traitement de boucles comptent parmi les instructions structurées. Des informations sont données à ce sujet au chapitre 15.

14.1 Présentation

Principe

L'affectation d'une valeur consiste à remplacer la valeur actuelle d'une variable par une nouvelle valeur, donnée sous forme d'expression. Celle-ci peut comporter des identificateurs de fonctions (FC) qui vont être exécutées et fournir des valeurs correspondantes (valeurs en retour).

Comme l'illustre le diagramme syntaxique, l'expression est calculée dans la partie droite de l'affectation de valeur, le résultat étant enregistré dans la variable dont le nom figure à gauche du signe d'affectation. Les variables autorisées sont représentées à la figure 14-1.

Affectation de valeur

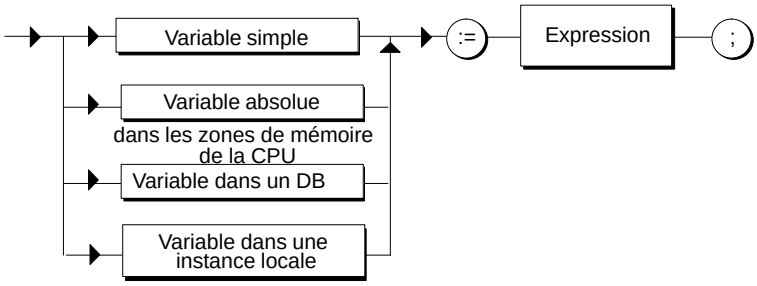


Figure 14-1 *Syntaxe : affectation de valeur*

Résultat

Le type d'une expression d'affectation correspond à celui de l'opérande à gauche.

14.2 Affectation de valeurs à des variables de type de données simple

Affectation

Toute expression et toute variable de type de données simple peuvent être affectées à une autre variable de type identique.

```
Identificateur := expression;
Identificateur := variable de type de données élementaire;
```

Exemples

Les exemples suivants représentent des affectations de valeurs autorisées

```
FUNCTION_BLOCK FBx
VAR
      COMMUTATEUR_1      :INT;
      COMMUTATEUR_2      :INT;
      VALEUR_CONSIGNE_1  :REAL;
      VALEUR_CONSIGNE_2  :REAL;
      INTERROGATION_1    :BOOL;
      HEURE_1            :S5TIME;
      HEURE_2            :TIME;
      DATE_1             :DATE;
      HEURE_JOUR_1       :TIME_OF_DAY;
END_VAR

BEGIN
// Affecter une constante à une variable
      COMMUTATEUR_1      := -17;
      VALEUR_CONSIGNE_1  := 100.1;
      INTERROGATION_1    := TRUE;
      HEURE_1            :=TIME#1H_20M_10S_30MS;
      HEURE_2            :=TIME#2D_1H_20M_10S_30MS;
      DATE_1             :=DATE#1996-01-10;
// Affecter une variable
      VALEUR_CONSIGNE_1  := VALEUR_CONSIGNE_2;
      COMMUTATEUR_2_     := COMMUTATEUR_1;
// Affecter une expression à une variable
      COMMUTATEUR_2:= COMMUTATEUR_1 * 3;
END_FUNCTION_BLOCK
```

Exemple 14-1 Affectation de valeurs à des variables de type simple

14.3 Affectation de valeurs à des variables de type STRUCT et UDT

Variables de type STRUCT et UDT

Les variables de type STRUCT et UDT sont des variables structurées qui désignent une structure complète ou l'une de ces composantes.

Voici des indications correctes de variables comme structures :

```
Image                 //Identificateur de structure
Image.élément         //Identificateur de composante de
                      //structure
Image.tableau         //Identificateur de tableau simple
                      //dans une structure
Image.tableau[2,5]    //Identificateur d'une composante de
                      //tableau dans une structure
```

Affectation d'une structure complète

Une structure complète ne peut être affectée à une autre structure que si leurs composantes possèdent les mêmes types de données et les mêmes noms. Voici une affectation correcte :

```
Nom_structure_1:=Nom_structure_2;
```

Affectation de composantes de structure

A toute composante de structure vous pouvez affecter une variable, une expression ou une autre composante de structure de type compatible. Voici des affectations correctes :

```
Nom_structure_1.élement1        := Valeur;
Nom_structure_1.élement1        := 20.0;
Nom_struct_1.élem1              := Nom_struct_2.élemt1;
Nom_strc_1.nom_tabl1            := Nom_strc_2.nom_tabl2;
Nom_structure_1.nom_tableau[10] := 100;
```

Exemples

L’exemple suivant illustre des affectations de valeurs de données de structure :

```
FUNCTION_BLOCK FB10
VAR
      VAR_AUX            :REAL;
      VALEUR_MESURE      :STRUCT   //Structure cible
                  TENSION:REAL;
                  RESISTANCE:REAL;
                  Tableau_simple:ARRAY[1..2,1..2]OFINT;
      END_STRUCT;

      VALEUR_EFFECTIVE   :STRUCT   //Structure source
                  TENSION: REAL;
                  RESISTANCE: REAL;
                  Tableau_simple:ARRAY[1..2,1..2]OFINT;
      END_STRUCT
END_VAR
BEGIN
//Affecter une structure complète à
//une structure complète
      VALEUR_MESURE:= VALEUR_EFFECTIVE;

//Affecter une composante de structure à
//une composante de structure
      VALEUR_MESURE.TENSION:= VALEUR_EFFECTIVE.TENSION;

// Affecter une composante de structure à
// une variable de même type
      VAR_AUX:= VALEUR_EFFECTIVE.RESISTANCE;

// Affecter une constante à
// une composante de structure
      VALEUR_MESURE.TENSION:= 4.5;

// Affecter une constante à un élément
// d’un tableau simple
      VALEUR_MESURE.TABLEAU_SIMPLE[1,2]:= 4
END_FUNCTION_BLOCK
```

Exemple 14-2 Affectations de valeurs à des variables de type STRUCT

14.4 Affectation de valeurs à des variables de type ARRAY

Variable de type tableau

Un tableau est formé de 1 à 6 dimensions et composantes de tableau au maximum et comporte des composantes de type identique. Deux possibilités s'offrent à vous pour affecter un tableau à une variable :

Vous pouvez référencer des **tableaux complets** ou des **tableaux partiels**. Dans le premier cas, vous indiquez le nom du tableau.

```
nom_tableau_1
```

Pour adresser une composante de tableau unique, vous indiquez le nom du tableau, suivi des valeurs d'indices entre crochets. A chaque dimension correspond un indice. Ils doivent être séparés par des virgules et indiqués entre crochets. Un indice doit être une expression arithmétique de type `INT`.

```
nom_tableau_1[2]
nom_tableau_1[4,5]
```

Affectation d'un tableau complet

Vous pouvez affecter un tableau complet à un autre tableau. Il faut pour cela que les types de données des composantes et les limites des tableaux (indices minimaux et maximaux) soient identiques. Voici un exemple d'affectation correct :

```
nom_tableau_1:=nom_tableau_2;
```

Affectation d'une composante de tableau

Pour affecter des valeurs à un tableau partiel, vous indiquez les indices entre crochets après le nom du tableau, en en omettant certains, en commençant par la droite. Vous adressez ainsi un tableau partiel, dont le nombre de dimensions correspond au nombre d'indices omis.

Il en résulte que bien que vous pouvez référencer des lignes et des composantes individuelles dans une matrice, vous ne pouvez pas référencer de colonnes successives (c'est-à-dire de ... à).

Voici des exemples d'affectation correcte :

```
nom_tableau_1[i]:=nom_tableau_2[j];
nom_tableau_1[i]:=expression;
identificateur_1 :=non_tableau_1[i];
```

Exemples

Les exemples suivants illustrent les affectations de valeurs à des tableaux :

```
FUNCTION_BLOCK FB3
VAR
      VALEURS_CONSIGNE  :ARRAY [0..127] OF INT;
      VALEURS_EFFECTIVES:ARRAY [0..127] OF INT;

// Déclaration d'une matrice
// (=tableau bidimensionnel)
// comportant 3 lignes et 4 colonnes
      REGULATEUR: ARRAY [1..3, 1..4] OF INT;

// Déclaration d'un vecteur
// (=tableau unidimensionnel)
// avec 4 composantes
      REGULATEUR_1:  ARRAY [1..4] OF INT;
END_VAR

BEGIN
// Affectation d'un tableau complet
// à un tableau
      VALEURS_CONSIGNE:=VALEURS_EFFECTIVES;

// Affectation d'un vecteur à la seconde ligne
// du tableau REGULATEUR
      REGULATEUR [2]:= REGULATEUR_1;

// Affectation d'une composante de tableau à
// une composante du tableau REGULATEUR
      REGULATEUR [1,4]:= REGULATEUR_1 [4];
END_FUNCTION_BLOCK
```

Exemple 14-3 Affectation de tableaux

14.5 Affectation de valeurs à des variables de type STRING

Variable de type STRING

Une variable de type STRING contient une chaîne d'au maximum 254 caractères.

Affectation

A chaque variable du type de données STRING, vous pouvez affecter une autre variable de même type. Voici des exemples d'affectations correctes :

```
variable_chaîne_1 := const_litt_chaîne ;
variable_chaîne_1 := variable_chaîne_2 ;
```

Exemple

Les exemples suivants illustrent des affectations de valeurs à des variables de type STRING :

```
FUNCTION_BLOCK FB3
VAR
      AFFICHAGE_1 :  STRING[50] ;
      STRUCTURE1  :  STRUCT
                        AFFICHAGE_2 : STRING[100] ;
                        AFFICHAGE_3 : STRING[50] ;
      END_STRUCT;
END_VAR
BEGIN
// Affectation d'une constante à une variable
// de type STRING
      AFFICHAGE_1 := 'Erreur dans module 1' ;
// Affectation d'une composante de structure
// à une variable de type STRING
      AFFICHAGE_1 := STRUCTURE1.AFFICHAGE_3 ;
// Affectation d'une variable de type STRING
// à une variable de type STRING
      If AFFICHAGE_1 <> STRUCTURE.AFFICHAGE_3 THEN
         AFFICHAGE_1 := STRUCTURE.AFFICHAGE_3     ;
      END_IF;
END_FUNCTION_BLOCK
```

Exemple 14-4 Affectation de valeurs à des variables de type STRING

14.6 Affectation de valeurs à des variables de type DATE_AND_TIME

Variable DATE_AND_TIME

Le type de données DATE_AND_TIME définit une zone de 64 bits (8 octets) pour y indiquer la date et l'heure.

Affectation

A toute variable du type de données DATE_AND_TIME, vous pouvez affecter une autre variable ou constante de même type. Voici des exemples d'affectations correctes :

```
variable_dt_1 := constante littérale de type date et heure ;
variable_dt_1 := variable_dt_2 ;
```

Exemple

Les exemples suivants illustrent les affectations de valeurs à des variables de type DATE_AND_TIME :

```
FUNCTION_BLOCK FB3
VAR
      HEURE_1     : DATE_AND_TIME;
      STRUCTURE   : STRUCT
             HEURE_2 : DATE_AND_TIME ;
             HEURE_3 : DATE_AND_TIME ;
      END_STRUCT;

END_VAR
BEGIN
// Affectation d'une constante
// à une variable de type DATE_AND_TIME

HEURE_1 := DATE_AND_TIME#1995-01-01-12:12:12.2 ;
STRUCTURE.HEURE_3 := DT#1995-02-02-11:11:11 ;

// Affectation d'un composante de structure
// à une variable de type DATE_AND_TIME

HEURE_1 := STRUCTURE.HEURE_2 ;


// Affectation d'une variable de type DATE_AND_TIME
// à un élément de structure DATE_AND_TIME
If HEURE_1 < STRUCTURE.HEURE_3 THEN
   STRUCTURE1.HEURE_3 := HEURE_1;
END_IF;

END_FUNCTION_BLOCK
```

Exemple 14-5 Affectation de valeurs à des variables de type DATE_AND_TIME

14.7 Affectation de variables absolues pour zones de mémoire

Variable absolue

Une variable absolue désigne les zone de mémoire autorisées d'une CPU. Comme décrit au chapitre 12, vous avez trois possibilités pour adresser ces zones afin de les affecter à des variables.

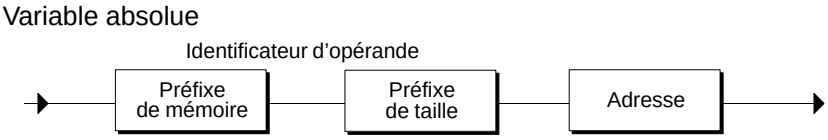


Figure 14-2 *Syntaxe : variable absolue*

Affectation

A l'exception de la périphérie d'entrée et de la mémoire image des entrées, vous pouvez affecter à toute variable absolue une variable ou une expression de même type.

Exemple

Les exemples suivants illustrent les affectations de valeurs à des identificateurs absolus :

```
FUNCTION_BLOCK_FB10
VAR
      MOT_ETAT1: WORD;
      MOT_ETAT2: BOOL;
      MOT_ETAT3: BYTE;
      MOT_ETAT4: BOOL;
      ADRESSE: INT:= 10;
END_VAR
BEGIN
      // Affectation d'un mot d'entrée
      // à une variable (adressage simple)
        MOT_ETAT1:= EW4 ;

      // Affectation d'un bit de sortie
      // à une variable (adressage simple)
        MOT_ETAT2:= a1.1 ;

      // Affectation d'un octet d'entrée
      // à une variable (adressage indexé)
        MOT_ETAT3:= EB[ADRESSE];

      // Affectation d'un bit d'entrée
      // à une variable (adressage indexé)
        FOR ADRESSE:= 0 TO 7 BY 1 DO
        MOT_ETAT4:= e[1,ADRESSE] ;
      END_FOR;
END_FUNCTION_BLOCK
```

Exemple 14-6 Affectations de valeurs à des variables absolus

14.8 Affectation à une variable globale

Variable dans un DB

L'adressage des variables globales dans des blocs de données s'effectue également par affectation d'une valeur à une variable de type identique ou inversement. Vous pouvez adresser ces données de manière structurée, absolue ou indexée (voir chapitre 12).

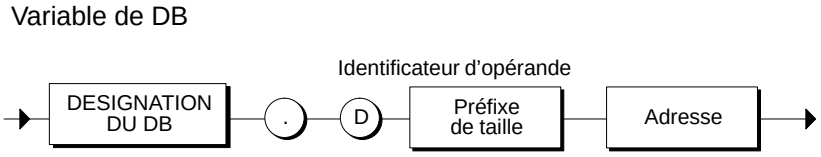


Figure 14-3 *Syntaxe : variables de DB*

Affectation

A toute variable globale, vous pouvez affecter une variable ou une expression de type identique. Voici des exemples d'affectation correcte :

```
DB11.DW10:=20;
DB11.DW10:=Etat;
```

Exemples

Dans l'exemple suivant, il est admis comme hypothèse, que dans le DB11 vous avez déclaré une variable " NOMBRE" de type INTEGER et une structure " NOMBRE1" avec la composante " NOMBRE2" de type INTEGER.

```
// Bloc de données DB11 requis
DATA_BLOCK DB11
STRUCT
      NOMBRE      :  INT:=1;
      NOMBRE1     :  STRUCT
                     NOMBRE2:INT := 256;
                     END_STRUCT;
MOT3  :           WORD:=W#16#aa;
MOT4  :           WORD:=W#16#aa;
MOT5  :           WORD:=W#16#aa;
MOT6  :           WORD:=W#16#aa;
MOT7  :           WORD:=W#16#aa;
MOT8  :           WORD:=W#16#aa;
MOT9  :           WORD:=W#16#aa;
MOT10:            WORD;
END_STRUCT

BEGIN
MOT10:=W#16#bb;
END_DATA_BLOCK
```

Exemple 14-7 Affectation de valeurs à des variables globales

Le bloc de données DB11 peut ensuite être utilisé de la manière suivante :

```
VAR
      REGULATEUR_1: ARRAY [1..4] OF INT;
      MOT_ETAT1   : WORD ;
      MOT_ETAT2   : ARRAY [1..4] OF INT;
      MOT_ETAT3   : INT ;
      ADRESSE     : INT ;
END_VAR
BEGIN
      // Affectation du mot 10 du DB11
      // à une variable (adressage simple)
      MOT_ETAT1:= DB11.DW10
      // La variable ”NOMBRE” du DB11
      // est affectée à la 1ère composante du tableau
      // (adressage structuré)
      REGULATEUR_1[1]:= DB11.NOMBRE;
       // Affectation de la composante ”NOMBRE2”
       // de la structure ”NOMBRE1”
       // à la variable MOT_ETAT3
      MOT_ETAT3:= DB11.NOMBRE1.NOMBRE2
      // Affectation d’un mot avec l’indice ADRESSE
      // du DB11 à une variable
      // (adressage indexé)
      FOR ADRESSE:= 1 TO 10 BY 1 DO
       MOT_ETAT2[ADRESSE]:= DB11.DW[ADRESSE] ;
      END_FOR;
```

Exemple 14-8 Affectation de valeurs à des variables globales de DB

Instructions de contrôle

15

Présentation

Les blocs que vous pouvez programmer de sorte à ce que toutes les instructions soient exécutées les unes après les autres, du début à la fin du bloc sont très peu nombreux. En règle générale, ce ne sont que certaines instructions (alternatives) qui sont exécutées ou encore répétées plusieurs fois (boucles) en fonction de certaines conditions. Ce sont des instructions de contrôle qui vous permettent de réaliser ce type de programmation dans un bloc SCL.

Structure du chapitre

15.1 Présentation

Instructions de sélection

Il est fréquent qu'en fonction de certaines conditions, diverses instructions d'un programme doivent être exécutées. Les instructions de sélection vous permettent de programmer de 2 à n alternatives afin de réaliser un branchement dans l'exécution du programme.

Tableau 15-1 Types de branchements

Type de branchement	Fonction
Instruction `IF`	L'instruction IF vous permet de réaliser le branchement de votre programme vers l'une parmi deux alternatives, en fonction d'une condition qui peut être soit vraie, soit fausse.
Instruction `CASE`	L'instruction CASE vous permet de réaliser le branchement de votre programme vers l'une parmi n alternatives. Il vous suffit de faire prendre à une variable, une valeur parmi n possibles.

Instructions d'itération

Vous pouvez programmer l'exécution de boucles en utilisant des instructions d'itération. Celles-ci vous permettent de définir quelles parties de votre programme doivent être répétées en fonction de certaines conditions.

Tableau 15-2 Types d'instructions réalisant l'exécution de boucles

Type de branchement	Fonction
Instruction `FOR`	Elle permet de répéter une suite d'instructions tant que la variable reste dans la plage des valeurs indiquée.
Instruction `WHILE`	Elle permet de répéter une suite d'instructions tant qu'une condition d'exécution est réalisée.
Instruction `REPEAT`	Elle permet de répéter une suite d'instructions jusqu'à ce qu'une condition d'arrêt soit réalisée.

Instructions de branchement

Un branchement dans un programme provoque le saut immédiat à un repère spécifié et par conséquement à une autre instruction dans le même bloc.

Tableau 15-3 Instructions de branchement

Type de branchement	Fonction
Instruction `CONTINUE`	Elle permet d'abandonner l'exécution de la boucle en cours.
Instruction `EXIT`	Elle permet de quitter une boucle à un endroit quelconque, que la condition d'arrêt soit remplie ou pas.
Instruction `GOTO`	Elle provoque le saut immédiat à un repère de saut défini.
Instruction `RETURN`	Elle permet de quitter le bloc en cours d'exécution.

Conditions

La condition est soit une expression de comparaison, soit une expression logique. Elle est du type BOOL et peut prendre les valeurs TRUE (vrai) ou FAlSE (faux).

Voici des exemples d'**expressions de comparaison** correctes :

```
COMPTEUR<=100
SQR(A)>0.005
Réponse = 0
SOLDE>=TRANSFERT
ch1< 'T'
```

Voici des exemples d'utilisation d'expressions de comparaison avec des opérateurs **logiques** :

```
(COMPTEUR<=100) AND (CH1<'*')
(SOLDE<100.0) OR (ETAT ='R')
(Réponse<0)OR((Réponse>5.0) AND (REPONSE<10.0))
```

Nota

Il est à noter que les opérandes logiques (ici les expressions de comparaison) figurent entre parenthèses afin d'éviter toute confusion dans l'ordre d'exécution.

15.2 Instruction IF

Principe

L'instruction IF est une instruction conditionnelle qui propose une ou plusieurs options pour sélectionner l'une (ou le cas échéant aucune) de ses sections des instructions.

Syntaxe

Instruction IF

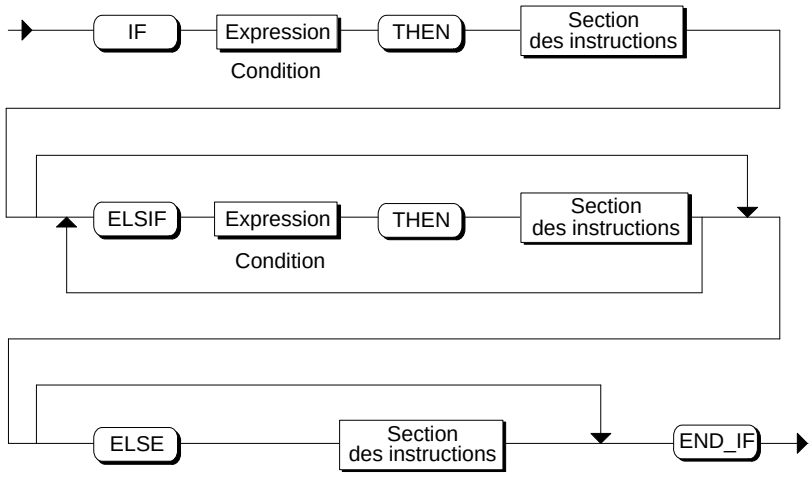


Figure 15-1 *Syntaxe : instruction IF*

L'exécution de l'instruction conditionnelle réalise l'évaluation des expressions logiques. Si la valeur d'une expression est *vraie*, alors la condition est réalisée, si par contre elle est *fausse*, celle-ci ne l'est pas.

Exécution

Règles appliquées à l'exécution de l'instruction IF :

1. Si la valeur de la première expression est *vraie*, la section des instructions après `THEN` est exécutée, sinon les expressions des branches `ELSIF` sont évaluées.
2. Si aucune des expressions booléennes des branches `ELSIF` n'est vraie, c'est la suite d'instructions après ELSE qui est exécutée (ou aucune suite d'instructions, au cas où la branche `ELSE` est absente).

Le nombre d'instructions ELSIF possibles est illimité.

Il est à noter que la branche `ELSIF` et/ou la branche `ELSE` sont facultatives. Leur absence revient à leur attribuer des instructions vides.

Nota

N'oubliez pas de terminer l'instruction `END_IF` par un point-virgule !

Nota

L'avantage d'utiliser une ou plusieurs branches ELSIF plutôt qu'une séquence d'instructions IF, réside dans le fait que les expressions logiques qui suivent une expression vraie ne sont plus évaluée, ce qui permet de diminuer la durée d'exécution d'un programme.

Exemple

L'exemple 15-1 illustre l'utilisation d'une instruction IF :

```
IF E1.1 THEN
      N:= 0;
      SUM:= 0;
      OK:= FALSE; // Affecter valeur FALSE au drapeau OK
ELSIF START = TRUE THEN
      N:= N + 1;
      SUM:= SUM + N;
ELSE
      OK:= FALSE;
END_IF;
```

Exemple 15-1 Instruction IF

15.3 Instruction CASE

Principe

L'instruction CASE permet de sélectionner 1 section de programme parmi n. Cette sélection porte sur la valeur en cours d'une expression de sélection.

Syntaxe

Instruction CASE

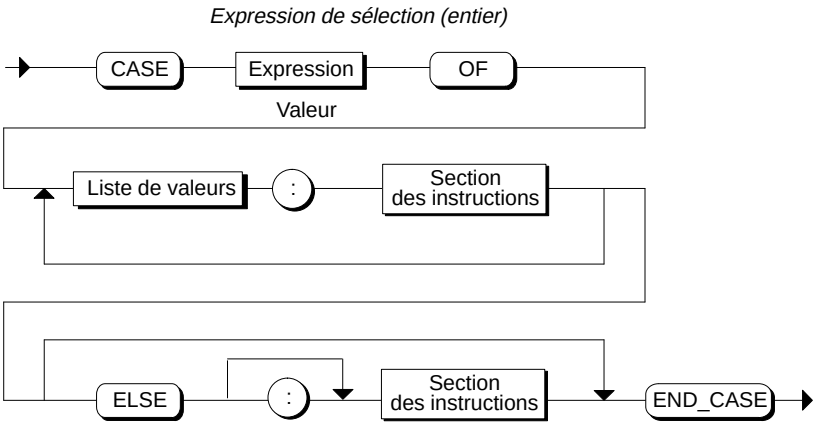


Figure 15-2 *Syntaxe : instruction CASE*

Exécution

Règles appliquées à l'exécution de l'instruction CASE :

1. Lors de l'exécution de l'instruction CASE, le programme vérifie si la valeur de l'expression de sélection figure dans une liste de valeurs donnée. Chacune des valeurs de cette liste correspond à l'une des valeurs autorisées pour l'expression de sélection. Cette dernière doit fournir une valeur de type INTEGER.
2. Si tel est le cas, la section des instruction affectée à la liste est exécutée.
3. La branche ELSE est optionnelle. Elle est exécutée si le résultat de la comparaison n'est pas celui attendu.

Nota

N'oubliez pas de terminer l'instruction END_CASE par un point-virgule !

Liste de valeurs

Elle contient les valeurs autorisées pour l'expression de sélection.

Liste de valeurs

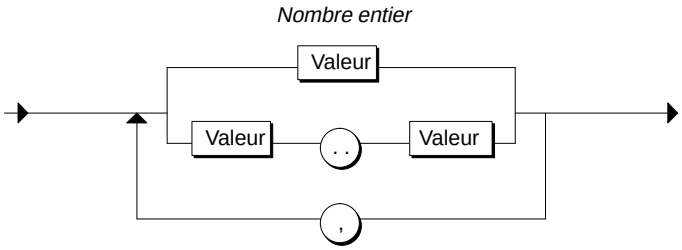


Figure 15-3 *Syntaxe : liste de valeurs*

Règles

Lorsque vous formulez la liste de valeurs, vous devez tenir compte des règles suivantes :

- toute liste de valeur peut commencer par une constante, une liste de constantes ou une plage de constantes,
- les valeurs de la liste doivent être du type INTEGER,
- chaque valeur doit être unique,

Exemples

L'exemple 15-2 illustre l'utilisation de l'instruction CASE. Conformément à la règle, la variable TW est de type INTEGER.

```
CASE TW OF
      1:    AFFICHAGE    := TEMP_FOUR;
      2:    AFFICHAGE    := VITESSE_MOTEUR;
      3:    AFFICHAGE    := TARE_BRUTE;
                    AW4:= 16#0003;
      4..10:AFFICHAGE    := INT_TO_DINT (TW);
                    AW4:= 16#0004;
      11,13,19:AFFICHAGE:= 99;
                    AW4:= 16#0005;
ELSE:           AFFICHAGE:= 0;
                 TW_ERROR:= 1;
END_CASE;
```

Exemple 15-2 Instruction CASE

Nota

Veillez à ce que la durée d'exécution des boucles ne soit pas trop longue, car la CPU se mettrait à l'arrêt suite à un retard d'acquittement.

15.4 Instruction FOR

Principe

Une instruction FOR, encore appelée instruction d'itération exécute une suite d'instructions dans une boucle, alors que des valeurs sont continuellement affectées à une variable (variable de contrôle). La variable de contrôle doit être l'identificateur d'une variable locale de type INT ou DINT.

Instruction FOR

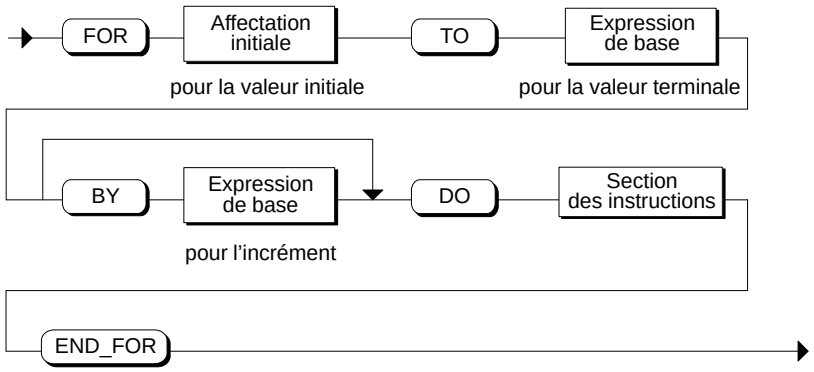


Figure 15-4 *Syntaxe : instruction FOR*

La définition d'une bouclce FOR inclut la détermination d'une valeur initiale et d'une valeur finale. Toutes deux doivent être du même type que la variable de contrôle.

Exécution

Règles appliquées à l'exécution de l'instruction FOR :

1. Au début de la boucle, une valeur initiale est affectée à la variable de contrôle qui est incrémentée ou décrémentée à chaque nouvelle itération de la boucle, jusqu'à ce que la valeur finale soit atteinte.
2. A chaque itération, la condition

   ```
   Valeur initiale <= |Valeur finale|
   ```

   est vérifiée. Si elle est remplie, la suite d'instructions est exécutée, dans le cas contraire la boucle et par conséquent la suite d'instructions sont sautées.

Nota

N'oubliez pas de terminer l'instruction `END_FOR` par un point-virgule !

Instruction d'initialisation

Pour former la valeur initiale de la variable de contrôle, vous disposez de l'instruction initiale représentée à la figure 15-5.

Instruction initiale

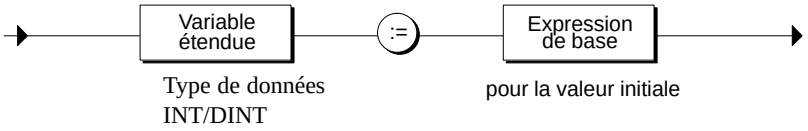


Figure 15-5 *Syntaxe : instruction d'initialisation*

Exemples :

```
FOR I  := 1 TO 20
FOR I  := 1 TO (valeur initiale +J)
```

Valeur finale et incrément

Pour former la valeur finale et l'incrément, vous pouvez respectivement créer une expression de base.

Règles

Les règles suivantes sont à appliquer pour l'instruction FOR :

- L'indication de *BY [incrément]* est facultative. En son absence, la valeur utilisée est +1.
- La valeur initiale, la valeur finale et l'incrément sont des expressions (voir chapitre 13) dont l'évaluation est effectuée une seule fois, au début de l'exécution de l'instruction FOR.
- Il n'est pas possible de modifier la valeur et l'incrément pendant l'exécution de la boucle.

Exemple

L'exemple 15-3 illustre l'utilisation de l'instruction FOR :

```
FUNCTION_BLOCK RECHERCHER
VAR
      INDEX       : INT;
      MOT_CLE     : ARRAY [1..50] OF STRING;
END_VAR

BEGIN
FOR INDEX:= 1 TO 50 BY 2 DO
      IF MOT_CLE [INDEX] = 'CLE' THEN
      EXIT;
      END_IF;
END_FOR;
END_FUNCTION_BLOCK
```

Exemple 15-3 Instruction FOR

15.5 Instruction WHILE

Principe

L'instruction WHILE permet de répéter l'exécution d'une suite d'instructions tant qu'une condition d'exécution est réalisée. Celle-ci applique les règles d'une expression logique.

Instruction WHILE

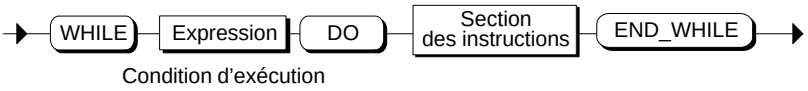


Figure 15-6 *Syntaxe : instruction WHILE*

La section des instructions qui suit `DO` est répétée tant que la condition d'exécution est TRUE.

Exécution

Règles appliquées à l'exécution de l'instruction `WHILE` :

1. La condition d'exécution est évaluée **avant** chaque exécution de la section des instructions.
2. Si la condition d'exécution d'exécution est TRUE, la section des instructions va être exécutée.
3. Si la condition d'exécution est FALSE, l'exécution de l'instruction WHILE est terminée, ce qui peut même déjà être le cas pour la première évaluation.

Nota

N'oubliez pas de terminer l'instruction `END_WHILE` par un point-virgule !

Exemple

L'exemple 15-4 illustre l'utilisation de l'instruction `WHILE` :

```
FUNCTION_BLOCK RECHECHER
VAR
      INDEX       : INT;
      MOT_CLE     : ARRAY [1..50] OF STRING;
END_VAR
BEGIN
INDEX:= 1;
WHILE INDEX <= 50 AND MOT_CLE[INDEX] <> 'CLE' DO
      INDEX:= INDEX + 2;
END_WHILE;
END_FUNCTION_BLOCK
```

Exemple 15-4 Instruction WHILE

15.6 Instruction REPEAT

Principe

L'instruction REPEAT répète l'exécution de la suite d'instructions entre REPEAT et UNTIL, jusqu'à ce que la condition d'abandon soit réalisée. Celle-ci applique les règles d'une expression logique.

Instruction REPEAT

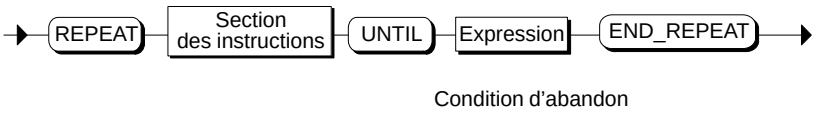


Figure 15-7 *Syntaxe : instruction REPEAT*

La condition est à chaque fois vérifiée **après** l'exécution de la suite d'instructions, qui est par conséquent exécutée au moins une fois, même si la condition d'abandon est réalisée dès le début.

Nota

N'oubliez pas de terminer l'instruction END_REPEAT par un point-virgule !

Exemple

L'exemple 15-5 illustre l'utilisation de l'instruction REPEAT

```
FUNCTION_BLOCK RECHERCHER
VAR
      INDEX       : INT;
      MOT_CLE     : ARRAY [1..50] OF STRING;
END_VAR

BEGIN
INDEX:= 0;
REPEAT
      INDEX:= INDEX + 2;
UNTIL
      INDEX > 50 OR MOT_CLE[INDEX] = 'CLE'
END_REPEAT;
END_FUNCTION_BLOCK
```

Exemple 15-5 Instruction REPEAT

15.7 Instruction CONTINUE

Principe

L'instruction CONTINUE permet d'interrompre momentanément une instruction d'itération (FOR, WHILE ou REPEAT) pour poursuivre dans la boucle .

Instruction CONTINUE

Figure 15-8 *Syntaxe : instruction CONTINUE*

La répétition éventuelle d'une suite d'instructions est déterminée par la condition initiale dans une boucle `WHILE`, et par la condition finale dans une boucle `REPEAT`.

Dans une instruction `FOR`, la variable de contrôle est incrémentée directement après une instruction `CONTINUE`.

Exemple

L'exemple 15-6 illustre l'utilisation de l'instruction CONTINUE :

```
FUNCTION_BLOCK CONTINUE
VAR
      INDEX       :INT;
      TABLEAU     :ARRAY[1..100] OF INT;
END_VAR
BEGIN
INDEX:= 0;
WHILE INDEX <= 100 DO
      INDEX:= INDEX + 1;
      // Si TABLEAU[INDEX] est égal à INDEX,
      // alors TABLEAU [INDEX] n'est pas modifié :
                  IF TABLEAU[INDEX] = INDEX THEN
                     CONTINUE;
                  END_IF;
                  TABLEAU[INDEX]:= 0;
      // Instructions suivantes
      //....
END_WHILE;
END_FUNCTION_BLOCK
```

Exemple 15-6 Instruction CONTINUE

15.8 Instruction EXIT

Principe

L'instruction EXIT permet de quitter une boucle (boucle FOR, WHILE ou REPEAT) à un endroit quelconque, que la condition d'abandon soit réalisée ou ne le soit pas.

Instruction EXIT

Figure 15-9 *Syntaxe : instruction EXIT*

L'instruction EXIT permet de quitter immédiatement l'instruction d'itération qui l'entoure directement.

L'exécution du programme se poursuit après la fin de la boucle d'itération (par exemple après `END_FOR`).

Exemple

L'exemple 15-7 illustre l'utilisation de l'instruction EXIT :

```
FUNCTION_BLOCK EXIT
VAR
      INDEX_1     := INT;
      INDEX_2     := INT;
      INDEX_RECHERCHE:= INT;
      MOT_CLE     : ARRAY[1..51] OF STRING;
END_VAR
BEGIN
INDEX_2           := 0;
FOR INDEX_1:= 1 TO 51 BY 2 DO
      // La boucle FOR est quittée si
      // MOT_CLE[INDEX_1] est égal à 'CLE' :
      IF MOT_CLE[INDEX_1] = 'CLE' THEN
                  INDEX_2:= INDEX_1;
                  EXIT;
      END_IF;
END_FOR;
// L'affectation de valeur suivante est effectuée
// après l'exécution de EXIT ou
// après la fin normale de la boucle FOR :
INDEX_RECHERCHE:= INDEX_2;
END_FUNCTION_BLOCK
```

Exemple 15-7 Instruction EXIT

15.9 Instruction GOTO

Principe

L'instruction GOTO vous permet de réaliser un branchement de programme. Elle provoque le saut immédiat à un repère de saut spécifié et par conséquent à une autre instruction dans le même bloc.

L'instruction GOTO ne doit être utilisée qu'à titre exceptionnel, par exemple pour la résolution d'erreurs. En effet, son utilisation est contraire aux règles de la programmation structurée.

Branchement de programme

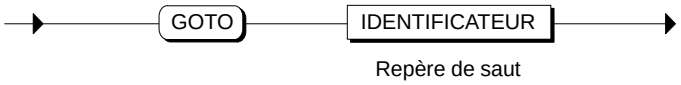


Figure 15-10 *Syntaxe : instruction GOTO*

Le repère de saut désigne un repère LABEL / END_LABEL dans la section de déclaration. Il précède l'instruction qui doit être exécutée immédiatement après GOTO.

Règles

Règles à appliquer pour utiliser l'instruction GOTO :

- Le branchement d'une instruction de saut doit se trouver dans le même bloc.
- Le branchement doit être univoque.
- Un saut dans un bloc de boucle n'est pas autorisé. Un saut depuis un bloc de boucle est possible.

Exemple L’exemple 15-8 illustre l’utilisation de l’instruction GOTO :

```
FUNCTION_BLOCKFB3//Ex_GOTO
VAR
      INDEX : INT;
      A     : INT;
      B     : INT;
      C     : INT;
      MOT_CLE: ARRAY[1..51] OF STRING;
END_VAR
LABEL
      REPERE1, REPERE2, REPERE3;
END_LABEL
BEGIN
IF A > B THEN GOTO REPERE1;
      ELSIF A > C THEN GOTO REPERE2;
END_IF;
//...
REPERE1     :     INDEX:= 1;
                  GOTO REPERE3;
REPERE2     :     INDEX:= 2;
//...
REPERE3     :     ;
//...
END_FUNCTION_BLOCK
```

Exemple 15-8 Instruction de branchement GOTO

15.10 Instruction RETURN

Principe

L'instruction RETURN permet de quitter le bloc en cours d'exécution (OB, FB, FC) et de revenir au bloc appelant ou au système d'exploitation, dans le cas d'un OB.

Instruction RETURN

Figure 15-11 *Syntaxe : instruction der RETURN*

Nota

A la fin de la section d'exécution d'un bloc de code ou de la section de déclaration d'un bloc de données, l'instruction RETURN est redondante, puisqu'elle est exécutée automatiquement.

Appel de fonctions et de blocs fonctionnels 16

Présentation

A partir d'un bloc SCL, vous avez la possibilité d'appeler d'autres fonctions (FC) ou blocs fonctionnels (FB). Vous pouvez appeler :

- des fonctions et blocs fonctionnels créés dans SCL,
- des fonctions et blocs fonctionnels programmés dans un autre langage de STEP 7 (par exemple LIST, CONT),
- des fonctions système (SFC) et blocs fonctionnels système (SFB), disponibles dans le système d'exploitation de votre CPU.

Structure du chapitre

16.1 Appel et transmission de paramètres

Transmission de paramètres

Lorsque vous appelez une fonction ou un bloc fonctionnel, un échange de données a lieu entre le bloc appelant et le bloc appelé. Les paramètres à transmettre doivent être indiqués sous forme de liste de paramètres lors de l'appel. Ils doivent figurer entre parenthèses et être séparés par des virgules.

Principe

L'exemple suivant illustre l'appel d'une fonction pour laquelle sont indiqués un paramètre d'entrée, un paramètre d'entrée/sortie et un paramètre de sortie.

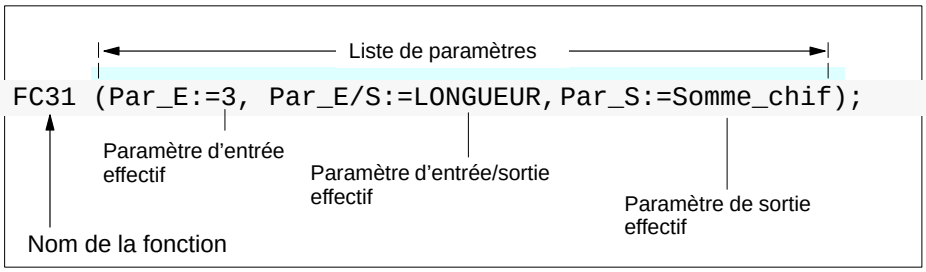


Figure 16-1 *Principe de la transmission de paramètres*

Comme le montre la figure 16-2, l'indication des paramètres a la forme d'une affectation de valeurs. En effet, elle consiste à affecter des valeurs (paramètres effectifs) aux paramètres que vous avez définis dans la section de déclaration du bloc appelé (paramètres formels).

Paramètres effectifs	Paramètres formels
3	⇒ Par_E
LONGUEUR	⇔ Par_E/S
Somme_chif	⇐ Par_S

Figure 16-2 *Affectation de valeurs dans la liste des paramètres*

Paramètres formels

Les paramètres formels sont ceux qui sont déclarés dans le bloc appelé. Leur unique fonction est de réserver la place pour les paramètres effectifs qui seront transmis au bloc lors de son appel. Vous les avez définis dans la section de déclaration du bloc (FB ou FC).

Tableau 16-1 Sections de déclaration autorisées pour les paramètres formels

Section de déclaration	Données	Mot-clé
Section des paramètres	Paramètres d'entrée	VAR_INPUT *Liste de déclaration* END_VAR
	Paramètres de sortie	VAR_OUTPUT *Liste de déclaration* END_VAR
	Paramètres d'entrée/sortie	VAR_IN_OUT *Liste de déclaration* END_VAR

16.2 Appel de blocs fonctionnels (FB ou SFB)

Instance globale et locale

Lorsque vous appelez un bloc fonctionnel dans SCL, vous pouvez utiliser

- aussi bien des blocs de données d'instance globaux,
- que des zones d'instances locales du bloc de données d'instance actuel.

C'est l'endroit où sont enregistrées les données qui distingue l'appel d'un FB comme instance locale de celui d'un FB comme instance globale. Dans le premier cas, les données ne sont pas enregistrées dans un DB particulier, mais dans le bloc de données d'instance du FB appelant.

Appel d'un FB

FB : bloc fonctionnel
SFB : bloc fonctionnel système

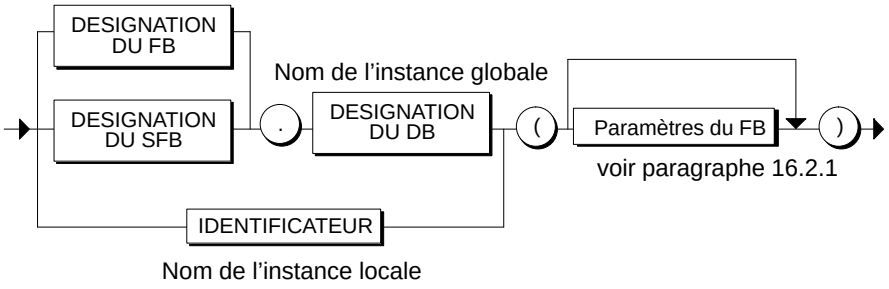


Figure 16-3 *Syntaxe : appel d'un bloc fonctionnel*

Appel comme instance globale

Dans l'instruction d'appel du FB, vous devez indiquer :

- le nom du bloc fonctionnel ou du bloc fonctionnel système (désignation du FB ou du SFB),
- le bloc de données d'instance (désignation du DB),
- l'affectation des paramètres (paramètres du FB).

L'appel d'une instance globale peut être absolu ou symbolique.

Appel absolu :

```
FB10.DB20 (X1:=5,X2:=78,......);
```

Affectation des paramètres

Appel symbolique :

```
MOTEUR.MARCHE (X1:=5,X2:=78,......);
```

Figure 16-4 *Appel du FB10 avec le bloc de données d'instance DB20*

Appel comme instance locale

Dans l’instruction d’appel du FB, vous devez indiquer :

- le nom de l’instance locale (IDENTIFICATEUR),
- l’affectation des paramètres (paramètres du FB),

L’appel d’une instance locale est toujours symbolique, par exemple :

```
MOTEUR (X1:=5,X2:=78,......);
                ↑
                Affectation des paramètres
```

Figure 16-5 *Appel d’une instance locale*

16.2.1 Paramètres du FB

Principe

Lorsque vous appelez un bloc fonctionnel – comme instance globale ou locale –, vous devez faire la distinction entre

- ses paramètres d'entrée et
- ses paramètres d'entrée/sortie

dans la liste des paramètres. Dans les deux cas, vous **affectez** les paramètres effectifs aux paramètres formels :

```
Paramètres      Paramètres
formels         effectifs
   Par_E   ⇐ 3          //Affectation de l'entrée
   Par_E/S ⇔ LONGUEUR //Affectation de l'entrée/sortie
```

Figure 16-6 *Affectation de valeurs dans la liste des paramètres*

Les paramètres de sortie ne sont pas indiqués à l'appel d'un FB, en revanche ils sont affectés après l'appel.

La syntaxe de l'affectation des paramètres du FB est la même qu'il s'agisse d'instances globales ou locales.

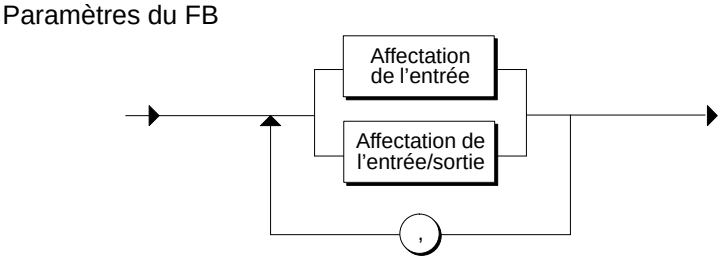


Figure 16-7 *Syntaxe : paramètres du FB*

Exemple

Voici un exemple d'appel de FB avec une affectation de l'entrée et une affectation de la sortie :

```
FB31.DB77(Par_E:=3, Par_E/S:=LONGUEUR);
```

Règles

Les règles suivantes s'appliquent à l'affectation des paramètres :

- L'ordre d'affectation est quelconque.
- Le type de données du paramètre effectif doit être le même que celui du paramètre formel.
- Chaque affectation est séparée de la précédente par une virgule.
- L'affectation de la sortie n'est pas possible dans un appel de FB. La valeur du paramètre de sortie que vous avez déclaré est enregistrée dans les données d'instance, où elle peut être adressée par chaque FB. Pour pouvoir la lire, vous devez définir son adressage à partir d'un FB (voir paragraphe 14.8).

Résultat après l'appel

Après l'exécution du bloc :

- les paramètres d'entrée actuels transmis restent inchangés,
- les valeurs des paramètres d'entrée/sortie transmises mais modifiées sont actualisées, à l'exception des paramètres d'entrée/sortie d'un type de données simple (voir le paragraphe 16.2.3),
- les paramètres de sortie du bloc appelant peuvent être lus dans le bloc de données d'instance ou dans la zone d'instances locale. Voir l'exemple 16-3.

16.2.2 Affectation de l’entrée (FB)

Principe

L’affectation de l’entrée consiste à affecter un paramètre effectif à un paramètre d’entrée formel. Le FB ne peut **pas** modifier ces paramètres effectifs. L’affectation du paramètre d’entrée actuel est facultative. Si vous ne l’indiquez pas, la valeur de l’appel précédent est conservée.

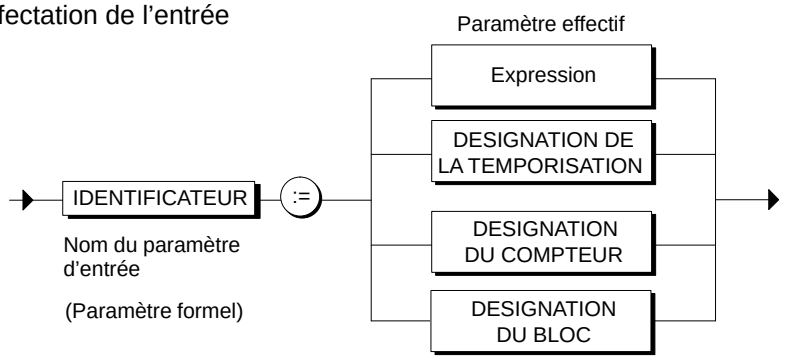


Figure 16-8 *Syntaxe : affectation de l’entrée*

Paramètres effectifs possibles

Les paramètres effectifs suivants peuvent être affectés à l’entrée :

Tableau 16-2 Paramètres effectifs dans les affectations d’entrées

Paramètre effectif	Explication
Expression	• expression arithmétique, logique ou de comparaison • constante • variable étendue
Désignation TEMPORISATION/COMPTEUR	Vous définissez une temporisation ou un compteur qui seront utilisés lors de l’exécution d’un bloc (voir aussi le chapitre 17).
Désignation BLOC	Vous définissez un bloc qui sera utilisé comme paramètre d’entrée. Le type de bloc (FB, FC, DB) doit être défini dans la déclaration du paramètre d’entrée. A l’affectation du paramètre, vous indiquez la désignation du bloc, absolue ou symbolique (voir aussi le chapitre 9).

16.2.3 Affectation de l’entrée/sortie (FB)

Principe

L’affectation de l’entrée/sortie consiste à affecter un paramètre effectif au paramètre d’entrée/sortie formel du FB appelé.

Contrairement aux paramètres d’entrée, les paramètres d’entrée/sortie peuvent être modifiés par le FB appelé. La nouvelle valeur que prend le paramètre lors de l’exécution du FB est inscrite dans le paramètre effectif, où elle écrase la valeur initiale.

Si des paramètres d’entrée/sortie sont déclarés dans le FB appelé, vous devez leur affecter une valeur au premier appel. L’indication de paramètres effectifs est ensuite facultative.

Affectation de l’entrée/sortie

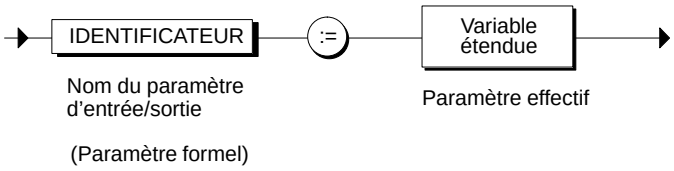


Figure 16-9 *Syntaxe : affectation de l’entrée/sortie*

Paramètres effectifs possibles

Puisque le paramètre effectif affecté peut être modifié comme paramètre d’entrée/sortie durant l’exécution du FB, il doit s’agir d’une variable. C’est la raison pour laquelle vous ne pouvez pas affecter un paramètre d’entrée à un paramètre d’entrée/sortie (la nouvelle valeur ne pourrait pas y être inscrite).

Tableau 16-3 Paramètres effectifs dans des affectations d’entrée/sortie

Paramètre effectif	Explication
Variable étendue	Vous pouvez utiliser les types de variables étendues suivants : variable simple et paramètre adressage de variable absolue adressage de bloc de données appel de fonction (voir aussi le chapitre 14)

Particularités

Tenez compte des particularités suivantes :

- La valeur modifiée du paramètre d'entrée/sortie est actualisée après l'exécution du bloc. Les paramètres d'entrée/sortie d'un type de données **simple** constituent cependant une exception. L'actualisation n'est réalisée que si un paramètre effectif a été attribué à l'appel du FB.
- Les paramètres effectifs suivants ne peuvent pas être affectés aux paramètres d'entrée/sortie d'un type de données **non simple** :
  - paramètre d'entrée/sortie d'un FB, ou
  - paramètre d'une FC
- **Paramètre ANY :** outre les restrictions ci-dessus, les constantes ne sont **pas** autorisées comme paramètres effectifs.

16.2.4 Exemple d'appel d'une instance globale

Principe

Un bloc fonctionnel comportant une boucle FOR pourrait se présenter comme dans l'exemple 16-1. Il y est supposé que le mnémonique TEST a été déclaré pour le FB17 dans la table des mnémoniques.

```
FUNCTION_BLOCK TEST
      VAR_INPUT
        VAL_FIN: INT; //Paramètre d'entrée
      END_VAR
      VAR_IN_OUT
        IQ1: REAL; //Param. d'entrée/sortie
      END_VAR
      VAR_OUTPUT
        CONTROLE: BOOL;//Paramètre de sortie
      END_VAR
      VAR
        INDEX: INT;
      END_VAR
BEGIN
      CONTROLE:= FALSE;
      FOR INDEX:= 1 TO VAL_FIN DO
        IQ1:= IQ1 * 2;
        IF IQ1 > 10000 THEN
           CONTROLE:= TRUE;
        END_IF;
      END_FOR;
END_FUNCTION_BLOCK
```

Exemple 16-1 Exemple de FB

Appel

Vous pouvez appeler ce FB de l'une des façons suivantes. On suppose que VARIABLE1 a été déclarée comme variable de type REAL dans le bloc appelant.

```
     //Appel absolu, instance globale :
FB17.DB10 (VAL_FIN:=10, IQ1:= VARIABLE1);

     //Appel symbolique, instance globale :
TEST.TEST_1 (VAL_FIN:= 10, IQ1:= VARIABLE1) ;
```

Exemple 16-2 Exemple d'appel de FB avec bloc de données d'instance

Résultat

A la fin de l'exécution du bloc, la valeur fournie pour le paramètre d'entrée/sortie IQ1 est disponible dans la VARIABLE1.

Lecture de la valeur de sortie

La lecture du paramètre de sortie CONTROLE peut être réalisée de plusieurs façons :

```
      //L’adressage du paramètre de sortie
      //est réalisé par
RESULTAT:= DB10.CONTROLE;
      //Vous avez également la possibilité d’affecter
      //directement le paramètre de sortie à un
      //paramètre d’entrée à l’appel d’un autre FB

FB17.DB12 (MARCHE_1:= DB10.CONTROLE);
```

Exemple 16-3 Résultat de l’appel du FB avec bloc de données d’instance

16.2.5 Exemple d’appel d’une instance locale

Principe

L’exemple 16-1 représente un bloc fonctionnel programmé avec une boucle FOR simple. On suppose que le mnémonique TEST a été déclaré pour le FB17 dans la table de mnémoniques.

Appel

En supposant que VARIABLE1 a été déclarée comme variable de type REAL dans le bloc appelant, vous pourriez appeler le FB comme suit :

```
// Appel, instance locale :
TEST_L (VAL_FIN:= 10, IQ1:= VARIABLE1) ;
```

Exemple 16-4 Exemple d’appel de FB comme instance locale

TEST_L doit avoir été déclarée dans la déclaration des variables :

```
VAR
  TEST_L : TEST;
END_VAR
```

Lecture des paramètres de sortie

La lecture du paramètre de sortie CONTROLE peut être réalisée de la manière suivante :

```
// L’adressage du paramètre de sortie est réalisé par
RESULTAT:= TEST_L.CONTROLE;
```

Exemple 16-5 Résultat de l’appel du FB comme instance locale

16.3 Appel de fonctions

Valeur en retour

Contrairement aux blocs fonctionnels, les fonctions fournissent un résultat, la valeur en retour. C'est la raison pour laquelle elles peuvent être traitées comme des opérandes. Les fonctions dont la valeur en retour est du type `VOID` constituent une exception.

Dans l'affectation de valeurs suivante, la fonction ECART est appelée avec des paramètres donnés :

```
LONGUEUR:= ECART (X1:=-3, Y1:=2);
```

`ECART` constitue la valeur en retour !

La fonction calcule la valeur en retour qui porte le même nom qu'elle et la transmet au bloc appelant, où elle remplace l'appel de la fonction.

Vous pouvez utiliser la valeur en retour dans les éléments suivants d'une FC ou d'un FB :

- une affectation de valeur,
- une expression logique, arithmétique ou de comparaison ou
- comme paramètre pour l'appel d'un autre bloc fonctionnel ou d'une autre fonction.

Les fonctions de type VOID constituent une exception. Elles ne possèdent pas de valeur en retour et ne peuvent donc pas être utilisées dans des expressions.

La figure 16-10 montre la syntaxe de l'appel d'une fonction :

Appel de fonction

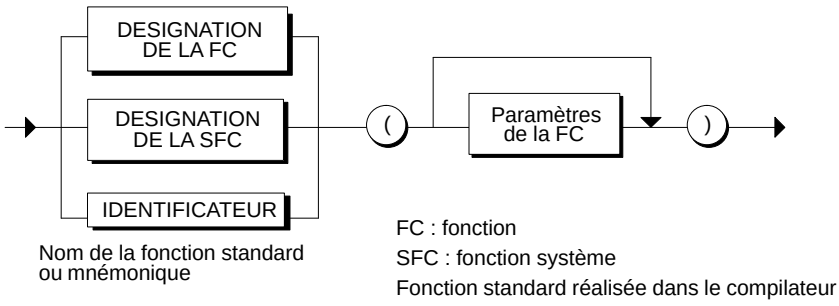


Figure 16-10 *Syntaxe : appel d'une fonction*

Nota

Si dans SCL vous appelez une fonction dont la valeur en retour n'a pas été affectée, il peut en résulter une exécution erronée du programme utilisateur.

Dans une fonction SCL, ce cas peut survenir lorsque bien que la valeur en retour ait été affectée, l'exécution du programme ne passe pas par l'instruction correspondante.
Dans une fonction LIST, CONT ou LOG, ce cas peut survenir lorsque la fonction a été programmée sans affectation de la valeur en retour ou lorsque l'exécution du programme ne passe pas par l'instruction correspondante.

Appel

Pour appeler une fonction, vous devez indiquer :

- son nom (DESIGNATION DE LA FC, DE LA SFC, IDENTIFICATEUR),
- la liste de paramètres.

Exemple

Le nom de la fonction qui désigne la valeur en retour peut être absolu ou symbolique, par exemple :

```
FC31 (X1:=5, Q1:= Somme_chiffres)
ECART (X1:=5, Q1:= Somme_chiffres)
```

Résultat de l'appel

Après l'appel d'une fonction, les résultats sont disponibles sous forme de

- valeur en retour ou
- paramètre de sortie et d'entrée/sortie (paramètre effectif).

De plus amples informations à ce sujet sont données au chapitre 18.

16.3.1 Paramètres de la FC

Principe

Contrairement aux blocs fonctionnels, les fonctions n'ont pas de mémoire dans laquelle ils pourraient enregistrer les valeurs des paramètres. Les données locales ne sont enregistrées que temporairement durant l'exécution de la fonction. C'est la raison pour laquelle, lorsque vous appelez une fonction, vous devez affecter des paramètres effectifs à tous les paramètres d'entrée, d'entrée/sortie et de sortie que vous avez déclaré dans la section de déclaration de la fonction.

La figure 16-11 montre la syntaxe de l'affectation de paramètres à une FC :

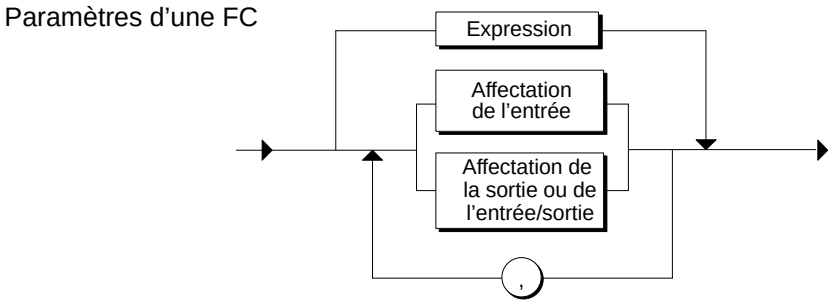


Figure 16-11 *Syntaxe : paramètres d'une FC*

Voici un exemple d'appel de fonction avec affectation des paramètre d'entrée, de sortie, d'entrée/sortie :

```
FC32 (Param1_E:=5,Param1_E/S:=LONGUEUR,
      Param1_S:=Somme_chiffres)
```

Règles

Les règles suivantes s'appliquent à l'affectation des paramètres :

- L'ordre d'affectation est quelconque.
- Le type de données du paramètre effectif doit être le même que celui du paramètre formel.
- Chaque affectation est séparée de la précédente par une virgule.

16.3.2 Affectation de l’entrée (FC)

Principe

L’affectation de l’entrée consiste à à affecter des paramètres effectifs aux paramètres d’entrée formels de la FC appelée. Celle-ci peut les utiliser sans toutefois pouvoir les modifier. L’affectation de l’entrée **est obligatoire** dans l’appel d’une FC, contrairement à ce qui est le cas pour l’appel d’un FB. Voici la syntaxe de l’affectation de l’entrée :

Affectation de l’entrée

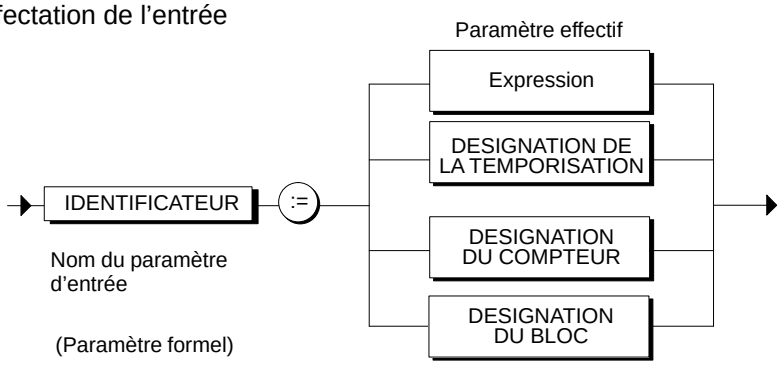


Figure 16-12 *Syntaxe : affectation de l’entrée*

Paramètres effectifs dans l’affectation d’une entrée

Les paramètres effectifs suivants peuvent être affectés à l’entrée :

Tableau 16-4 Paramètres effectifs dans l’affectation de l’entrée

Paramètre effectif	Explication
Expression	Une expression correspond à une valeur et comporte des opérandes et des opérateurs. Voici les types d’expressions disponibles : expression arithmétique, logique ou de comparaison constante variable étendue
Désignation TEMPORISATION COMPTEUR	Vous définissez une temporisation ou un compteur qui seront utilisés à l’exécution du bloc (voir aussi le chapitre 17).
Désignation BLOC	Vous définissez un bloc qui sera utilisé comme paramètre d’entrée. Le type de bloc (FB, FC, DB) doit être défini dans la déclaration du paramètre d’entrée. A l’affectation du paramètre, vous indiquez la désignation du bloc, absolue ou symbolique (voir aussi le chapitre 9).

Particularités

Veuillez noter que le paramètre d’entrée/sortie d’un FB et le paramètre d’une FC ne sont pas permis comme paramètres effectifs pour les paramètres d’entrée formels d’une FC dont le type de données est autre que simple.

16.3.3 Affectation de la sortie ou de l’entrée/sortie (FC)

Principe

L’affectation de la sortie vous permet de définir où vont être inscrites les valeurs de sortie obtenues après l’exécution d’une fonction. L’affectation d’une entrée/sortie vous permet d’affecter une valeur effective à un paramètre d’entrée/sortie.

La figure 16-13 montre la syntaxe de l’affectation de la sortie ou de l’entrée/sortie :

Affectation de la sortie ou de l’entrée/sortie

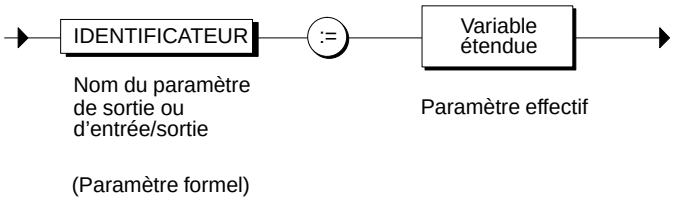


Figure 16-13 *Syntaxe : affectation de la sortie ou de l’entrée/sortie*

Paramètres effectifs dans l’affectation d’une sortie ou d’une entrée/sortie

Dans l’affectation de la sortie ou de l’entrée/sortie, les paramètres effectifs doivent être des variables, puisque la FC doit inscrire des valeurs dans les paramètres. C’est la raison pour laquelle, vous ne pouvez pas affecter un paramètre d’entrée à une entrée/sortie (la nouvelle valeur ne pourrait pas y être inscrite). Seule une variable étendue peut être utilisée pour l’affectation d’une sortie ou d’une entrée/sortie :

Tableau 16-5 Paramètres effectifs dans les affectation de sortie ou d’entrée/sortie

Paramètre effectif	Explication
Variable étendue	Vous disposez des types suivants de variables étendues : variable simple et paramètre adressage de variable absolue adressage de bloc de données appel de fonction (voir aussi le chapitre 14)

Particularités

Tenez compte des particularités suivantes :

- La valeur modifiée du paramètre d'entrée/sortie est actualisée après l'exécution du bloc.
- Les paramètres effectifs suivants ne peuvent pas être affectés aux paramètres d'entrée/sortie d'un type de données **autre que simple** :
  - paramètre d'entrée d'un FB,
  - paramètre d'entrée/sortie d'un FB, ou
  - paramètre d'une FC
- **Paramètre ANY :** par principe, le premier point précité s'y applique également. Pour les paramètres d'entrée/sortie d'un type de données **non simple**, ne sont pas autorisés comme paramètres effectifs :
  - les paramètres d'entrée de FB,
  - les paramètres d'entrée de FC.

  Outre les restrictions ci-dessus, les constantes ne sont **pas** autorisées comme paramètres effectifs. Si vous avez déclaré comme résultat de la fonction (valeur en retour) le type ANY, sachez en outre que :
  - Les paramètres effectifs que vous affectez aux paramètres ANY doivent être du même type que ces derniers. L'ensemble de types de données numériques (INT, DINT, REAL) ou l'ensemble de types de données binaires (BOOL, BYTE, WORD, DWORD) constituent par exemple une classe. Les autres types de données ont une classe qui leur est propre.
  - Le compilateur SCL part du principe que le type de données du résultat actuel de la fonction est le type qui l'a emporté parmi les paramètres effectfs affectés aux paramètres ANY.
    Vous pouvez combiner le résultat de la fonction avec toutes les opérations définies pour ce type de données.
- **Paramètre POINTER :** par principe, le premier point précité s'y applique également. Pour les paramètres d'entrée/sortie d'un type de données **non simple**, ne sont pas autorisés comme paramètres effectifs :
  - les paramètres d'entrée de FB,
  - les paramètres d'entrée de FC.

16.3.4 Exemple d'appel de fonction

Principe

Soit la fonction ECART permettant de calculer l'écart entre deux points (X1,Y1) et (X2,Y2) dans un plan à l'aide de leur coordonnées cartésiennes (il est supposé que le mnémonique ECART a été déclaré pour la FC37 dans la table des mnémoniques) :

```
FUNCTION ECART: REAL
      VAR_INPUT
                  X1: REAL;
                  X2: REAL;
                  Y1: REAL;
                  Y2: REAL;
      END_VAR
      VAR_OUTPUT
                  Q2: REAL;
      END_VAR
      BEGIN
                  ECART:= SQRT
                  ( (X2-X1)**2 + (Y2-Y1)**2 );
                  Q2:= X1+X2+Y1+Y2;
END_FUNCTION
```

Exemple 16-6 Calcul de l'écart

Diverses possibilités s'offrent à vous pour exploiter la valeur de la fonction :

dans une affectation de valeur, par exemple :

```
LONGUEUR:= ECART (X1:=-3, Y1:=2, X2:=8.9, Y2:=7.4,
Q2:=Somme_chiffres);
```

dans une expression arithmétique ou logique, par exemple :

```
RAYON + ECART (X1:=-3, Y1:=2, X2:=8.9, Y2:=7.4,
Q2:=Somme_chiffres)
```

dans l'affectation de paramètres à un bloc appelé, par exemple :

```
FB32 (DISTANCE:= ECART (X1:=-3, Y1:=2, X2:=8.9,
Y2:=7.4, Q2:=Somme_chiffres);
```

Exemple 16-7 Calcul d'une valeur dans une FC

16.4 Paramètres définis automatiquement

Présentation

Les paramètres définis automatiquement sont ceux que vous pouvez utiliser sans devoir préalablement les déclarer dans un bloc. SCL en met deux à votre disposition :

- le paramètre d'entrée `EN`,
- le paramètre de sortie `ENO`.

Tous deux sont du type de données `BOOL` et se trouvent dans la zone des données temporaires du bloc.

Paramètre d'entrée EN

Tout bloc fonctionnel et toute fonction possèdent le paramètre d'entrée EN, défini automatiquement. Si EN est vrai, le bloc appelé est exécuté, sinon il ne l'est pas. L'affectation d'une valeur au paramètre EN est facultative.

Sachez que EN ne doit pas être déclaré dans la section de déclaration d'un bloc ou d'une fonction.

Puisque `EN` est un paramètre d'entrée, vous ne pouvez pas le modifier dans un bloc.

Nota

Si la fonction n'est pas appelée car `EN:=FALSE`,sa valeur en retour n'est pas définie.

Exemple

Voici un exemple d'utilisation du paramètre EN :

```
FUNCTION_BLOCK FB57
VAR
      RESULTAT    : REAL;
      MA_VALID    : BOOL;
END_VAR
//...
BEGIN
MA_VALID:= FALSE;
// Appel d'une fonction,
// avec affectation de valeur au paramètre EN :

RESULTAT:= FC85 (EN:= MA_VALID, PAR_1:= 27);
// La FC85 n'a pas été exécutée, car la valeur FALSE
// a été affectée à MA_VALID
//...
END_FUNCTION_BLOCK
```

Exemple 16-8 Utilisation de EN

Paramètre de sortie ENO

Tout bloc fonctionnel et toute fonction possèdent le paramètre de sortie ENO de type BOOL, défini automatiquement. A la fin de l'exécution d'un bloc, la valeur actuelle de la variable OK est inscrite dans le paramètre ENO.

Immédiatement après avoir appelé un bloc, vous pouvez vérifier, à l'aide de la valeur du paramètre ENO, si toutes les opérations dans le bloc ont été exécutées correctement ou si des erreurs sont survenues.

Exemple

Voici un exemple d'utilisation du paramètre EN0.

```
FUNCTION_BLOCK FB57
      //...
      //...
BEGIN
// Appel d'un bloc fonctionnel :
FB30.DB30 (X1:=10, X2:=10.5);

// Vérification du déroulement correct
// dans le bloc appelé :

IF ENO THEN
      // tout c'est correctement déroulé
      //...
ELSE
      // une erreur est survenue,
      // et doit être corrigée
      //...
END_IF;
      //...
      //...
END_FUNCTION_BLOCK
```

Exemple 16-9 Utilisation de ENO

Exemple

Voici une combinaison de EN et EN0

```
// EN et ENO peuvent aussi être combinés

FB30.DB30(X1:=10, X2:=10.5);

// La fonction suivante ne doit s'exécuter
// que si le FB30 l'a été
// sans erreur :

RESULTAT:= FC85 (EN:= ENO, PAR_1:= 27);
```

Exemple 16-10 Combinaison de EN et ENO

Compteurs et temporisations

17

Présentation

Vous avez la possibilité de soumettre l'exécution de votre programme à des événements de temporisation ou à des valeurs de comptage.

STEP 7 vous propose à cet effet diverses fonctions de comptage et de temporisation que vous pouvez utiliser dans votre programme SCL sans devoir préalablement les déclarer.

Structure du chapitre

17.1 Fonctions de comptage

Présentation

STEP 7 met à votre disposition une série de fonctions de comptage standard que vous pouvez utiliser dans votre programme SCL sans devoir préalablement les déclarer. Il vous suffit de leur affecter les paramètres requis. Voici les fonctions de comptage que vous propose STEP 7 :

- compteur incrémental (Counter Up)
- compteur décrémental (Counter Down)
- compteur incrémental et décrémental (Counter Up Down)

Appel

Vous appelez une fonction de comptage de la même manière que vous appelez une fonction. La désignation d'une fonction peut remplacer tout opérande dans une expression, si le type du résultat de la fonction est compatible avec celui de l'opérande remplacé.

Tableau 17-1 Nom des fonctions de comptage

Nom de la fonction	Signification
S_CU	Compteur incrémental (Counter Up)
S_CD	Compteur décrémental (Counter Down)
S_CUD	Compteur incrémental et décrémental (Counter Up Down)

Valeur de la fonction

La valeur de la fonction (valeur en retour) fournie là où elle est appelée correspond à la valeur de comptage actuelle (`format DCB`) dans le type de données `WORD`. Vous trouverez de plus amples informations à ce sujet au paragraphe 17.1.1.

Paramètres d'appel

La désignation et la description des paramètres d'appel des trois fonctions de comptage figurent dans le tableau 17-2. Il faut généralement distinguer les types de paramètres suivants :

- paramètres de commande (p. ex. mettre à « 1 », mettre à « 0 », indication du sens de comptage)
- valeur de présélection du compteur
- sortie de l'état (indiquant la fin du comptage)
- état du compteur sous forme binaire

Tableau 17-2 Paramètres d'appel

Désignation	Paramètre	Type de données	Description
C_NO		COUNTER	Numéro du compteur (DESIGNATION DU COMPTEUR) ; la plage dépend de la CPU.
CU	Entrée	BOOL	Entrée CU : incrémenter
CD	Entrée	BOOL	Entrée CD : décrémenter
S	Entrée	BOOL	Entrée de mise à « 1 » du compteur
PV	Entrée	WORD	Valeur pour présélection du compteur comprise entre 0 et 999 (saisie sous forme 16#<valeur>, la valeur étant une valeur DCB)
R	Entrée	BOOL	Entrée de remise à « 0 » du compteur
Q	Sortie	BOOL	Etat du compteur
CV	Sortie	WORD	Etat du compteur (binaire)

Exemple

Dans l'exemple 17-1, une zone de mémoire globale de type COUNTER appelée Z12 est réservée lors du calcul de la fonction.

```
Valeur_comptage:= S_CUD (C_NO      :=Z12,
                         CD    :=E.0,
                         CU    :=E.1,
                         S     :=E.2 & E.3,
                         PV    :=120,
                         R     :=FALSE,
                         CV    :=binVal,
                         Q     :=actFlag);
```

Exemple 17-1 Appel d'une fonction de décrémentation

Appel dynamique

Au lieu d'appeler le compteur par un numéro absolu (par exemple `C_NO=Z10`), vous pouvez également indiquer une variable de type de données COUNTER. L'avantage réside dans le fait que vous pouvez réaliser l'appel de la fonction de comptage de manière dynamique, en affectant à chaque appel un numéro absolu différent à sa variable.

```
Exemple :
FUNCTION_BLOCK COMPTEUR;
Var_Input
 MonCompteur: Counter;
End_Var
:
currVAL:=S_CD (C_NO:=MonCompteur,........);
```

Règles

Puisque les valeurs des paramètres (par exemple CD:=E.0) sont enregistrées globalement, leur indication est dans certains cas facultative. Vous devez tenir compte des règles suivantes pour l'affectation des paramètres :

- Lorsque vous appelez le compteur, vous devez toujours affecter une valeur à son paramètre de désignation C_NO.
- Selon la fonction de comptage choisie, vous devez affecter une valeur au paramètre CU (compteur incrémental) ou au paramètre CD (compteur décrémental).
- L'indication de la paire de paramètres PV (valeur de présélection) et S (mettre à « 1 ») est facultative.
- Le résultat dans le format DCB correspond toujours à la valeur de la fonction.

Nota

Les abréviations SIMATIC et CEI des noms des fonctions et des paramètres sont les mêmes, à l'exception de la désignation du compteur : *SIMATIC :* Z et *CEI :* C.

Exemple d'appel de fonctions de comptage

L'exemple 17-2 montre l'appel des fonctions de comptage :

```
Function_block FB1


VAR
 currVal, binVal: word;
 actFlag: bool;
END_VAR
```

```
BEGIN
currVal           :=S_CD(C_NO:=Z10, CD:=TRUE, S:=TRUE,
                     PV:=100, R:=FALSE, CV:=binVal,
                     Q:=actFlag);
```

```
currVal           :=S_CU(C_NO:=Z11, CU:=M0.0, S:=M0,1,
                     PV:=16#110, R:=M0.2, CV:=binVal,
                     Q:=actFlag);
```

```
currVal           :=S_CUD(C_NO:=Z12, CD:=E.0,
                     CU:=E.1,S:=E.2 & E.3, PV:=120,
                     R:=FALSE,CV:=binVal, Q:=actFlag);
```

```
currVal           :=S_CD(C_NO:=Z10,CD:=FALSE,
                     S:=FALSE,
                     PV:=100, R:=TRUE, CV:=bVal,
                     Q:=actFlag);


end_function_block
```

Exemple 17-2 Appel des fonctions de comptage

17.1.1 Saisie et exploitation de la valeur de comptage

Présentation

Pour la saisie de la valeur de présélection ou l'exploitation du résultat de la fonction, vous devez utiliser la représentation interne de la valeur de comptage (voir figure 17-1).

Lorsque vous mettez le compteur à « 1 » (paramètre S), la valeur que vous avez présélectionnée s'y inscrit. La plage des valeurs autorisées est comprise entre 0 et 999. Vous pouvez modifier la valeur de comptage dans cette plage en utilisant les fonctions d'incrémentation/décrémentation, d'incrémentation ou de décrémentation.

Format

La figure 17-1 donne la représentation de la valeur de comptage en bits :

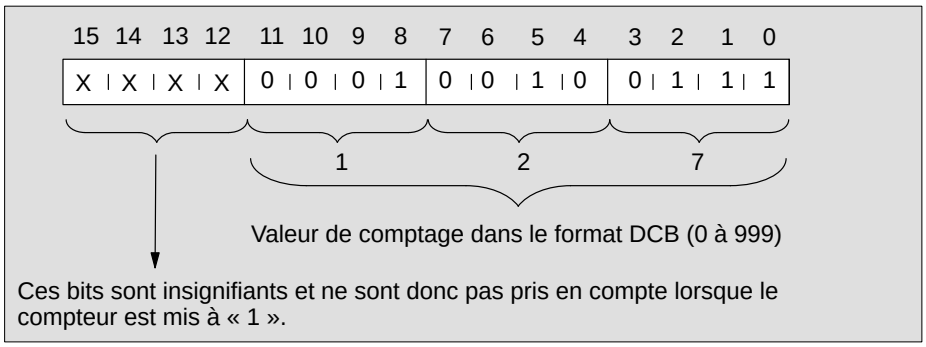


Figure 17-1 *Représentation de la valeur de comptage en bits*

Saisie

Pour charger une valeur de comptage prédéfinie, vous pouvez utiliser les formats suivants :

- décimal sous forme de valeur entière : p. ex. 295, si cette valeur correspond à un codage DCB autorisé.
- format DCB (saisie sous forme de constante hexadécimale) : p. ex. 16#127

Exploitation

Pour exploiter le résultat, vous pouvez utiliser les deux formats suivants :

- comme résultat de la fonction (type WORD) : format DCB
- comme paramètre de sortie CV (type WORD) : binaire

17.1.2 Incrémenter (Counter Up)

Description

Le compteur incrémental vous permet uniquement d'effectuer des opérations d'incrémentation.

Fonctionnement

Tableau 17-3 Fonctionnement du compteur incrémental

Opération	Fonctionnement
Incrémenter	La valeur du compteur est augmentée de « 1 », lorsque l'état de signal à l'entrée **CU** passe de « 0 » à « 1 » et si la valeur de comptage est inférieure à 999.
Mettre le compteur à « 1 »	Lorsque l'état de signal à l'entrée **S** passe de « 0 » à « 1 », le compteur prend la valeur de l'entrée **PV**. Un tel changement de signal est toujours requis pour mettre un compteur à « 1 ».
Mettre le compteur à « 0 »	Le compteur est mis à « 0 » lorsque l'entrée **R** = 1. A la mise à « 0 » du compteur, la valeur de comptage prend la valeur « 0 ».
Interroger le compteur	L'interrogation de l'état de signal à la sortie **Q** fournit la valeur « 1 », si la valeur de comptage est supérieure à « 0 ». Il fournit la valeur « 0 », si la valeur de comptage est égale à « 0 ».

17.1.3 Décrémenter (Counter Down)

Description

Le compteur décrémental vous permet uniquement d'effectuer des opérations de décrémentation.

Fonctionnement

Tableau 17-4 Fonctionnement du compteur décrémental

Opération	Fonctionnement
Décrémenter	La valeur du compteur est diminuée de « 1 », lorsque l'état de signal à l'entrée **CD** passe de « 0 » à « 1 » (front montant) et si la valeur de comptage est supérieure à « 0 ».
Mettre le compteur à « 1 »	Lorsque l'état de signal à l'entrée **S** passe de « 0 » à « 1 », le compteur prend la valeur de l'entrée **PV**. Un tel changement de signal est toujours requis pour mettre un compteur à « 1 ».
Mettre le compteur à « 0 »	Le compteur est mis à « 0 » lorsque l'entrée **R** = 1. A la mise à « 0 » du compteur, la valeur de comptage prend la valeur « 0 ».
Interroger le compteur	L'interrogation de l'état de signal à la sortie **Q** fournit la valeur « 1 », si la valeur de comptage est supérieure à « 0 ». Il fournit la valeur « 0 », si la valeur de comptage est égale à « 0 ».

17.1.4 Incrémenter / décrémenter (Counter Up Down)

Description

Le compteur incrémental et décrémental vous permet d'exécuter aussi bien des opérations d'incrémentation que de décrémentation. Lorsque deux impulsions sont simultanées, les deux opérations sont traitées. La valeur de comptage reste inchangée.

Fonctionnement

Tableau 17-5 Fonctionnement du compteur incrémental/décrémental

Opération	Fonctionnement
Incrémenter	La valeur du compteur est augmentée de « 1 », lorsque l'état de signal à l'entrée **CU** passe de « 0 » à « 1 » et si la valeur de compatge est inférieure à 999.
Décrémenter	La valeur du compteur est diminuée de « 1 », lorsque l'état de signal à l'entrée **CD** passe de « 0 » à « 1 » et si la valeur de comptage est supérieure à « 0 ».
Mettre le compteur à « 1 »	Lorsque l'état de signal à l'entrée **S** passe de « 0 » à « 1 », le compteur prend la valeur de l'entrée **PV**. Un tel changement de signal est toujours requis pour mettre un compteur à « 1 ».
Mettre le compteur à « 0 »	Le compteur est mis à « 0 » lorsque l'entrée **R** = 1. A la mise à « 0 » du compteur, la valeur de comptage prend la valeur « 0 ».
Interroger le compteur	L'interrogation de l'état de signal à la sortie **Q** fournit la valeur « 1 », si la valeur de comptage est supérieure à « 0 ». Il fournit la valeur « 0 », si la valeur de comptage est égale à « 0 ».

17.1.5 Exemple de fonction S_CD (décrémenter)

Affectation des paramètres

Le tableau 17-6 montre un exemple d'affectation de paramètres à la fonction S_CD.

Tableau 17-6 Paramètres d'appel

Paramètre	Description
C_NO	`MonCompteur`
CD	`Entrée E0.0`
S	`METTRE_1`
PV	`Présélection 16#0089`
R	`METTRE_0`
Q	`A0.7`
CV	`VAL_BIN`

Exemple

L’exemple 17-3 représente la fonction de comptage S_CD :

```
FUNCTION_BLOCK COMPTER
VAR_INPUT
 MONCOMPTEUR: COUNTER;
END_VAR
VAR_OUTPUT
 RESULTAT: INT;
END_VAR
VAR
 METTRE_1    : BOOL;
 METTRE_0    : BOOL;
 VAL_DCB     : WORD; //Etat du compteur, format DCB
 VAL_BIN     : WORD; //Etat du compteur, format binaire
 PRESELECTION: WORD;
END_VAR
BEGIN
 A0.0:= 1;
 METTRE_1:= E0.2;
 METTRE_0:= E0.3;
 PRESELECTION:= 16#0089;
 VAL_DCB:=   S_CD
             (C_NO   := MONCOMPTEUR,//décrémenter
             CD      := E.0,
             S       := METTRE_1,
             PV      := PRESELECTION,
             R       := METTRE_0,
             CV      := VAL_BIN,
             Q       := A0.7);
RESULTAT:=WORD_TO_INT(VAL_BIN);//EXPLOITATION RESULTAT
                      //comme paramètre de sortie
AW4:= VAL_DCB;        //Vers la sortie pour affichage
END_FUNCTION_BLOCK
```

Exemple 17-3 Exemple de fonction de comptage

17.2 Fonctions de temporisation (TIMER)

Présentation

Les temporisations sont des éléments fonctionnels de votre programme vous permettant d'exécuter et de contrôler des séquences déclenchées par horloge. STEP 7 met à votre disposition une série de fonctions de temporisation standard que vous pouvez utiliser dans votre programme SCL pour :

- régler des temps d'attente
- permettre des temps de surveillance
- créer des impulsions
- chronométrer des durées

Appel

Vous appelez une fonction de temporisation de la même manière qu'une fonction de comptage. La désignation d'une fonction peut remplacer tout opérande dans une expression, si le type du résultat de la fonction est compatible avec celui de l'opérande remplacé.

Tableau 17-7 Fonctions de temporisation de STEP 7

Nom de la fonction	Signification
S_PULSE	Impulsion (Pulse)
S_PEXT	Impulsion prolongée (Pulse Extended)
S_ODT	Retard à la montée (On Delay Time)
S_ODTS	Retard à la montée mémorisé (Stored On Delay Time)
S_OFFDT	Retard à la retombée (Off Delay Time)

Valeur de la fonction

La valeur de la fonction (valeur en retour) fournie là où elle est appelée correspond à la valeur de temps de type de données `S5TIME`. Vous trouverez de plus amples informations à ce sujet au paragraphe 17.2.1.

Paramètres d'appel

Les paramètres auxquels vous devez affecter une valeur sont donnés sous forme de tableau dans la description de chaque fonction de temporisation. Le tableau 17-8 indique les noms des cinq fonctions de temporisation avec les types de données correspondants.

Il faut généralement distinguer les types de paramètres suivants :

- paramètre de commande (par exemple mettre à « 1 », mettre à « 0 »),
- valeur de présélection pour le temps initial,
- état de la sortie, indique si la temporisation fonctionne encore,
- valeur de temps restante,

Tableau 17-8 Paramètres d'appel

Paramètre	Type de données	Description
T_NO	TIMER	Numéro d'identification de la temporisation ; la plage dépend de la CPU.
S	BOOL	Entrée de démarrage
TV	S5TIME	Présélection de la valeur de temps (format DCB)
R	BOOL	Entrée de remise à zéro
Q	BOOL	Etat de la temporisation
BI	WORD	Valeur de temps restante (binaire)

Exemple

Dans l'exemple 17-4, une zone de mémoire globale de type TIMER appelée T10 est réservée lors de l'exécution de la temporisation.

```
RETARD:= S_ODT (T_NO       := T10,
                S          := TRUE,
                TV         := T#1s,
                R          := FALSE,
                BI         := biVal,
                Q          := actFlag
                           );
```

Exemple 17-4 Appel de la fonction de temporisation S_ODT

Appel dynamique

Au lieu d'appeler la temporisation par un numéro absolu (par exemple `T10`), vous pouvez également indiquer une variable de type de données TIMER. L'avantage réside dans le fait que vous pouvez réaliser l'appel de la fonction de temporisation de manière dynamique, en affectant à chaque appel un numéro absolu différent à sa variable.

```
Exemple :
FUNCTION_BLOCK TEMPORISATION
VAR_INPUT
MaTempo: Timer;
END_VAR
:
currTime:=S_ODT (T_NO:=MaTempo,.........)
```

Règles

Puisque les valeurs des paramètres sont enregistrées globalement, leur indication est dans certains cas facultative. Vous devez tenir compte des règles suivantes pour l'affectation des paramètres :

- Lorsque vous appelez la temporisation, vous devez toujours affecter une valeur abslolue ou symbolique à son paramètre de désignation T_NO.
- L'indication de la paire de paramètres TV (valeur de présélection) et S (mettre à « 1 ») est facultative.
- L'affectation de la sortie est facultative. Vous pouvez adresser Q et BI par une affectation de valeur.
- Le résultat dans le format S5TIME correspond toujours à la valeur de la fonction.

Nota

Les abréviations SIMATIC et CEI des noms des fonctions sont les mêmes.

Exemple d'appel d'une fonction de temporisation

L'exemple 17-5 montre l'appel des fonctions de temporisation :

```
FUNCTION_BLOCK FB2

VAR
currTime: S5time;
biVal: word;
actFlag: bool;
END VAR

BEGIN
currTime:= S_ODT (T_NO:=T10, S:=TRUE, TV:=T#1s,
                  R:=FALSE, BI:=biVal,
                  Q:=actFlag);
```

```
currTime:= S_ODTS (T_NO:=T11, S:=M0,0, TV:=T#1s,
                   R:= M0.1, BI:=biVal,
                   Q:= actFlag);
```

```
currTime:=S_OFFDT (T_NO:=T12, S:=I0.1&actFlag,
                   TV:= T#1s,R:=FALSE,BI:=biVal,
                   Q:= actFlag);
```

```
currTime:= S_PEXT (T_NO:=T13, S:=TRUE,
                   TV:=T#1s,R:=I0.0, BI:=biVal,
                   Q:=actFlag);
```

```
currTime:= S_PULSE (T_NO:=T14, S:=TRUE,
                    TV:=T#1s,R:=FALSE, BI:=biVal,
                     Q:=actFlag);
END_FUNCTION_BLOCK
```

Exemple 17-5 Appel des fonctions de temporisation

17.2.1 Saisie et exploitation de la valeur de temps

Présentation

Pour la saisie de la valeur de présélection ou l'exploitation du résultat de la fonction en code DCB, vous devez utiliser la représentation interne de la valeur de temps (voir figure 17-2).

Lors de la mise à jour de la temporisation, la valeur de temps est diminuée d'une unité par intervalle, celui-ci ayant été déterminé à partir de la base de temps. La valeur de temps est diminuée jusqu'à ce qu'elle soit égale à « 0 ». La plage des valeurs est comprise entre 0 et 9990 secondes.

Format

La figure 17-2 donne la représentation interne de la valeur de temps.

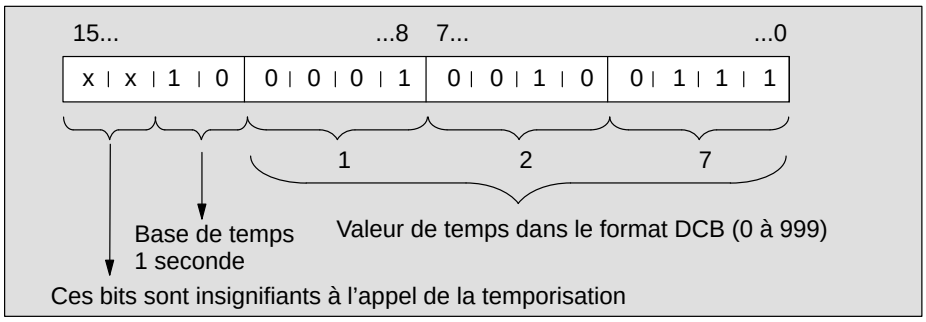


Figure 17-2 *Représentation de la valeur de temps*

Saisie

Pour charger une valeur de temps prédéfinie, vous pouvez utiliser les représentations suivantes :

- représentation par niveaux : `TIME#aH_bbM_ccS_dddMS`
- représentation décimale : `TIME#2.4H`

Exploitation

Pour exploiter le résultat, vous pouvez utiliser l'un des deux formats suivants :

- comme résultat de la fonction (type S5TIME) : format DCB,
- comme paramètre de sortie (valeur de temps sans base de temps du type `WORD`) : format binaire.

Base de temps

Les bits 12 et 13 du mot de temporisation contiennent la base de temps dans le format binaire. C'est elle qui définit l'intervalle dans lequel la valeur de temps est diminuée d'une unité (voir tableau 17-9 et figure 17-2). La base de temps la plus petite est égale à 10 ms, la plus grande à 10 s.

Tableau 17-9 Base de temsp avec le code binaire correspondant

Base de temps	Code binaire
10 ms	00
100 ms	01
1 s	10
10 s	11

Nota

Les valeurs n'étant enregistrées que durant un intervalle de temps donné, celles qui n'en sont pas un multiple exact sont tronquées.

Les valeurs dont la résolution est trop grande pour la plage souhaitée sont arrondies, si bien que l'objectif est la plage voulue et non **pas** la résolution désirée.

17.2.2 Démarrer une temporisation sous forme d'impulsion

Description

Le temps maximal durant lequel le signal de sortie reste à « 1 » est égal à la valeur de temps t programmée.

Si l'état de signal « 0 » apparaît à l'entrée pendant la durée d'exécution de la temporisation, la sortie est mise à « 0 » (le temps est arrêté). La durée d'exécution est donc interrompue avant la fin.

La figure 17-3 représente le fonctionnement de la temporisation « Démarrer une temporisation sous forme d'impulsion » :

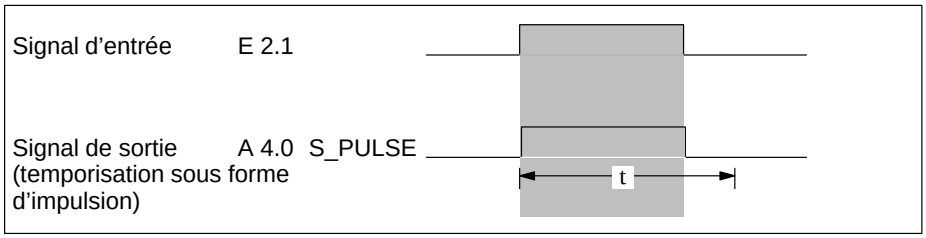


Figure 17-3 *Temporisation « Démarrer une temporisation sous forme d'impulsion »*

Fonctionnement

Tableau 17-10 Fonctionnement de l'opération « Démarrer une temporisation sous forme d'impulsion »

Opération	Fonctionnement
Démarrer la temporisation	L'opération « Démarrer une temporisation sous forme d'impulsion » démarre la temporisation indiquée, lorsque l'état de signal à l'entrée de démarrage **S** passe de « 0 » à « 1 ». Un changement d'état de signal est toujours requis pour valider la temporisation.
Présélectionner la durée d'exécution	La temporisation continue de s'exécuter avec la valeur indiquée à l'entrée **TV** jusqu'à ce que la durée programmée se soit écoulée et tant que l'entrée S = 1.
Arrêter la temporisation avant la fin	Si le signal à l'entrée **S** passe de « 1 » à « 0 » avant que la valeur de temps ne se soit écoulée, la temporisation est arrêtée.
Initialiser	La temporisation est initialisée lorsque le signal à l'entrée de remise à zéro **R** passe de « 0 » à « 1 », pendant que la temporisation s'exécute. La valeur de temps et la base de temps sont alors également remises à zéro. L'état de signal « 1 » à l'entrée R n'a aucun effet lorsque la temporisation ne s'exécute pas.
Interroger l'état de signal	Tant que la temporisation s'exécute, l'interrogation à « 1 » de l'état de signal à la sortie **Q** fournit le résultat « 1 ». Si la durée d'exécution a été arrêtée avant la fin, l'interrogation de l'état de signal à la sortie Q fournit le résultat « 0 ».
Interroger la valeur de temps actuelle	Vous pouvez interroger la valeur de temps actuelle à la sortie **BI** à l'aide de la valeur de la fonction S_PULSE.

17.2.3 Démarrer une temporisation sous forme d'impulsion prolongée

Description

Durant le temps (t) programmé, le signal de sortie reste à « 1 », quelle que soit la durée pendant laquelle le signal d'entrée reste à « 1 ». Lorsque l'impulsion de démarrage est à nouveau initiée, la durée s'écoule une nouvelle fois, se sorte que l'impulsion de sortie est prolongée dans la temps (redémarrage).

La figure 17-4 représente le fonctionnement de la temporisation « Démarrer une temporisation sous forme d'impulsion prolongée ».

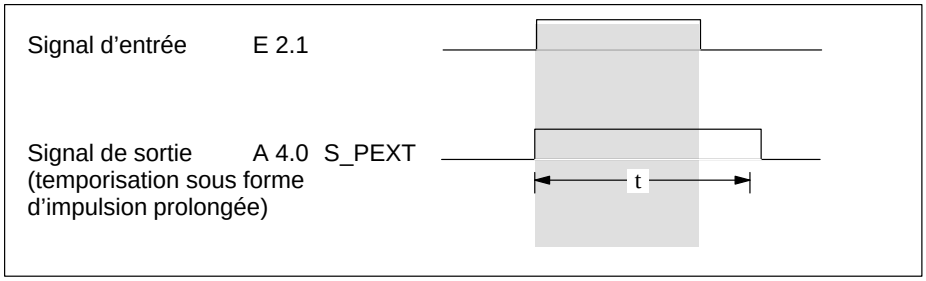


Figure 17-4 *Temporisation « Démarrer une temporisation sous forme d'impulsion prolongée »*

Tableau 17-11 Fonctionnement de l'opération « Démarrer une temporisation sous forme d'impulsion prolongée »

Fonctionnement

Opération	Fonctionnement
Démarrer la temporisation	L'opération « Démarrer une temporisation sous forme d'impulsion prolongée » démarre la temporisation indiquée, lorsque l'état de signal à l'entrée de démarrage (**S**) passe de « 0 » à « 1 ». Un changement d'état de signal est toujours requis pour valider la temporisation.
Redémarrer la durée d'exécution	Si l'état de signal à l'entrée **S** passe une nouvelle fois à « 1 » pendant la durée d'exécution, la temporisation est redémarrée avec la valeur de temps indiquée.
Présélectionner la durée d'exécution	La temporisation s'exécute avec la valeur indiquée à l'entrée **TV** jusqu'à ce que la durée programmée se soit écoulée.
Initialiser	La temporisation est initialisée lorsque le signal à l'entrée de remise à zéro (**R**) passe de « 0 » à « 1 », pendant que la temporisation s'exécute. La valeur de temps et la base de temps sont alors également remises à zéro. L'état de signal « 1 » à l'entrée R n'a aucun effet lorsque la temporisation ne s'exécute pas.
Interroger l'état de signal	Tant que la temporisation s'exécute, l'interrogation à « 1 » de l'état de signal à la sortie **Q** fournit le résultat « 1 », quelle que soit la durée du signal d'entrée.
Interroger la valeur de temps actuelle	Vous pouvez interroger la valeur de temps actuelle à la sortie **BI** à l'aide de la valeur de la fonction S_PEXT.

17.2.4 Démarrer une temporisation sous forme de retard à la montée

Description

Le signal de sortie ne passe de « 0 » à « 1 » que lorsque la temporisation programmée s'est écoulée et que le signal d'entrée est toujours à « 1 ». Ceci signifie que la sortie est activée avec un retard. Les signaux d'entrée dont la durée est plus courte que celle de la temporisation programmée n'apparaissent pas à la sortie.

La figure 17-5 représente le fonctionnement de la temporisation « Démarrer une temporisation sous forme de retard à la montée ».

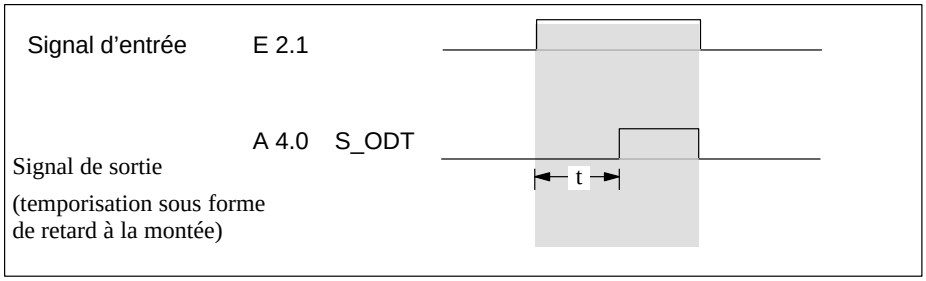


Figure 17-5 *Temporisation « Démarrer une temporisation sous forme de retard à la montée »*

Tableau 17-12 Fonctionnement de l'opération « Démarrer une temporisation sous forme de retard à la montée »

Fonctionnement

Opération	Fonctionnement
Démarrer la temporisation	L'opération « Démarrer une temporisation sous forme de retard à la montée » démarre la temporisation indiquée, lorsque l'état de signal à l'entrée de démarrage **S** passe de « 0 » à « 1 ». Un changement d'état de signal est toujours requis pour valider la temporisation.
Arrêter la temporisation	Si le signal à l'entrée **S** passe de « 1 » à « 0 » pendant que la temporisation s'exécute, celle-ci est arrêtée.
Présélectionner la durée d'exécution	La temporisation continue de s'exécuter avec la valeur indiquée à l'entrée **TV** tant que l'état de signal à l'entrée S = 1.
Initialiser	La temporisation est initialisée lorsque le signal à l'entrée de remise à zéro **R** passe de « 0 » à « 1 », pendant que la temporisation s'exécute. La valeur de temps et la base de temps sont alors également remises à zéro. La temporisation est également remise à zéro lorsque R=1 quand elle ne s'exécute pas.
Interroger l'état de signal	Une interrogation à « 1 » de l'état de signal à la sortie **Q** fournit le résultat « 1 », lorsque la temporisation s'est écoulée correctement et que l'entrée S est toujours à « 1 ». Si la temporisation a été arrêtée, l'interrogation à « 1 » de l'état de signal donne toujours « 0 ». Une interrogation à « 1 » de l'état de signal à la sortie **Q** fournit également le résultat « 0 », lorsque la temporisation ne s'exécute pas et que l'état de signal à l'entrée S est toujours égal à « 1 ».
Interroger la valeur de temps actuelle	Vous pouvez interroger la valeur de temps actuelle à la sortie **BI** à l'aide de la valeur de la fonction S_ODT.

17.2.5 Démarrer une temporisation sous forme de retard à la montée mémorisé

Description

Le signal de sortie ne passe de « 0 » à « 1 » que lorsque la temporisation programmée s'est écoulée, quelle que soit la durée pendant laquelle le signal d'entrée reste à « 1 ».

La figure 17-6 représente le fonctionnement de la temporisation « Démarrer une temporisation sous forme de retard à la montée mémorisé ».

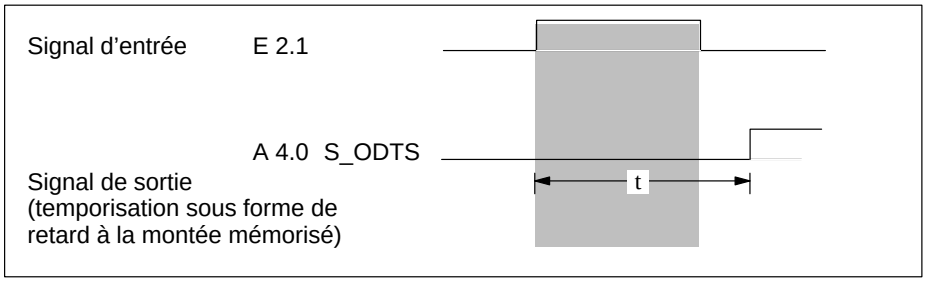


Figure 17-6 *Temporisation « Démarrer une temporiastion sous forme de retard à la montée mémorisé »*

Fonctionnement

Tableau 17-13 Fonctionnement de l'opération « Démarrer une temporisation sous forme de retard à la montée mémorisé »

Opération	Fonctionnement
Démarrer la temporisation	L'opération « Démarrer une temporisation sous forme de retard à la montée mémorisé » démarre la temporisation indiquée, lorsque l'état de signal à l'entrée de démarrage **S** passe de « 0 » à « 1 ». Un changement d'état de signal est toujours requis pour valider la temporisation.
Redémarrer la temporisation	La temporisation est redémarrée avec la valeur indiquée lorsque l'entrée **S** passe de « 0 » à « 1 » pendant que la temporisation s'exécute.
Présélectionner la durée d'exécution	La temporisation continue de s'exécuter avec la valeur indiquée à l'entrée **TV**, même lorsque l'état de signal à l'entrée S passe à « 0 » avant que la temporisation ne se soit écoulée.
Initialiser	La temporisation est initialisée lorsque le signal à l'entrée de remise à zéro **R** passe de « 0 » à « 1 », quelle que soit la valeur du RLG (résultat logique, voir **/232/**) à l'entrée S.
Interroger l'état de signal	Une interrogation à « 1 » de l'état de signal à la sortie **Q** fournit le résultat « 1 », lorsque la temporisation s'est écoulée, quel que soit l'état de signal à l'entrée S.
Interroger la valeur de temps actuelle	Vous pouvez interroger la valeur de temps actuelle à la sortie **BI** à l'aide de la valeur de la fonction S_ODTS.

17.2.6 Démarrer une temporisation sous forme de retard à la retombée

Description

Lorsque l'état de signal à l'entrée S passe de « 0 » à « 1 », la sortie Q prend l'état de signal « 1 ». Lorsque l'état de signal à l'entrée de démarrage passe de « 1 » à « 0 », la temporisation est démarrée. Ce n'est qu'une fois la durée écoulée que la sortie prend l'état de signal « 0 ». La sortie est donc désactivée avec un retard.

La figure 17-7 représente le fonctionnement de la temporisation « Démarrer une temporisation sous forme de retard à la retombée ».

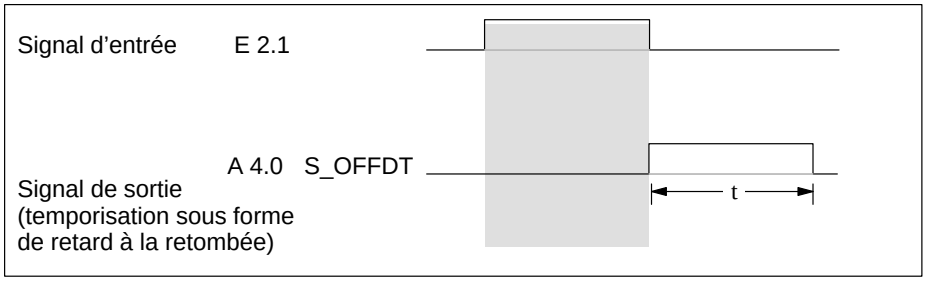


Figure 17-7 *Temporisation « Démarrer une temporisation sous forme de retard à la retombée »*

Fonctionnement

Tableau 17-14 Fonctionnement de l'opération « Démarrer une temporisation sous forme de retard à la retombée »

Opération	Fonctionnement
Démarrer la temporisation	L'opération « Démarrer une temporisation sous forme de retard à la retombée » démarre la temporisation indiquée, lorsque l'état de signal à l'entrée de démarrage **S** passe de « 1 » à « 0 ». Un changement d'état de signal est toujours requis pour valider la temporisation.
Redémarrer la temporisation	La temporisation est redémarrée lorsque l'état de signal à l'entrée **S** passe une nouvelle fois de « 1 » à « 0 » (par exemple après l'initialisation).
Présélectionner la durée d'exécution	La temporisation s'exécute avec la valeur indiquée à l'entrée **TV**.
Initialiser	La temporisation est initialisée lorsque le signal à l'entrée de remise à zéro **R** passe de « 0 » à « 1 », pendant que la temporisation s'exécute.
Interroger l'état de signal	Une interrogation à « 1 » de l'état de signal à la sortie **Q** fournit le résultat « 1 », lorsque l'état de signal à l'entrée S = 1 ou lorsque la temporisation s'exécute.
Interroger la valeur de temps actuelle	Vous pouvez interroger la valeur de temps actuelle à la sortie **BI** à l'aide de la valeur de la fonction S_OFFDT.

17.2.7 Exemple de programme pour une impulsion prolongée

Exemple S_PEXT

L'exemple 17-6 montre un programme d'application de la fonction de temporisation « Impulsion prolongée ».

```
FUNCTION_BLOCK HORLOGE
VAR_INPUT
 MA_TEMPO: TIMER;
END_VAR
VAR_OUTPUT
 RESULTAT: S5TIME;
END_VAR
VAR
 METTRE_1         : BOOL;
 METTRE_0         : BOOL;
 VAL_DCB          : S5TIME;//Valeur temps et valeur
                  //restante, format DCB
 VAL_BIN          : WORD; //valeur temps, binaire
 PRESELECTION     : S5TIME;
END_VAR
BEGIN
 A0.0:= 1;
 METTRE_1:= E0.0;
 METTRE_0:= E0.1;
 PRESELECTION:= T#25S;
```

```
 VAL_DCB:= S_PEXT(T_NO:= MA_TEMPO,
                  S           := METTRE_1,
                  TV          := PRESELECTION,
                  R           := METTRE_0,
                  BI          := VAL_BIN,
                  Q           := A0.7);
```

```
RESULTAT:=VAL_DCB; //Exploitation résultat
                   //comme paramètre de sortie
AW4:= VAL_BIN     //vers sortie pour affichage
END_FUNCTION_BLOCK
```

Exemple 17-6 Exemple de fonction de temporisation

17.2.8 Choix de la temporisation correcte

La figure 17-8 représente les cinq temporisations qui ont été décrites dans ce chapitre. Vous pouvez ainsi choisir celle qui correspond à vos besoins.

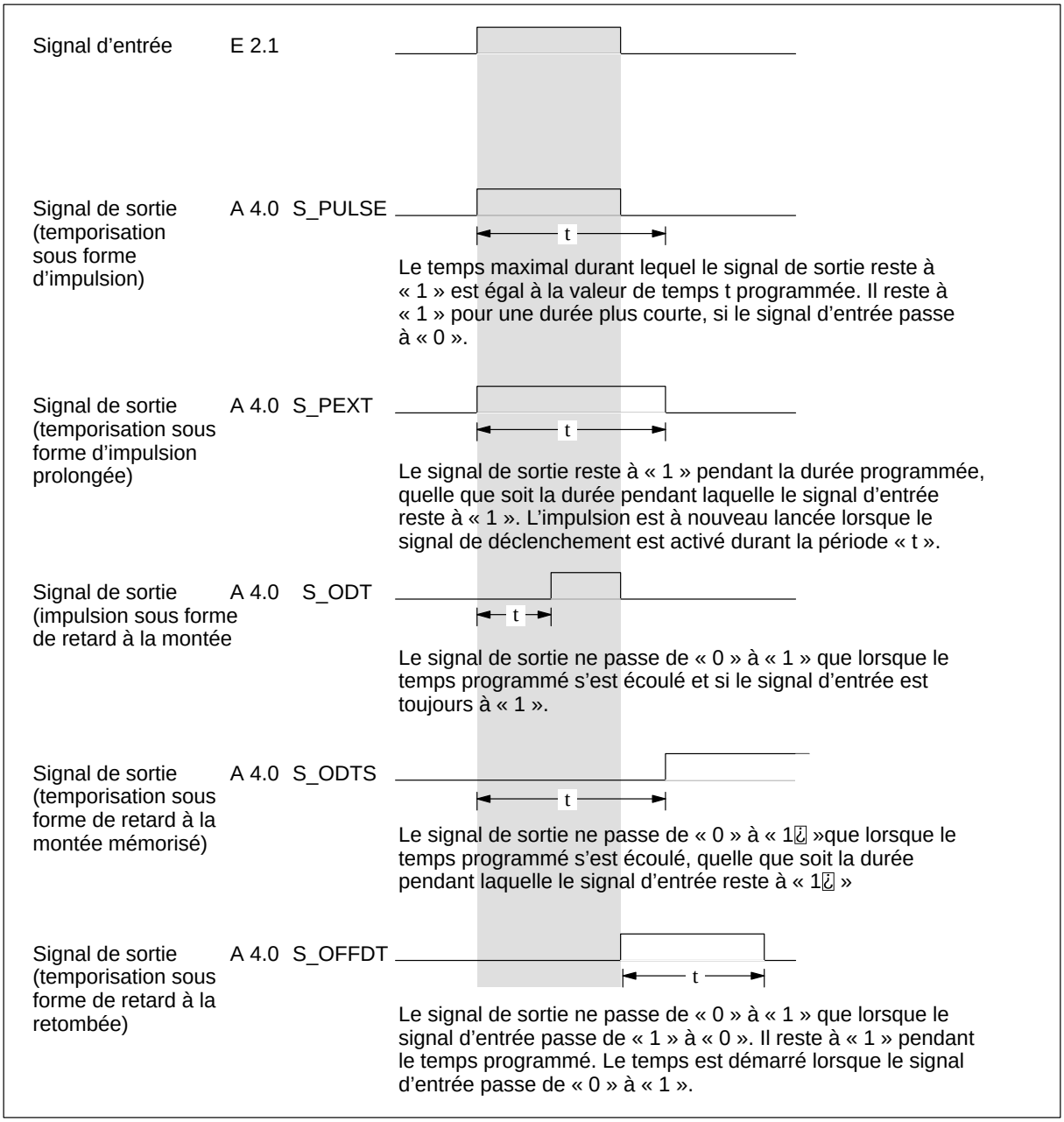


Figure 17-8 *Choix de la temporisation correcte*

Fonctions standard de SCL

18

Présentation

Pour résoudre les tâches répétitives, SCL met à votre disposition une série de fonctions standard que vous pouvez appeler dans vos blocs SCL.

Structure du chapitre

18.1 Conversion de types de données

Présentation

Lorsque vous combinez deux opérandes de types de données différents ou affectez des expressions à des variables, vous devez à chaque fois vous assurer de la compatibilité de leur type de données. Les cas suivants conduisent à des résultats erronés :

- lorsque vous passez à un type d'une autre classe, comme par exemple du type de données binaire au type de données numérique,
- lorsque dans une même classe de types, le type de données cible est moins puissant que le type de données source.

C'est la raison pour laquelle, dans ces deux cas, vous devez réaliser une conversion **explicite** du type de données. De plus amples informations à ce sujet vous sont données au paragraphe 18.2.

En dehors de ces deux cas, le compilateur effectue une conversion automatique dans un format commun, que par la suite nous désignerons par conversion **implicite**.

Conversion automatique de types de données

Le compilateur effectue une conversion implicite des types de données au sein de l'une des deux classes de types de données auxiliaires définies dans le tableau 18-1, en respectant l'ordre indiqué. Le format commun de deux opérandes est toujours le type de données le plus petit, dont la plage des valeurs contient les deux opérandes. Ainsi, le format commun de BYTE et de INTEGER est INTEGER.

Sachez que lors de la conversion d'un type de données dans la classe `ANY_BIT` , les bits à gauche sont mis à `0`.

Tableau 18-1 Ordre de conversion automatique de types de données

Classes	Ordre de conversion
ANY_BIT	`BOOL` ⇒ `BYTE` ⇒ `WORD` ⇒ `DWORD`
ANY_NUM	`INT` ⇒ `DINT` ⇒ `REAL`

L'exemple suivant illustre la conversion automatique de types de données :

```
FUNCTION_BLOCK FB10
VAR
      REGULATEUR_PID_1:BYTE;
      REGULATEUR_PID_2:WORD;
END_VAR
BEGIN
IF (REGULATEUR_PID_1 <> REGULATEUR_PID_2) THEN
      // ...
(*  *  Dans  l'instruction  conditionnelle  IF  /  THEN
précédente, la variable REGULATEUR_PID_1 est convertie
automatiquement en variable de type de données WORD *)
END_FUNCTION_BLOCK
```

Exemple 18-1 Conversion automatique de types de données

18.2 Fonctions standard de conversion de types de données

Conversion explicite

Vous réalisez la conversion explicite de types de données à l'aide des fonctions standard dont la liste figure dans les tableaux 18-2 et 18-3.

Appel de fonction

La description détaillée des appels de fonctions généraux est donnée au chapitre 16.

- Paramètre d'entrée :

Toute fonction de conversion d'un type de données possède exactement un paramètre d'entrée, appelé IN. Puisqu'il s'agit d'une fonction possédant un seul paramètre, il suffit d'indiquer le paramètre effectif.

- Valeur de la fonction

Le résultat correspond toujours à la valeur de la fonction. Les tableaux 18-2 et 18-3 donnent les règles de conversion de données. Le tableau 18-3 précise en outre si le drapeau OK est influencé par la fonction respective.

- Nom de la fonction

Puisque les types de données du paramètre d'entrée et de la valeur de la fonction sont explicités dans le nom de la fonction, ils ne sont pas détaillés dans les tableaux 18-2 et 18-3 : dans le cas de la fonction `BOOL_TO_BYTE`, par exemple, le type de données du paramètre d'entrée est `BOOL`, celui de la valeur de la fonction étant `BYTE`.

Liste des fonctions de conversion (classe A)

Le tableau 18-2 donne la liste des fonctions de conversion des types de données de classe A. Quoique le compilateur les initie implicitement, vous pouvez également les indiquer de manière explicite. Leur résultat est toujours défini.

Tableau 18-2 Fonctions de conversion des types de données de classe A

Nom de la fonction	Règle de conversion
BOOL_TO_BYTE	Complément par des zéros à gauche
BOOL_TO_DWORD	
BOOL_TO_WORD	
BYTE_TO_DWORD	
BYTE_TO_WORD	
CHAR_TO_STRING	Conversion en chaîne (de longueur 1) contenant le même caractère.
DINT_TO_REAL	Conversion en `REAL` conformément à la norme IEEE. La valeur peut être modifiée – en raison de l'autre précision pour `REAL`.
INT_TO_DINT	Pour un paramètre d'entrée négatif, le mot de poids fort de la valeur de lafonction est complété par `16#FFFF`, dans tous les autres cas par des zéros.
INT_TO_REAL	Conversion en `REAL, conformément à la norme` IEEE. La valeur reste identique.
WORD_TO_DWORD	Complément par des zéros à gauche

Liste des fonctions de conversion (classe B)

Le tableau 18-3 donne la liste des fonctions de conversion des types de données de classe B. Vous devez les spécifier explicitement. Si la taille du type de données cible est insuffisante, leur résultat peut être indéfini.

Vous pouvez vérifier vous-même cette éventualité, en insérant une vérification de limites ou alors la faire vérifier par le système, en sélectionnant l'option « Drapeau OK » avant l'effectuer la compilation. Dans les cas où le résultat est indéfini, le système affecte la valeur FALSE à la variable OK.

Tableau 18-3 Fonctions de conversion des types de données de classe B

Nom de la fonction	Règle de conversion	OK
BYTE_TO_BOOL	Copie du bit de poids faible	O
BYTE_TO_CHAR	Reprise de la chaîne binaire	N
CHAR_TO_BYTE	Reprise de la chaîne binaire	N
CHAR_TO_INT	La chaîne binaire du paramètre d'entrée est inscrite dans l'octet de poids faible de la valeur de la fonction. L'octet de poids fort est complété par des zéros.	N
DATE_TO_DINT	Reprise de la chaîne binaire	N
DINT_TO_DATE	Reprise de la chaîne binaire	O
DINT_TO_DWORD	Reprise de la chaîne binaire	N
DINT_TO_INT	Copie du bit de signe. La valeur du paramètre d'entrée est interprétée dans le type de données `INT`. Si la valeur est inférieure à `-32_768` ou supérieure à `32_767`, la variable OK prend la valeur `FALSE`.	
DINT_TO_TIME	Reprise de la chaîne binaire	N
DINT_TO_TOD	Reprise de la chaîne binaire	O
DWORD_TO_BOOL	Copie du bit de pois faible	O
DWORD_TO_BYTE	Copie des 8 bits de poids faible	O
DWORD_TO_DINT	Reprise de la chaîne binaire	N
DWORD_TO_REAL	Reprise de la chaîne binaire	N
DWORD_TO_WORD	Copie des 16 bits de poids faible	O
INT_TO_CHAR	Reprise de la chaîne binaire	O
INT_TO_WORD	Reprise de la chaîne binaire	N
REAL_TO_DINT	Arrondi de la valeur `REAL` IEEE à `DINT`. Si la valeur est inférieure à `-2_147_483_648` ou supérieure à `2_147_483_647`, la variable OK prend la valeur `FALSE`.	O
REAL_TO_DWORD	Reprise de la chaîne binaire	N
REAL_TO_INT	Arrondi de la valeur `REAL` IEEE à `INT`. Si la valeur est inférieure à `-32_768` ou supérieure à `32_767`, la variable `OK` prend la valeur `FALSE`.	O

Tableau 18-3 Fonctions de conversion des types de données de classe B (suite)

Nom de la fonction	Règle de conversion	OK
STRING_TO_CHAR	Copie du premier caractère de la chaîne. Si la longueur de la chaîne est différente de 1, la variable OK prend la valeur FALSE.	O
TIME_TO_DINT	Reprise de la chaîne binaire	N
TOD_TO_DINT	Reprise de la chaîne binaire	N
WORD_TO_BOOL	Copie du bit de poids faible	O
WORD_TO_BYTE	Copie des 8 bits de poids faible	O
WORD_TO_INT	Reprise de la chaîne binaire	N
WORD_TO_BLOCK_DB	Le modèle binaire de WORD est interprété comme numéro de bloc de données.	N
BLOCK_DB_TO_WORD	Le numéro de bloc de données est interprété comme modèle bianire de WORD.	N

Nota

Vous avez en outre la possibilité d'utiliser des **fonctions CEI** pour effectuer la conversion des types de données. Copiez à cet effet la fonction souhaitée de la bibliothèque de STEP 7 STDLIBS\IEC dans le réperoire de votre programme. Vous trouverez des informations détaillées sur chacune de ces fonctions CEI dans la publication **/235/**.

Exemples pour la conversion explicite

Dans l'exemple 18-2, une conversion explicite s'avère indispensable, car le type de données cible est moins puissant que le type de données source.

```
FuNCTION_BLOCK FB10
VAR
      COMMUTATEUR : INT;
      REGULATEUR  : DINT;
END_VAR

BEGIN
COMMUTATEUR := DINT_TO_INT (REGULATEUR);
(* INT est moins puissant que DINT *)
// ...
END_FUNCTION_BLOCK
```

Exemple 18-2 Type de données cible incompatible avec type de données source

Dans l'exemple 18-3, une conversion explicite des données est nécessaire, car le type de données REAL n'est pas autorisé dans une expression arithmétique comportant l'opérateur MOD.

```
FUNCTION_BLOCK FB20
       VAR
         val_ent:INT:=17;
         CONV2 := INT;
       END_VAR

BEGIN
       CONV2 := val_ent MOD REAL_TO_INT (2.3);
       (* MOD ne peut être appliquée qu'à des données de
       type INT ou DINT. *)
       // ...
END_FUNCTION_BLOCK
```

Exemple 18-3 Conversion en raison d'un type de données non autorisé

Dans l'exemple 18-4, la conversion est indispensable car le type de données n'est pas autorisé pour un opérateur logique. NOT ne peut être appliquée qu'à des données de type BOOL, BYTE, WORD ou DWORD.

```
FUNCTION_BLOCK FB30
       VAR
         val_ent:INT:=17;
         CONV1 :=WORD;
       END_VAR

BEGIN
       CONV1 := NOT INT_TO_WORD(val_int);
       (* NOT ne peut être appliquée qu'à des données
       de type INT. *)
// ...
END_FUNCTION_BLOCK
```

Exemple 18-4 Conversion en raison d'un type de données incorrect

L’exemple 18-5 montre la conversion des valeurs d’entrée et de sortie de la périphérie :

```
FUNCTION_BLOCK FB40
      VAR
        rayon_oui: WORD;
        rayon : INT;
      END_VAR

BEGIN
      rayon_oui    := EB0;
      rayon        := WORD_TO_INT(rayon_oui);
(* Conversion lors du passage à une autre classe de types.
La valeur vient de l’entrée et doit être converti pour
pouvoir être exploitée.*)

      rayon        := Rayon(surface:= données_cercle.surface);
      AB0          := WORD_TO_BYTE(INT_TO_WORD(rayon));
(* Le rayon est recalculé à partir de la surface. Il est
donné sous forme d’entier. Avant la sortie, la valeur
est convertie dans une autre classe de types (INT_TO_WORD)
puis dans un type moins puissant (WORD_TO_BYTE) *)
      // ...
END_FUNCTION_BLOCK
```

Exemple 18-5 Conversion d’entrées et de sorties

Fonctions d'arrondi et de troncature

Les fonctions d'arrondi et de troncature de nombres font également partie de la conversion des types de données. Le tableau 18-4 indique le nom, le type de données (du paramètre d'entrée et de la valeur de la fonction) et la tâche de ces fonctions.

Tableau 18-4 Fonctions d'arrondi et de troncature

Nom de la fonction	Type de données du paramètre d'entrée	Type de données de la valeur de la fonction	Tâche
ROUND	REAL	DINT	Arrondir (former un nombre `DINT`)
TRUNC	REAL	DINT	Tronquer (former un nombre `DINT`)

Les exemples suivants illustrent les différents effets de ces fonctions :

```
• ROUND (3.14)     // fonction d'arrondi
                   // résultat : 3
• ROUND (3.56)     // fonction d'arrondi
                   // résultat : 4
• TRUNC (3.14)     // fonction de troncature
                   // résultat : 3
• TRUNC (3.56)     // fonction de troncature
                   // résultat : 3
```

18.3 Fonctions standard numériques

Fonctionnalité

Toute fonction numérique standard possède **un** paramètre d'entrée. Son résultat correspond toujours à la valeur de la fonction. Chacun des tableaux 18-5, 18-6 ou 18-7 spécifie un groupe de fonctions numériques standard, en indiquant leur nom et type de données. Le type de données ANY_NUM correspond à INT, DINT ou REAL.

Liste des fonctions générales

Les fonctions générales permettent de calculer la valeur abslolue, le carré ou la racine carrée d'une grandeur.

Tableau 18-5 Fonctions générales

Nom de la fonction	Type de données du paramètre d'entrée	Type de données de la valeur de la fonction	Description
ABS	ANY_NUM[1]	ANY_NUM	Valeur absolue
SQR	ANY_NUM[1]	REAL	Carré
SQRT	ANY_NUM[1]	REAL	Racine carrée

1 *Sachez que les paramètres d'entrée de type ANYNUM sont convertis de manière interne en variables de type Real.*

Liste des fonctions logarithmiques

Les fonctions logarithmiques permettent de calculer la valeur d'une puissance ou le logarithme d'une grandeur.

Tableau 18-6 Fonctions logarithmiques

Nom de la fonction	Type de données du paramètre d'entrée	Type de données de la valeur de la fonction	Description
EXP	ANY_NUM[1]	REAL	e puissance IN
EXPD	ANY_NUM[1]	REAL	10 puissance IN
LN	ANY_NUM[1]	REAL	Logarithme népérien
LOG	ANY_NUM[1]	REAL	Logarithme décimal

1 *Sachez que les paramètres d'entrée de type ANYNUM sont convertis de manière interne en variables de type Real.*

Nota

Vous avez en outre la possibilité d'utiliser des **fonctions CE**I comme fonctions numériques standard. Copiez à cet effet la fonction souhaitée de la bibliothèque de STEP 7 STDLIBS\IEC dans le réperoire de votre programme. Vous trouverez des informations détaillées sur chacune de ces fonctions CEI dans la publication **/235/**.

Liste des fonctions trigonométriques

Les fonctions trigonométriques indiquées dans le tableau 18-7 calculent des angles en radians.

Tableau 18-7 Fonctions trigonométriques

Nom de la fonction	Type de données du paramètre d'entrée	Type de données de la valeur de la fonction	Description
ACOS	ANY_NUM[1]	REAL	Arccosinus
ASIN	ANY_NUM[1]	REAL	Arcsinus
ATAN	ANY_NUM[1]	REAL	Arctangente
COS	ANY_NUM[1]	REAL	Cosinus
SIN	ANY_NUM[1]	REAL	Sinus
TAN	ANY_NUM[1]	REAL	Tangente

1 *Sachez que les paramètres d'entrée de type* `ANYNUM` *sont convertis de manière interne en variables de type Real.*

Exemples

Le tableau 18-8 représente des appels de fonctions numériques standard et indique le résultat correspondant :

Tableau 18-8 Appels de fonctions numériques standard

Appel	Résultat
RESULTAT := ABS (-5);	5
RESULTAT := SQRT (81.0);	9
RESULTAT := SQR (23);	529
RESULTAT := EXP (4.1);	60.340 ...
RESULTAT := EXPD (3);	1_000
RESULTAT := LN (2.718_281);	1
RESULTAT := LOG (245);	2.389_166 ...
PI := 3. 141 592; RESULTAT := SIN (PI / 6);	0.5
RESULTAT := ACOS (0.5);	1.047_197 (=PI / 3)

18.4 Fonctions standard sur les chaînes binaires

Fonctionnalité

Toute fonction standard sur les chaînes binaires possède deux paramètres d'entrée désignés par `IN` ou `N`. Leur résultat correspond toujours à la valeur de la fonction. Le tableau 18-9 indique leur nom ainsi que les types de données des deux paramètres d'entrée et de la valeur de la fonction. La signification des paramètres d'entrée est la suivante :

- paramètre d'entrée IN : mémoire dans laquelle sont réalisées les opérations de décalage de bits,
- paramètre d'entrée N : nombre de rotations pour les fonctions de rotation ROL et ROR ou nombre de chiffres à décaler pour les fonctions SHL et SHR.

Liste des fonctions

Le tableau 18-9 indique les fonctions standard sur chaînes binaires.

Tableau 18-9 Fonctions standard sur chaînes binaires

Nom de la fonction	Type de données du paramètred'entrée `IN`	Type de données du paramètred'entrée `N`	Type de données de la valeur de la fonction	Tâche
ROL	BOOL	INT	BOOL	Une rotation vers la gauche du nombre de chiffres indiqué par le paramètre `N` va être effectuée sur la valeur données par le paramètre `IN`.
	BYTE	INT	BYTE	
	WORD	INT	WORD	
	DWORD	INT	DWORD	
ROR	BOOl	INT	BOOL	Une rotation vers la droite du nombre de chiffres indiqué par le paramètre `N` va être effectuée sur la valeur données par le paramètre `IN`.
	BYTE	INT	BYTE	
	WORD	INT	WORD	
	DWORD	INT	DWORD	
SHL	BOOL	INT	BOOL	Dans la valeur représentée par le paramètre `IN`, le nombre de chiffres indiqué par le paramètre `N` vont être décalés vers la gauche et complétés par des 0 à droite.
	BYTE	INT	BYTE	
	WORD	INT	WORD	
	DWORD	INT	DWORD	
SHR	BOOL	INT	BOOL	Dans la valeur représentée par le paramètre `IN`, le nombre de chiffres indiqué par le paramètre `N` vont être décalés vers la droite et complétés par des 0 à gauche.
	BYTE	INT	BYTE	
	WORD	INT	WORD	
	DWORD	INT	DWORD	

Nota

Vous avez en outre la possibilité d'utiliser des **fonctions CE**I pour effectuer des opérations sur les chaînes binaires. Copiez à cet effet la fonction souhaitée de la bibliothèque de STEP 7 `STDLIBS\IEC` dans le réperoire de votre programme. Vous trouverez des informations détaillées sur chacune de ces fonctions CEI dans la publication **/235/**.

Exemples

Le tableau 18-10 représente des appels de fonctions standard sur les chaînes binaires et indique le résultat correspondant :

Tableau 18-10 Appels de fonctions standard sur les chaînes binaires

Appel	Résultat
RESULTAT := ROL (IN:=2#1101_0011, N:=5); // IN := 211 décimal	2#0111_1010 (= 122 décimal)
RESULTAT := ROR (IN:=2#1101_0011, N:=2); // IN := 211 décimal	2#1111_0100 (= 244 décimal)
RESULTAT := SHL (IN:=2#1101_0011, N:=3); // IN := 211 décimal	2#1001_1000 (= 152 décimal)
RESULTAT := SHR (IN:=2#1101_0011, N:=2); // IN := 211 décimal	2#0011_0100 (= 52 décimal)

Interface d’appel

19

Présentation

Le système d’exploitation des CPU S7 contient des fonctions système et des fonctions standard que vous pouvez utiliser pour la programmation dans SCL. Il s’agit des :

- blocs d’organisation (OB)
- fonctions système (SFC)
- blocs fonctionnels système (SFB)

Structure du manuel

19.1 Interface d’appel

Présentation

Un bloc peut être appelé de manière symbolique ou absolue. Vous devez à cet effet indiquer soit le mnémonique du bloc, que vous avez déclaré dans la table des mnémoniques, soit son numéro de désignation absolue.

A l’appel du bloc, vous devez affecter les **paramètres effectifs**, dont les valeurs vont être utilisées par le bloc durant l’exécution du programme, aux **paramètres formels**, dont vous avez défini le nom et le type de données lorsque vous avez créé le bloc paramétrable.

Toutes les informations relatives à ce sujet sont fournies dans le manuel **/235/** . Celui-ci présente non seulement les principales fonctions disponibles dans S7, mais décrit également, dans une partie de référence, les interfaces qui permettent de les utiliser dans votre programme utilisateur.

Exemple SFC 31

Les instructions suivantes permettent d’appeler la fonction système SFC 31 (interrogation de l’alarme horaire) :

```
FUNCTION_BLOCK FY20

VAR
      Resultat: INT;
END_VAR
BEGIN
      // ...
      Resultat:= SFC31 (NO_OB:= 10,ETAT:= MW100 );
      // ...
      // ...
END_FUNCTION_BLOCK
```

Exemple 19-1 Interrogation de l’alarme horaire

Résultats

La valeur de la fonction est du type entier. Si elle est > = 0, le bloc a été exécuté sans erreur, si elle est < 0, une erreur s’est produite. Après avoir appelé le bloc, vous avez la possibilité d’exploiter le paramètre de sortie ENO défini automatiquement.

Appel conditionnel

Si vous voulez réaliser un appel conditionnel, vous devez affecter la valeur 0 au paramètre d’entrée **EN** (par exemple à l’entrée E0.3), afin que le bloc ne soit pas appelé. Si vous affectez la valeur 1 au paramètre EN, la fonction est appelée. Si aucune erreur ne se produit durant l’exécution du bloc, le **paramètre de sortie ENO** prend alors également la valeur « 1 » (sinon il est mis à « 0 »).

Nota

Dans un bloc fonctionnel ou bloc fonctionnel système, l'information qui, s'il s'agissait d'une fonction, serait transmise par la valeur en retour, doit ici être enregistrée dans un paramètre de sortie. Celui-ci est ensuite exploité grâce au bloc de données d'instance. De plus amples informations à ce sujet sont données au chapitre 16.

19.2 Interface d’échange avec les OB

Blocs d’organisation

Les blocs d’organisation constituent l’interface entre le système d’exploitation de la CPU et le programme utilisateur. Ils permettent d’amener certaines parties de programmes à être exécutées :

- à la mise en route de la CPU
- de manière cyclique ou à une cadence donnée
- à des instants ou jours donnés
- après écoulement d’une durée donnée
- à l’apparition d’erreurs
- lors du déclenchement d’alarmes de processus ou de communication

L’ordre de traitement des blocs d’organisation dépend de leur priorité.

OB disponibles

Les CPU ne sont pas toutes en mesure de traiter l’ensemble des OB disponibles dans S7. Vérifiez dans les caractéristiques techniques de votre CPU, quels sont les OB à votre diposition.

Informations supplémentaires

L’aide en ligne ainsi que les manuels suivants constituent une aide supplémentaire :

- **/70/** Manuel: *Automate programmable S7-300, Installation et configuration - Caractéristiques des CPU*
  Ce manuel contient les fiches techniques décrivant les caractéristiques des diverses CPU S7-300, ainsi que les événements de déclenchement des OB.
- **/100/** Manuel de mise en œuvre: *Systèmes d’automatisation S7-400, M7-400, Installation et configuration*
  Ce manuel contient les fiches techniques décrivant les caractéristiques des diverses CPU S7-400, ainsi que les événements de déclenchement des OB.

Annexes

Description formelle du langage

A

Description du langage SCL

Les diagrammes syntaxiques servent de base à la description du langage faite dans les différents chapitres. Ils permettent d'illustrer la structure syntaxique (c'est-à-dire grammaticale) dans SCL. Ces diagrammes sont regroupés au complet, avec les éléments de langage correspondants, dans les annexes B et C.

Structure de cette annexe

A.1 Présentation

Diagramme syntaxique

Le diagramme syntaxique est une représentation graphique de la structure du langage. Cette structure est constituée d'une suite de règles, pouvant elles-mêmes se fonder sur des règles déjà énoncées.

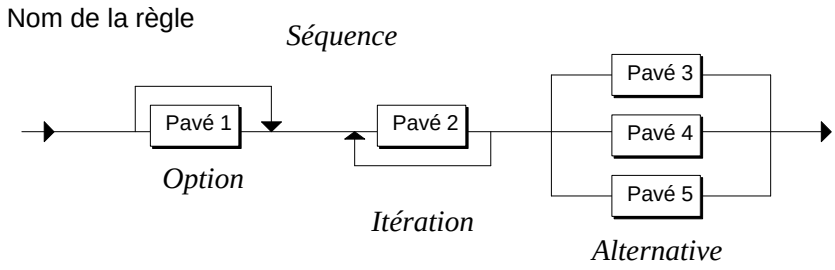


Figure A-1 *Exemple de diagramme syntaxique*

Le diagramme syntaxique se lit de gauche à droite. En voici la structure :

- séquence : suite de pavés
- option : branchement facultatif
- itération : répétition d'un branchement
- alternative : branchement

Types de pavés

Un pavé peut être un élément de base ou alors être lui-même composé d'autres éléments de base. La figure suivante illustre les types de symboles représentant les pavés :

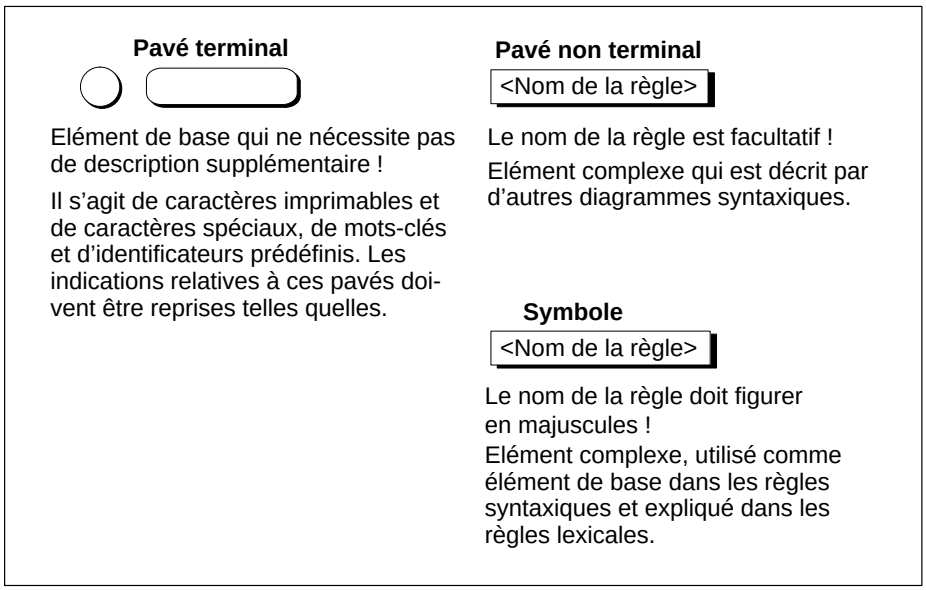


Figure A-2 *Types de symboles représentant les pavés*

Règles

Les règles que vous appliquez pour structurer votre programme SCL peuvent être réparties dans deux groupes, les **règles lexicales** et les **règles syntaxiques**.

Règles lexicales

Elles décrivent la structure des éléments (symboles) soumis à l'analyse lexicale par le compilateur. Le format utilisé dans leur notation n'est pas libre, ce qui signifie que vous devez respecter scrupuleusement les règles énoncées, en particulier :

- l'insertion de caractères de mise en forme n'est pas autorisée,
- l'insertion de blocs ou de lignes de commentaires n'est pas possible,
- l'insertion d'attributs d'identificateurs est impossible.

IDENTIFICATEUR

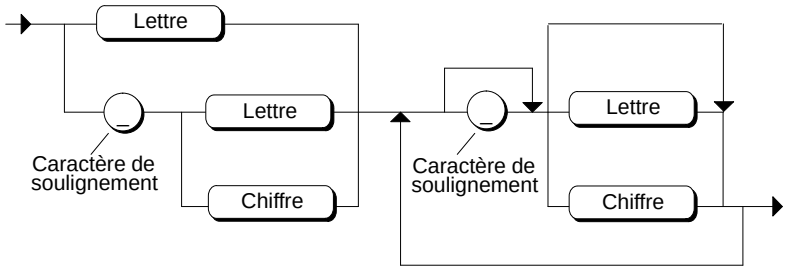


Figure A-3 *Exemple de règle lexicale*

Il s'agit de la règle lexicale de l'IDENTIFICATEUR, qui décrit sa structure (nom), comme par exemple :

```
TABLEAU_MES_12
VALEUR_CONSIGNE_B_1
```

Règles syntaxiques

Les règles syntaxiques décrivent la structure de SCL, en se fondant sur les règles lexicales. Vous pouvez créer votre programme SCL dans un format libre, dans le respect des règles énoncées :

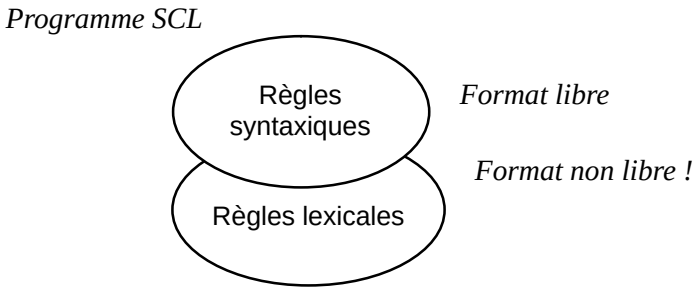


Figure A-4 *Hiérarchie des règles et format libre*

Conventions

Chaque règle est précédée de son nom. Si elle est utilisée dans une règle de niveau hiérarchique supérieur, son nom y figure sous forme de pavé nom terminal.

Si le nom de la règle est inscrit en majuscules, il s'agit d'un symbole qui est décrit dans les règles lexicales.

Sémantique

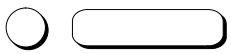
Les règles énoncées permettent uniquement de représenter la structure formelle de la langue, sans toujours expliciter leur signification. Des informations supplémentaires accompagnent donc les règles pour les points importants. En voici des exemples :

- dans le cas d'éléments de signification différente : par exemple dans la règle d'indication de la date, pour SUITE DE CHIFFRES DECIMAUX, année, mois ou jour. Le nom désigne explicitement l'utilisation.
- d'importantes restrictions sont signalées à côté des règles : il est par exemple précisé qu'un mnémonique doit être défini dans la table des mnémoniques.

A.2 Présentation des pavés terminaux

Définition

Un pavé terminal est un élément de base dont la description ne nécessite pas l'usage d'une règle supplémentaire. Il est représenté par les schémas suivants dans les diagrammes syntaxiques :



Le pavé terminal est représenté par un rectangle aux bouts arrondis ou par un cercle. Il contient soit un nom (en majuscules), soit directement une information.

Celui-ci définit le nombre de caractères ASCII utilisables.

Figure A-5 *Schémas représentant les pavés terminaux*

Présentation

Les paragraphes A.3 et A.4 traitent de l'utilisation de caractères individuels, à savoir :

- les lettres, chiffres, caractères imprimables et caractères spéciaux,
- caractères de mise en forme et séparateurs dans les règles lexicales,
- préfixes de constantes littérales,
- caractères de mise en forme et séparateurs dans les règles syntaxiques,
- opérateurs.

Les paragraphes A.5 et A.6 traitent des mots-clés des identificateurs prédéfinis à partir de chaînes de caractères. Les tableaux sont classés d'après l'ordre alphabétique des abréviations SIMATIC (allemandes). Les abréviations CEI (internationales) correspondantes sont indiquées en cas de différence pour les :

- mots-clés et identificateurs prédéfinis
- identificateurs d'opérandes et mots-clés de blocs

A.3 Pavés terminaux dans les règles lexicales

Présentation

Les tableaux suivants décrivent les pavés terminaux en indiquant l'ensemble des éléments du jeu de caractères ASCII correspondant :

Lettres et chiffres

Il s'agit des caractères les plus utilisés. L'IDENTIFICATEUR (voir figure A-3) est par exemple composé de lettres, de chiffres et du caractère de soulignement.

Tableau A-1 Lettres et chiffres

Caractère	**Sous-groupe**	**Eléments du jeu de caractères**
Lettre	Majuscule	A.. Z
	Minuscule	a.. z
Chiffre	Chiffre décimal	0.. 9
Chiffre octal	Chiffre octal	0.. 7
Chiffre hexadécimal	Chiffre hexadécimal	0.. 9, A.. F, a.. f
Bit	Chiffre binaire	0, 1

Caractères imprimables et caractères spéciaux

Les chaînes de caractères, commentaires et mnémoniques utilisent le jeu de caractères ASCII étendu complet.

Tableau A-2 Caractères imprimables et caractères spéciaux

Caractère	**Sous-groupe**	**Eléments du jeu de caractères**
Caractère imprimable	dépend du code de caractères utilisé. Pour le code ASCII, à partir du caractère équivalant au caractère décimal 31, sans DEL, ni les caractères de substitution suivants :	tous les caractères imprimables
Caractère de substitution	Dollar	$
	Apostrophe	'
Caractère de commande	$P ou $p	Saut de page (formfeed, page)
	$L ou $l	Changement de ligne (linefeed)
	$R oder $r	Retour chariot (carriage return)
	$T oder $t	Tabulation
Représentation de substitution en code hexadécimal	$hh	Caractères quelconques saisis en code hexadécimal (hh)

A.4 Caractères de mise en forme, séparateurs et opérateurs

Dans les règles lexicales

Le tableau A-3 montre l'utilisation de caractères individuels du jeu de caractères ASCII comme caractères de mise en forme et comme séparateurs dans les règles lexicales (voir annexe B).

Tableau A-3 Caractères de mise en forme et séparateurs dans les règles lexicales

Caractère	Description
:	Séparateur entre heures, minutes et secondes. Attributs
.	Séparateur dans l'adressage absolu, représentation de nombres réels et d'intervalles de temps
' '	Caractère et chaîne de caractères
" "	Caractère d'introduction de mnémoniques, d'après les règles de l'éditeur de mnémoniques
_ caractère de soulignement	Séparateur de valeurs numériques dans les constantes littérales, peut être utilisé dans les IDENTIFICATEURS
$	Caractère d'alignement pour introduire un caractère de commande ou un caractère de substitution
$> $<	Interruption de la chaîne si sa longueur dépasse une ligne ou pour introduire des commentaires

Dans les constantes littérales

Le tableau suivant montre l'utilisation de caractères individuels et de chaînes de caractères comme constantes littérales dans les règles lexicales (voir annexe B). Il s'applique à la fois aux abréviations SIMATIC et CEI.

Tableau A-4 Abréviations pour les constantes, classées par ordre alphabétique

Préfixe	Caractérise	Règle lexicale
2#	CONSTANTE LITTERALE ENTIERE	Suite de chiffres binaires
8#	CONSTANTE LITTERALE ENTIERE	Suite de chiffres octaux
16#	CONSTANTE LITTERALE ENTIERE	Suite de chiffres hexadécimaux
D#	Indication de temporisation	DATE
DATE#	Indication de temporisation	DATE
DATE_AND_TIME#	Indication de temporisation	DATE ET HEURE
DT#	Indication de temporisation	DATE ET HEURE
E	Séparateur pour CONST LITTER REELLE	Exposant
e	Séparateur pour CONST LITTER REELLE	Exposant
D	Séparateur pour intervalle de temps (Day)	Jours (règle : représentation par niveaux)
H	Séparateur pour intervalle de temps (Hour)	Heures (règle : représentation par niveaux)
M	Séparateur pour intervalle de temps (Minutes)	Minutes (règle : représentation par niveaux)
MS	Séparateur pour intervalle temps (Milliseconds)	Millisec. (règle : représentation par niveaux)
S	Séparateur pour intervalle de temps (Seconds)	Secondes (règle : représentation par niveaux)
T#	Indication de temporisation	DUREE
TIME#	Indication de temporisation	DUREE
TIME_OF_DAY#	Indication de temporisation	HEURE DU JOUR
TOD#	Indication de temporisation	HEURE DU JOUR

Dans les règles syntaxiques

Le tableau suivant montre l'utilisation de caractères individuels comme caractères de mise en forme et comme séparateurs dans les règles syntaxiques, de même que dans les commentaires et les attributs (voir annexes B.2 et B.3).

Tableau A-5 Caractères de mise en forme et séparateurs dans les règles syntaxiques

Caractère	Description	Règle syntaxique, commentaire ou attribut
:	Séparateur pour indication du type dans une instruction après un repère de saut	Déclaration de variable, déclaration d'instance, fonction, section des instructions, instruction CASE
;	Fin de déclaration ou d'instruction	Déclaration de constante, déclaration de variable, section des instructions, section d'affectation du DB, section des constantes, section des repères de saut, déclaration de composantes
,	Séparateur dans les listes et les sections de repères de saut	Déclaration de variable, spécification du type de données ARRAY, liste d'initialisation de tableau, paramètres du FB, de la FC, liste de valeurs, déclaration des instances
..	Indication de plage	Spécification du type de données ARRAY, liste de valeurs
.	Séparateur pour nom de FB et de DB, adressage absolu	Appel du FB, variable structurée
( )	Appel de fonction et de bloc fonctionnel ; mise entre parenthèses dans une expression ; liste d'initialisation de tableaux	Appel de la fonction, appel du FB, expression, liste d'initialisation de tableau, multiplication simple, expression de puissance
[ ]	Déclaration de tableau, variable structurée, tableau partiel, indexation pour variables globales et chaînes de caractères	Spécification de type de données ARRAY, spécification du type de données STRING
(* *)	Bloc de commentaire	voir annexe B
//	Ligne de commentaire	voir annexe B
{}	Bloc d'attributs	Indication d'attributs
%	Introduction pour l'indicateur direct	Pour programmer conformément à la norme CEI, vous pouvez utilisez %M4.0 à la place de M4.0.

Opérateurs

Le tableau A-6 représente tous les opérateurs de SCL, les mots-clés comme par exemple AND, ainsi que les autres opérateurs utilisés comme caractère individuels. Il s'applique à la fois aux abréviations SIMATIC et CEI

Tableau A-6 Opérateurs de SCL

Opérateur	Description	Règle syntaxique
:=	Opérateur d'affectation, affectation de départ, initialisation de types de données	Affectation de valeur, section d'affectation de DB, section des constantes, affectation de la sortie, affectation de l'entrée, affectation de l'entrée/sortie
+, -	Opérateurs arithmétiques : opérateurs unaires, signes	Expression, expression simple, expression de puissance
+, -, *, / MOD; DIV	Opérateurs arithmétiques de base	Opérateur arithmétique de base, expression, multiplication simple
**	Opérateurs arithmétiques de base : opérateur de puissance	Expression
NOT	Opérateurs logiques : négation	Expression
AND, &, OR; XOR,	Opérateurs logiques de base	Opérateur logique de base
<,>,<=,>=,=,<>	Opérateurs de comparaison	Opérateur de comparaison

A.5 Mots-clés et identificateurs prédéfinis

Mots-clés et identificateurs prédéfinis

Le tableau A-7 contient la liste alphabétique des mots-clés et des identificateurs prédéfinis dans SCL. Il en donne également une description et cite la règle syntaxique correspondante, énoncée en annexe C, dans laquelle ils figurent sous forme de pavés terminaux. Les mots-clés restent les mêmes quelles que soient les abréviations choisies.

Tableau A-7 Liste alphabétique des mots-clés et des identificateurs prédéfinis dans SCL

Mot-clé	Description	Règle syntaxique
AND	Opérateur logique	Opérateur logique de base
ANY	Désignation du type de données ANY	Spécification du type de données d'un paramètre
ARRAY	Début de la spécification d'un tableau, suivi de la liste des indices indiquée entre '[' et ']'	Spécification du type de données ARRAY
BEGIN	Début de la section des instructions d'un bloc de code ou de la section d'initialisation d'un bloc de données	Bloc d'organisation, fonction, bloc fonctionnel, bloc de données
BLOCK_DB	Désignation du type de données BLOCK_DB	Spécification du type de données d'un paramètre
BLOCK_FB	Désignation du type de données BLOCK_FB	Spécification du type de données d'un paramètre
BLOCK_FC	Désignation du type de données BLOCK_FC	Spécification du type de données d'un paramètre
BLOCK_SDB	Désignation du type de données BLOCK_SDB	Spécification du type de données d'un paramètre
BOOL	Type de données simple pour données binaires	Type de données binaire
BY	Début de l'incrément	Instruction FOR
BYTE	Type de données simple	Type de données binaire
CASE	Début de l'instruction de contrôle de selection	Instruction CASE
CHAR	Type de données simple	Type caractère
CONST	Début de la définition d'une constante	Section des constantes
CONTINUE	Instruction de contrôle pour les boucles FOR, WHILE et REPEAT	Instruction CONTINUE
COUNTER	Type de données du compteur, uniquement utilisable dans la section des paramètres	Spécification du type de données d'un paramètre
DATA_BLOCK	Début d'un bloc de données	Bloc de données
DATE	Type de données simple pour la date	Type de temporisation
DATE_AND_TIME	Type de données complexe pour la date et l'heure	Voir tableau C-4
DINT	Type de données simple pour un entier avec double précision	Type de données numérique
DIV	Opérateur de division	Opérateur arithmétique de base, multiplication simple
DO	Début de la section des instructions pour l'instruction FOR	Instruction FOR, Instruction WHILE
DT	Type de données simple pour la date et l'heure	Voir tableau C-4
DWORD	Type de données simple double mot	Type de données binaire
ELSE	Début du cas où aucune condition n'est remplie	Instruction IF
ELSIF	Début de l'alternative	Instruction IF
EN	Drapeau pour la validation du bloc	

Tableau A-7 Liste alphabétique des mots-clés et des identificateurs prédéfinis dans SCL (suite)

Mot-clé	Description	Règle syntaxique
END_CASE	Fin de l'instruction CASE	Instruction CASE
END_CONST	Fin de la définition d'une constante	Section des constantes
END_DATA_BLOCK	Fin d'un bloc de données	Bloc de données
END_FOR	Fin de l'instruction FOR	Instruction FOR
END_FUNCTION	Fin d'une fonction	Fonction
END_FUNC-TION_BLOCK	Fin d'un bloc fonctionnel	Bloc fonctionnel
END_IF	Fin de l'instruction IF	Instruction IF
END_LABEL	Fin d'une section déclaration de repères de saut	Section de déclaration
END_ORGANIZA-TION_BLOCK	Fin d'un bloc d'organisation	Bloc d'organisation
END_REPEAT	Fin de l'instruction REPEAT	Instruction REPEAT
END_STRUCT	Fin de la spécification d'une structure	Spécification du type de données STRUCT
END_TYPE	Fin d'un UDT	Type de données utilisateur
END_VAR	Fin de la section de déclaration	section des variables temporaires, section des variables statiques, section des paramètres
END_WHILE	Fin de l'instruction WHILE	Instruction WHILE
ENO	Drapeau d'erreur d'un bloc	
EXIT	Abandon direct de la boucle	EXIT
FALSE	Constante booléenne prédéfinie : condition logique non remplie, valeur = 0	
FOR	Début de l'instruction de contrôle pour l'exécution d'une boucle	Instruction FOR
FUNCTION	Début d'une fonction	Fonction
FUNCTION_BLOCK	Début d'un bloc fonctionnel	Bloc fonctionnel
GOTO	Instruction de branchement vers un repère de saut	Branchement de programme
IF	Début d'instruction de contrôle pour la sélection	Instruction IF
INT	Type de données simple pour un nombre entier avec simple précision	Type de données numérique
LABEL	Début de section de déclaration de repères de saut	Section des repères de saut
MOD	Opérateur arithmétique pour le reste d'une division	Opérateur arithmétique de base, multiplication simple
NIL	Pointeur zéro	
NOT	Opérateur logique, fait partie des opérateurs unaires	Expression, opérande
OF	Début de la spécification du type de données	Spécification du type de données ARRAY, instruction CASE
OK	Drapeau indiquant si les instructions d'un bloc ont été exécutées sans erreur	
OR	Opérateur logique	Opérateur logique de base
ORGANIZATION_ BLOCK	Début d'un bloc d'organisation	Bloc d'organisation
POINTER	Type de données pointeur, uniquement autorisé dans la déclaration des paramètres dans la section de déclaration, n'est pas traité dans SCL	Voir chapitre 10
REAL	Type de données simple	Type de données numérique

Tableau A-7 Liste alphabétique des mots-clés et des identificateurs prédéfinis dans SCL (suite)

Mot-clé	Description	Règle syntaxique
REPEAT	Début de l'instruction de contrôle pour l'exécution d'une boucle	Instruction REPEAT
RETURN	Instruction de commande pour quitter un sous-programme	Instruction RETURN
S5TIME	Type de données simple pour les indications de temporisations, format S5 spécial	Type de temporisation
STRING	Type de données pour chaîne de caractère	Spécification du type de données STRING
STRUCT	Début de spécification d'une structure, suivie de la liste des composantes	Spécification du type de données STRUCT
THEN	Début des tâches à exécuter si la condition est remplie	Instruction IF
TIME	Type de données simple pour les indications de temporisations	Type de temporisation
TIMER	Type de données pour temporisation, uniquement utilisable dans la section des paramètres	Spécification du type de données des paramètres
TIME_OF_DAY	Type de données simple pour l'heure du jour	Type de temporisation
TO	Début de la valeur finale	Instruction FOR
TOD	Type de données simple pour l'heure du jour	Type de temporisation
TRUE	Constante booléenne prédéfinie : condition logique remplie, valeur différente de 0	
TYPE	Début d'un UDT	Type de données utilisateur
UNTIL	Début de la condition d'abandon pour l'instruction REPEAT	Instruction REPEAT
VAR	Début d'une section de déclaration	Section des variables statiques
VAR_TEMP	Début d'une section de déclaration	Section des variables temporaires
VAR_INPUT	Début d'une section de déclaration	Section des paramètres
VAR_IN_OUT	Début d'une section de déclaration	Section des paramètres
VAR_OUTPUT	Début d'une section de déclaration	Section des paramètres
VOID	Aucune valeur en retour dans l'appel d'une fonction	Voir chapitre 8
WHILE	Début d'instruction de contrôle pour l'exécution d'une boucle	Instruction WHILE
WORD	Type de données simple mot	Type de données binaire
XOR	Opérateur logique	Opérateur logique

A.6 Identificateurs d'opérandes et mots-clés de blocs

Données système globales

Le tableau A-8 donne la liste des abréviations ainsi que la description des abréviations SIMATIC correspondant aux identificateurs d'opérandes SCL :

- indication de l'identificateur d'opérande :
  préfixe de mémoire (A, E, M, PA, PE) ou bloc de données (D)
- indication de la taille de l'élément de données :
  préfixe de taille (facultatif ou B, D, W, X)

L'abréviation résulte de la combinaison entre l'identificateur d'opérande (préfixe de mémoire ou D pour bloc de données) et le préfixe de taille qui correspondent tous deux à des règles lexicales. Les abréviations CEI figurent en regard des abréviations SIMATIC.

Tableau A-8 Identificateurs d'opérandes des données système globales

Abréviations SIMATIC	Abréviations CEI	Préfixe de mémoire ou bloc de données	Préfixe de taille
A	Q	Sortie (par la mémoire image)	Bit
AB	QB	Sortie (par la mémoire image)	Octet
AD	QD	Sortie (par la mémoire image)	Double mot
AW	QW	Sortie (par la mémoire image)	Mot
AX	QX	Sortie (par la mémoire image)	Bit
D	D	Bloc de données	Bit
DB	DB	Bloc de données	Octet
DD	DD	Bloc de données	Double mot
DW	DW	Bloc de données	Mot
DX	DX	Bloc de données	Bit
E	I	Entrée (par la mémoire image)	Bit
EB	IB	Entrée (par la mémoire image)	Octet
ED	ID	Entrée (par la mémoire image)	Double mot
EW	IW	Entrée (par la mémoire image)	Mot
EX	IX	Entrée (par la mémoire image)	Bit
M	M	Mémento	Bit
MB	MB	Mémento	Octet
MD	MD	Mémento	Double mot
MW	MW	Mémento	Mot
MX	MX	Mémento	Bit
PAB	PQB	Sortie (par la périphérie directe)	Octet
PAD	PQD	Sortie (par la périphérie directe)	Double mot
PAW	PQW	Sortie (par la périphérie directe)	Mot
PEB	PIB	Entrée (par la périphérie directe)	Octet
PED	PID	Entrée (par la périphérie directe)	Double mot
PEW	PIW	Entrée (par la périphérie directe)	Mot

Mots-clés de blocs Ils sont utilisés pour l'adressage absolu de blocs. Le tableau est classé d'après les abréviations SIMATIC et indique les abréviations CEI correspondantes.

Tableau A-9 Mots-clés de blocs ainsi que compteurs et temporisations

Abréviations SIMATIC	Abréviations CEI	Préfixe de mémoire ou bloc de données
DB	DB	Bloc de données (Data-Block)
FB	FB	Bloc fonctionnel (Function-Block)
FC	FC	Fonction (Function)
OB	OB	Bloc d'organisation (Organization-Block)
SDB	SDB	Bloc de données système (System-Data-Block)
SFC	SFC	Fonction système (System-Function)
SFB	SFB	Bloc fonctionnel système (System-Function-Block)
T	T	Temporisation (Timer)
UDT	UDT	Type de données global ou utilisateur (Userdefined Data-Type)
Z	C	Compteur (Counter)

A.7 Présentation des pavés non terminaux

Définition

Un pavé non terminal est un élément complexe qui est décrit par une autre règle. Il est représenté par un rectangle. Le nom qui y figure correspond à la règle suivante, etc.

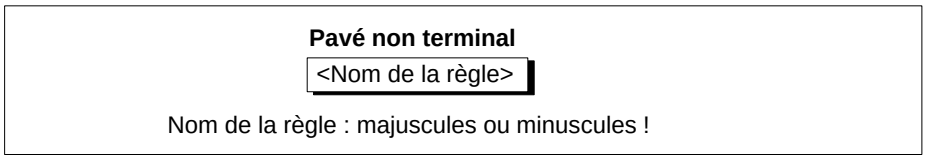


Figure A-6 *Pavé non terminal*

Cet élément figure dans les règles lexicales et dans les règles syntaxiques.

A.8 Présentation des symboles

Définition

Un symbole est un élément complexe utilisé comme élément de base dans les règles syntaxiques et décrit dans les règles lexicales. Il est représenté par un rectangle. Le NOM en lettre majuscules correspond au nom de la règle lexicale suivante (sans rectangle).

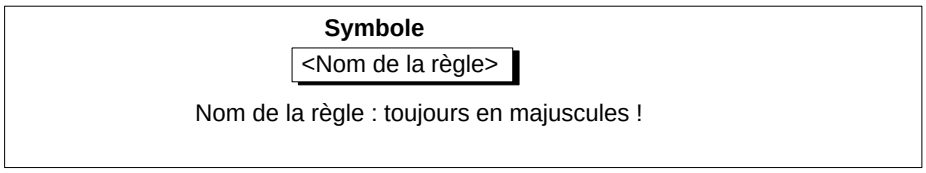


Figure A-7 *Symbole*

Présentation

Les symboles définis représentent les identificateurs qui découlent des règles lexicales. Ils décrivent :

- les identificateurs
- l'attribution de noms dans SCL
- les constantes et drapeaux prédéfinis

A.9 Identificateurs

Identificateurs dans SCL

Les identificateurs vous permettent d'adresser les éléments de langage de SCL. Le tableau A-10 décrit les divers types d'identificateurs :

Tableau A-10 Ensemble des types d'identificateurs dans SCL

Type d'identificateur	Remarques, exemples
Mots-clés	p. ex. instructions de contrôle `BEGIN, DO, WHILE`
Noms prédéfinis	Noms de • types de données standard (p. ex. BOOL, BYTE, INT) • fonctions standard prédéfinies • constantes standard `TRUE` et `FALSE`
Identificateurs d'opérandes dans les identificateurs standard	pour les données système et les blocs de données globaux : p. ex. `E1.2, MW10, FC20, T5, DB30, DB10.D4.5`
Noms quelconques de votre choix d'après la règle IDENTIFICATEUR	Noms de • variables déclarées • composantes de structures • paramètres • constantes déclarées • repères de saut
Mnémoniques dans l'éditeur de mnémoniques	suivent la règle lexicale IDENTIFICATEUR ou la règle lexicale des mnémoniques, c'est-à-dire figurent entre guillemets, comme par exemple "xyz"

Majuscules et minuscules

L'utilisation de majuscules ou de minuscules est insignifiante pour les mots-clés. Depuis la version 4.0 de SCL, la distinction entre majuscules et minuscules n'est pas non plus faite pour les noms prédéfinis et les noms de votre choix, les variables par exemple, et pour les mnémoniques déclarés dans la table des mnémoniques. Le tableau A-11 vous en donne un résumé.

Tableau A-11 Distinction entre les majuscules et les minuscules pour les types d'identificateurs

Type d'identificateur	Distinction majuscules/minuscules ?
Mots-clés	non
Noms prédéfinis pour les types de données standard	non
Noms des fonctions standard prédéfinies	non
Noms des constantes standard prédéfinies	non
Identificateurs d'opérandes pour les identificateurs absolus	non
Noms quelconques	non
Mnémoniques de la table	non

Les noms des fonctions standard, comme par exemple BYTE_TO_WORD et ABS peuvent donc également être écrits en minuscules. Il en est de même pour les paramètres des fonctions de temporisation ou de comptage, comme par exemple SV, se ou ZV.

A.10 Attribution de noms dans SCL

Choix du nom

Deux possibilités s'offrent à vous pour l'attribution de noms :

- Vous pouvez attribuer vous-mêmes des noms dans SCL en suivant la règle des IDENTIFICATEURS, représentée à la figure A-8. Il s'agit de la règle générale applicable aux noms dans SCL.
- Vous avez en outre la possibilité d'importer des noms à l'aide de la table des mnémoniques dans STEP 7. Les noms suivent alors la règle des IDENTIFICATEURS ou celle des mnémoniques. Le mnémonique doit figurer entre apostrophes et tous les caractères imprimables y compris les espaces sont autorisés.

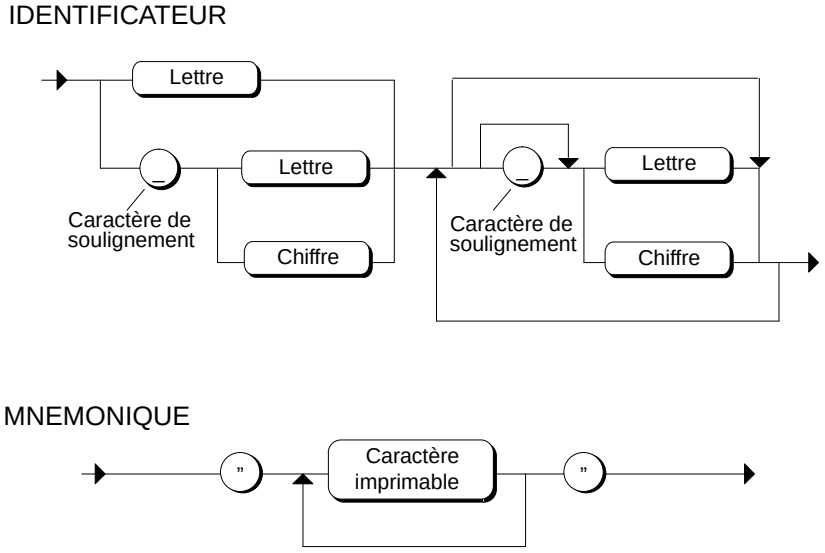


Figure A-8 *Règles lexicales : IDENTIFICATEUR et mnémonique*

Règles pour l'attribution du nom

Règles dont vous devez tenir compte :

- Lorsque vous attribuez un nom, choisissez le de sorte à ce qu'il soit univoque et évocateur. Vous contribuerez ainsi à une meilleure lisibilité du programme.
- Vérifiez si le nom a déjà été attribué dans le système, par exemple à un identificateur pour un type de données ou à des fonctions standard.
- Le domaine de validité des noms autorisés globalement s'étend sur l'ensemble du programme. Pour un nom autorisé localement, le domaine de validité se limite à un bloc. Vous pouvez donc utiliser les même noms dans des blocs différents. Le tableau A-12 résume les possibilités qui s'offrent à vous.

Restrictions

Lorsque vous attribuez un nom, vous devez tenir compte des restrictions suivantes :

Dans leur domaine de validité, les noms doivent être univoques, ce qui signifie que vous ne n'êtes pas autorisés à attribuer une nouvelle fois un nom qui l'a déjà été dans le même bloc. En outre, vous ne pouvez pas attribuer les noms suivants, déjà utilisés dans le système :

- Noms de mots-clés : p. ex. `CONST, END_CONST, BEGIN`
- Noms d'opérateurs : p. ex. `AND, XOR`
- Noms d'identificateurs prédéfinis : p. ex. les noms pour les types de données comme `BOOL, STRING, INT`
- Noms des constantes prédéfinies `TRUE` et `FALSE`
- Noms des fonctions standard : p. ex. `ABS, ACOS, ASIN, COS, LN`
- Noms d'identificateurs d'opérandes ou d'opérandes absolus pour des données système globales: p. ex. `EB, EW, ED, AB, AW, AD MB, MD`

Utilisation de la règle IDENTIFICATEURS

Le tableau A-12 précise dans quels cas vous pouvez attribuer des noms en utilisant la règle des IDENTIFICATEURS.

Tableau A-12 Utilisation de la règle IDENTIFICATEURS

IDENTIFICATEUR	Description	Règle
Nom de bloc	Nom symbolique d'un bloc	DESIGNATION DE BLOC, appel de fonction
Nom de temporisation et de compteur	Nom symbolique d'une temporisation et d'un compteur	DESIGNATION DE TEMPORISATION, DESIGNATION DE COMPTEUR
Nom d'attribut	Nom d'un attribut	Affectation d'attribut
Nom de constante	Déclaration d'une constante symbolique, utilisation	Section des constantes, constante
Repère de saut	Déclaration d'un repère de saut, utilisation d'un repère de saut	Section de repères de saut, section des instructions, instruction GOTO
Nom de variable	Déclaration d'une variable temporaire ou statique	Déclaration de variable, variable simple, variable structurée
Nom d'instance locale	Déclaration d'une instance locale	Déclaration de l'instance, nom d'appel de FB

DESIGNATION DE BLOC

Dans la règle de DISIGNATION DE BLOC, vous pouvez utiliser alternativement un IDENTIFICATEUR ou un mnémonique :

DESIGNATION DE BLOC

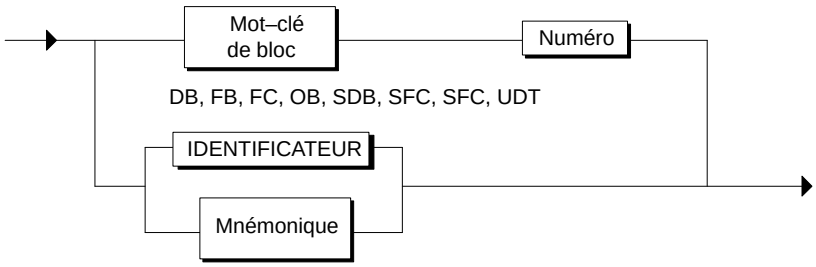


Figure A-9 *Règle lexicale : DESIGNATION DE BLOC*

Il en est de même des règles de DESIGNATION DE TEMPORISATION et de DESIGNATION DE COMPTEUR, qui sont similaires à la règle de DESIGNATION DE BLOC.

A.11 Constantes et drapeaux prédéfinis

Constantes et drapeaux prédéfinis

Les deux tableaux sont les mêmes pour les abréviations SIMATIC et CEI.

Tableau A-13 Constantes prédéfinies

Abréviation	Description
FALSE	Constante booléenne (standard) prédéfinie, de valeur 0. En logique booléenne, elle signifie qu'une condition n'est pas remplie.
TRUE	Constante booléenne (standard) prédéfinie, de valeur 1. En logique booléenne, elle signifie qu'une condition est remplie.

Tableau A-14 Drapeaux

Abréviation	Description
EN	Drapeau de validation d'un bloc
ENO	Drapeau d'erreur d'un bloc
OK	Le drapeau affiche FALSE lorsqu'une instruction n'a pas été traitée correctement.

Règles lexicales

B

Structure de cette annexe

Règles lexicales

Les règles lexicales décrivent la structure des éléments (symboles) sur lesquels porte l'analyse lexicale réalisée par le compilateur. Le format de la notation n'est pas libre, ce qui signifie que les règles doivent être appliquées scrupuleusement et en particulier que :

- l'insertion de caractères de mise en forme n'est pas autorisée.
- l'insertion de blocs et de lignes de commentaires n'est pas possible.
- l'insertion d'attributs pour les identificateurs n'est pas possible.

Répartition

Les règles lexicales sont divisées en trois groupes :

- Identificateurs
- Constantes littérales
- Adressage absolu

B.1 Identificateurs

Tableau B-1 Identificateurs

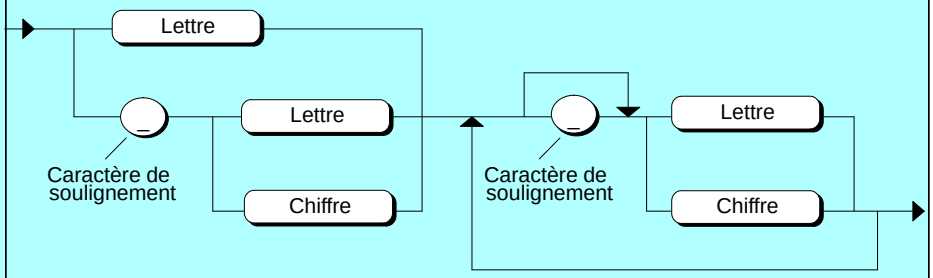
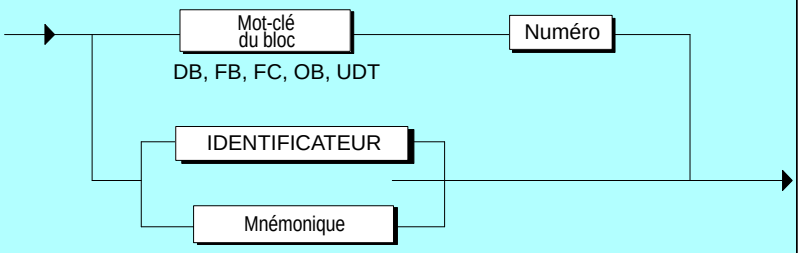
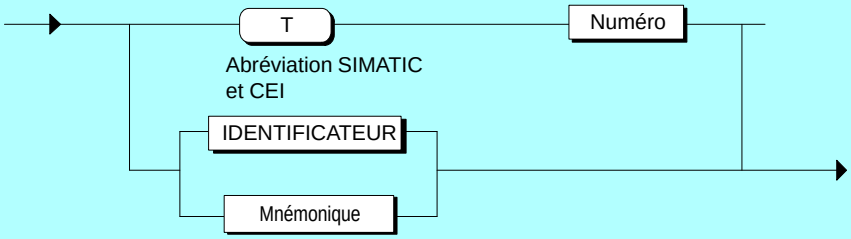
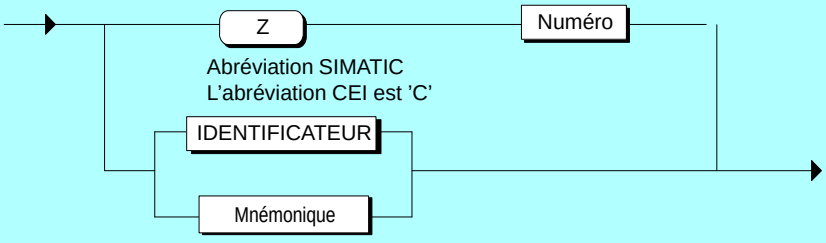
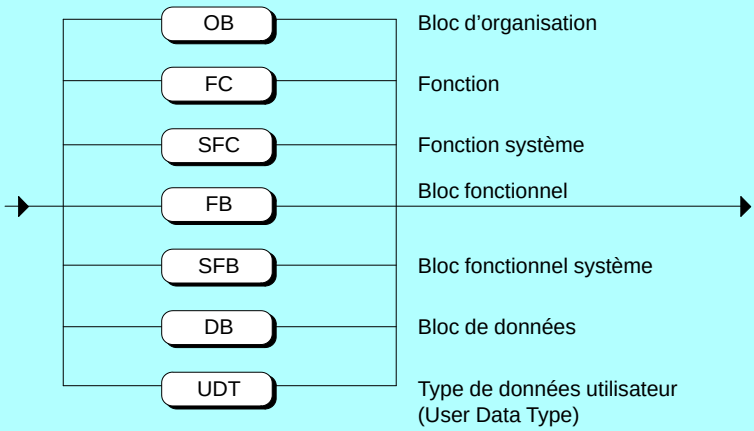
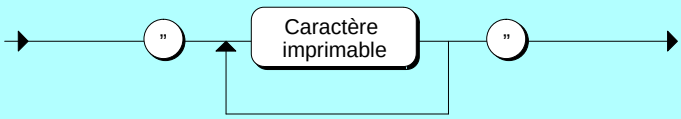
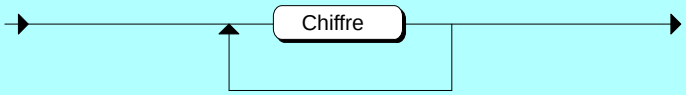
Règle	Diagramme syntaxique
IDENTIFICATEUR	
DESIGNATION DE BLOC	Cette règle s'applique également au nom de règles suivants : DESIGNATION DU DB DESIGNATION DU FB DESIGNATION DE LA FC DESIGNATION DE L'OB DESIGNATION DE L'UDT 
DESIGNATION DE TEMPORISATION	
DESIGNATION DE COMPTEUR	

Tableau B-1 Identificateurs (suite)

Règle	Diagramme syntaxique
Mot-clé de bloc	
Mnémonique	
Numéro	

B.1.1 Constantes littérales

Tableau B-2 Constantes littérales

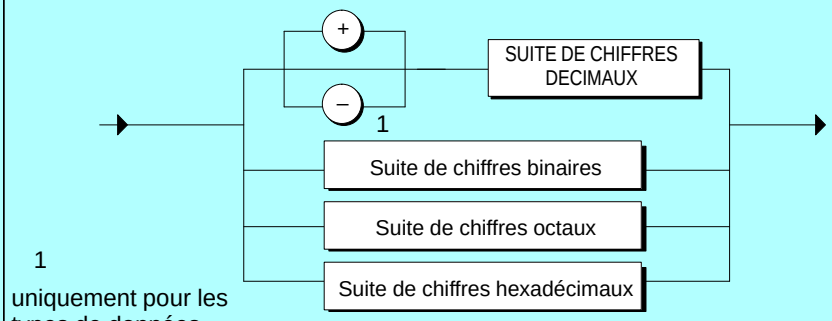
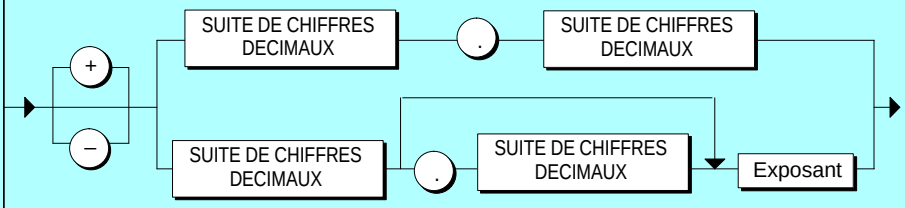
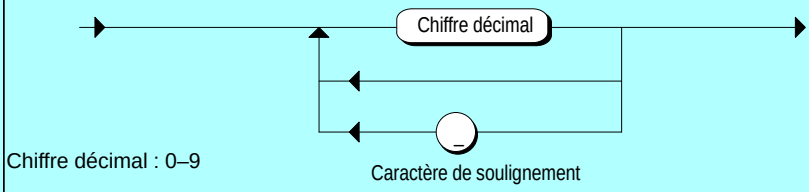
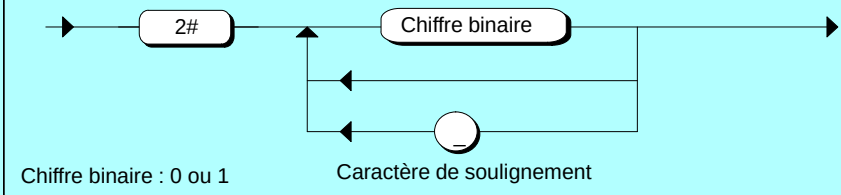
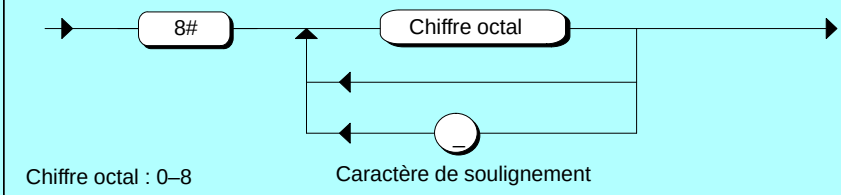
Règle	Diagramme syntaxique
CONSTANTE LITTERALE ENTIERE	
CONSTANTE LITTERALE REELLE	
SUITE DE CHIFFRES DECIMAUX	
SUITE DE CHIFFRES BINAIRES	
SUITE DE CHIFFRES OCTAUX	

Tableau B-2 Constantes littérales (suite)

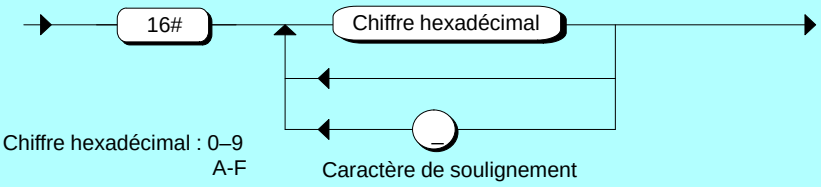
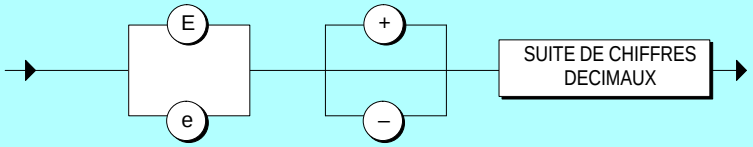
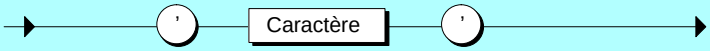
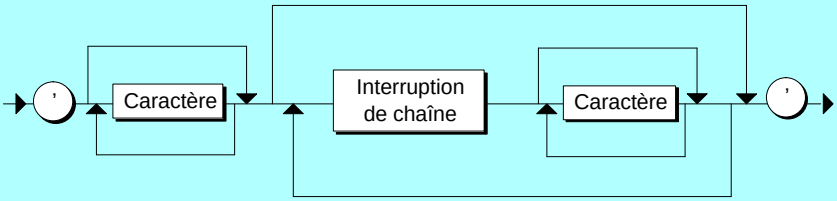
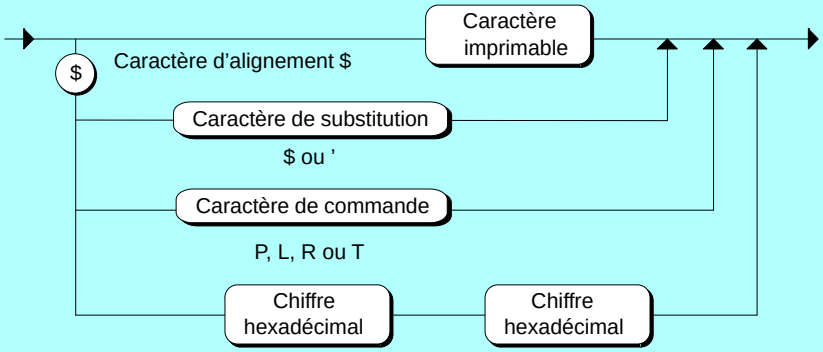
Règle	Diagramme syntaxique
SUITE DE CHIFFRES HEXADECIMAUX	
Exposant	
Constante littérale de type caractère	
Constante littérale de type chaîne de caractères	
Caractère	

Tableau B-2 Constantes littérales (suite)

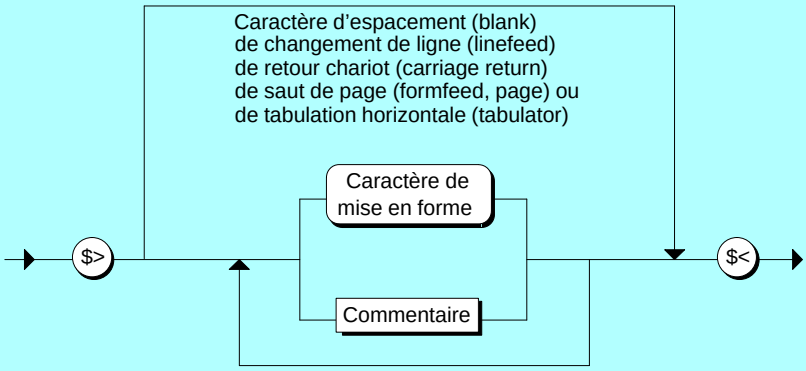
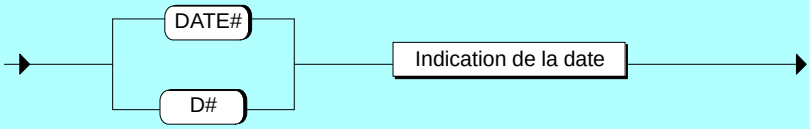
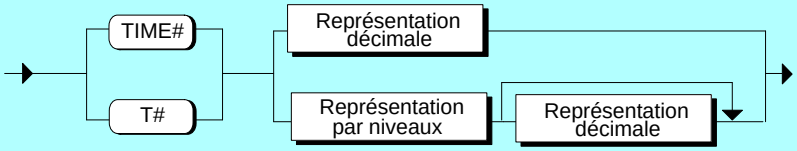
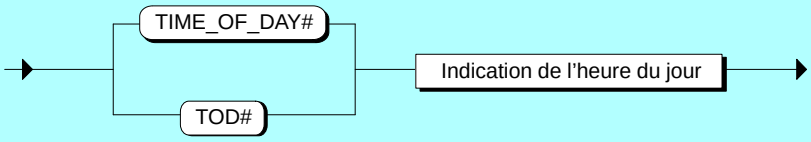
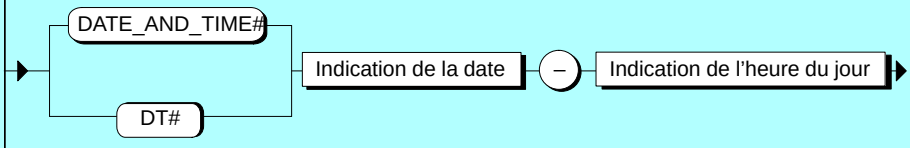
Règle	Diagramme syntaxique
Interruption de chaîne	
DATE	
DUREE	 – chaque unité de temps (p. ex. heures, minutes) ne doit être indiquée qu'une seule fois – l'ordre – jours, heures, minutes, secondes, millisecondes – doit être respecté
HEURE DE JOUR	
DATE ET HEURE	

Tableau B-2 Constantes littérales (suite)

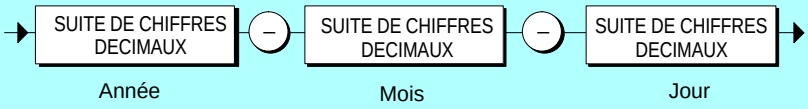
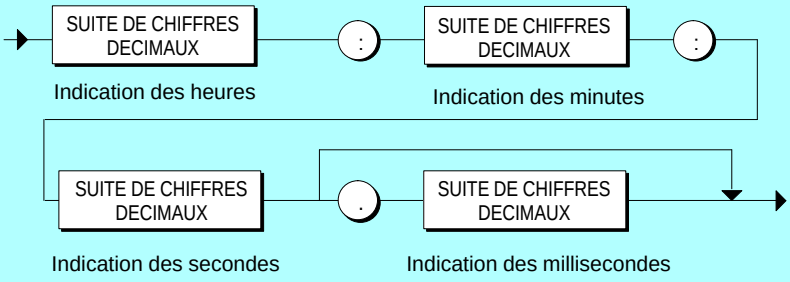
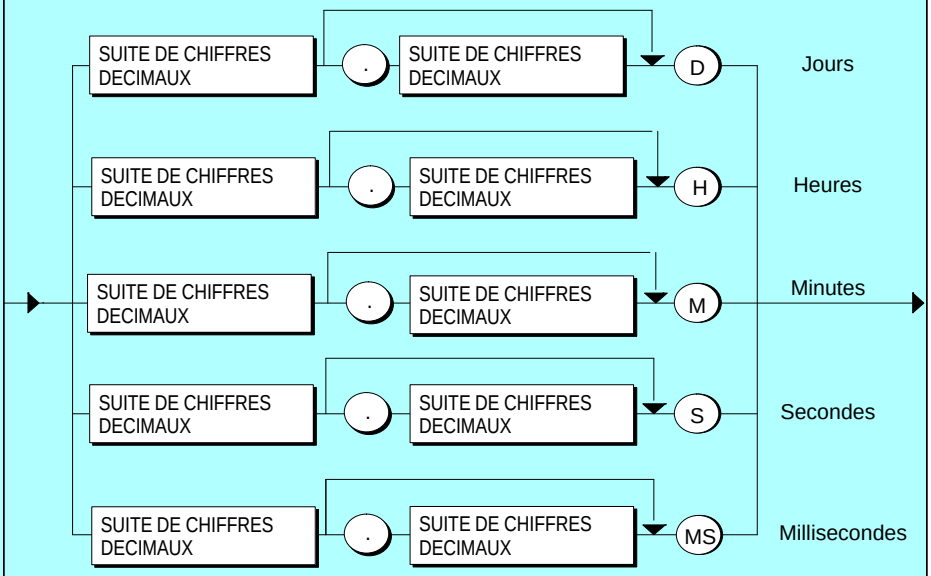
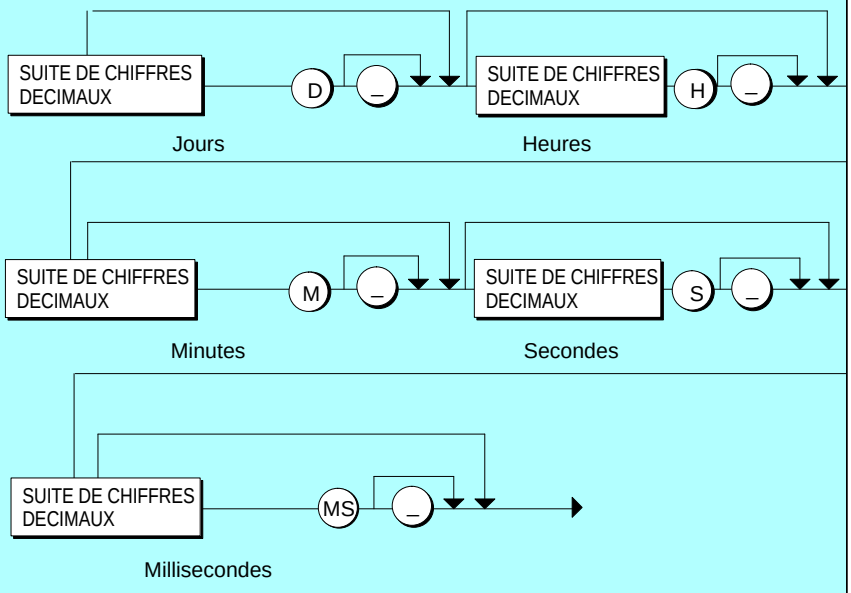
Règle	Diagramme syntaxique
Indication de la date	
Indication de l'heure du jour	

Tableau B-2 Constantes littérales (suite)

Règle	Diagramme syntaxique
Représentation décimale	 Pour passer à la représentation décimale, il faut que les unités temporelles n'aient pas encore été définies
Représentation par niveaux	 au minimum une indication est requise

B.1.2 Adressage absolu

Tableau B-3 Adressage absolu

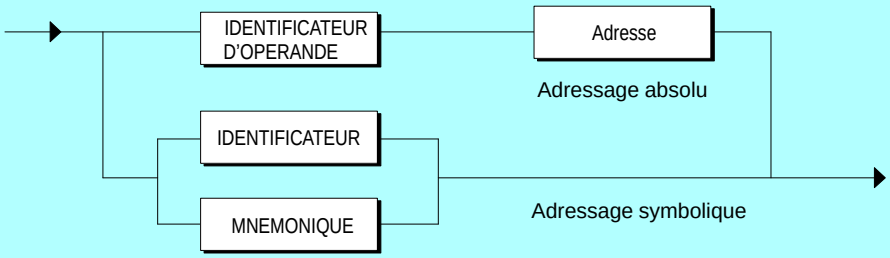
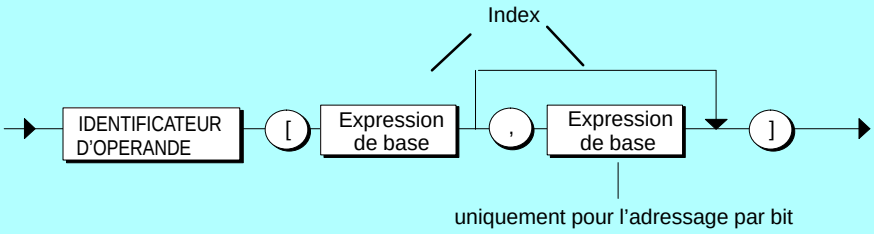
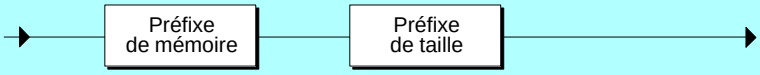
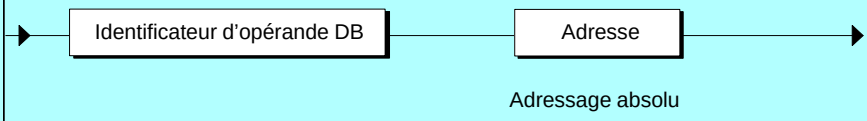
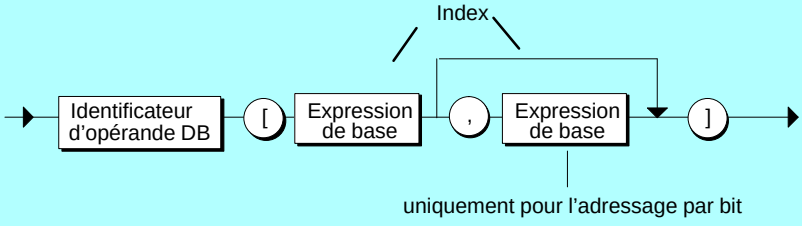
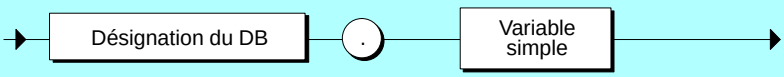
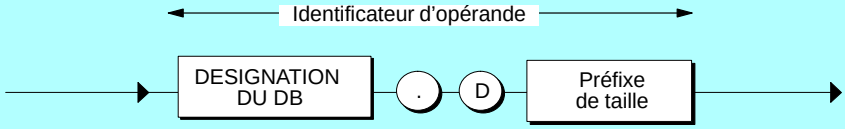
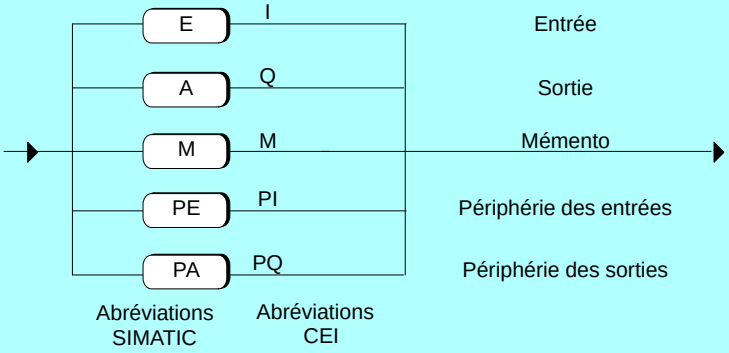
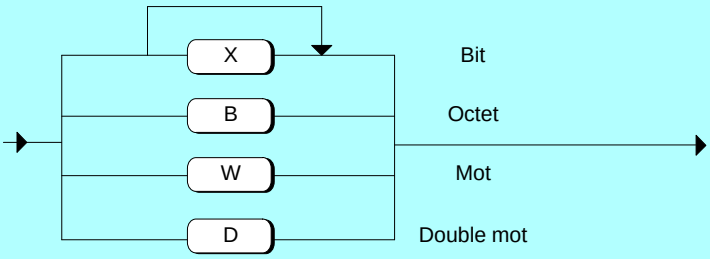
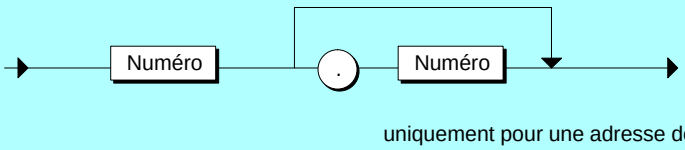
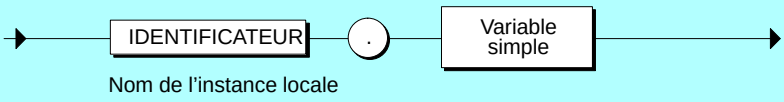
Règle	Diagramme syntaxique
ADRESSAGE DE MEMOIRE SIMPLE	
ADRESSAGE DE MEMOIRE INDEXE	
IDENTIFICATEUR D'OPERANDE POUR MEMOIRE	
ADRESSAGE ABSOLU DE DB	
ADRESSAGE INDEXE DE DB	
ADRESSAGE STRUCTURE DE DB	

Tableau B-3 Adressage absolu (suite)

Règle	Diagramme syntaxique
Identificateur d'opérande DB	
Préfixe de mémoire	
Préfixe de taille pour la mémoire et le DB	
Adresse pour la mémoire et le DB	
Adressage de l'instance locale	

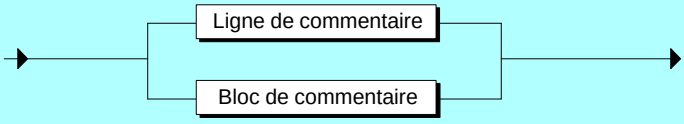
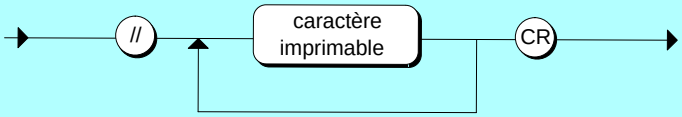
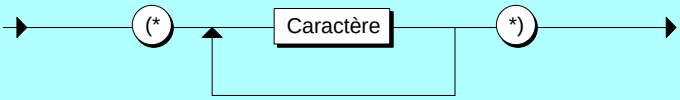
B.2 Commentaires

Important

Voici les restrictions dont vous devez tenir compte lorsque vous insérer des commentaires :

- l'imbrication de commentaires n'est pas autorisée
- l'insertion de commentaires est possible à un endroit quelconque de la règle syntaxique, mais ne l'est pas dans une règle lexicale.

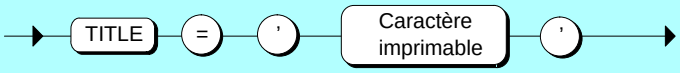
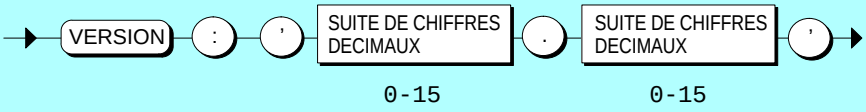
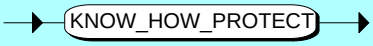
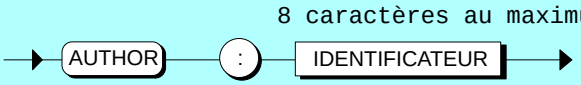
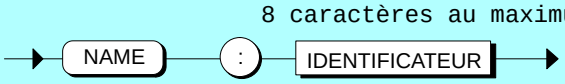
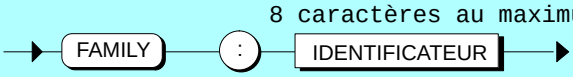
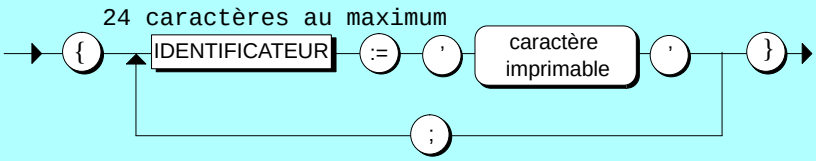
Tableau B-4 Commentaires

Règle	Diagramme syntaxique
COMMENTAIRE	
LIGNE DE COMMENTAIRE	
BLOC DE COMMENTAIRE	

B.3 Attributs de blocs

Important Les attributs de blocs peuvent suivre la DISGNATION DU BLOC ou précéder la déclaration de la première section de variables ou de paramètres.

Tableau B-5 Attributs

Règle	Diagramme syntaxique
TITRE	
VERSION	
PROTECTION DE BLOC	
AUTEUR	
NOM	
FAMILLE DE BLOC	
Attributs système de blocs	

Règles syntaxiques

C

Définition

Les règles syntaxiques définissent la structure de SCL, en s'appuyant sur les règles lexicales. Vous avez la possibilité de créer votre programme SCL dans un format libre, dans le respect de ces règles.

Structure de cette annexe

Conventions

Chacune des règles est précédée de son nom. Si elle est appliquée dans une règle de niveau hiérarchique supérieur, son nom figure dans un rectangle.

Si le nom de la règle est écrit en majuscules, vous devez vous reporter aux règles lexicales.

Les informations figurant dans un rectangle aux bouts arrondis ou dans un cercle sont décrites en annexe A.

Important

La liberté dans le format signifie que :

- l'insertion de caractères de mise en forme est partout possible.
- l'insertion de lignes ou de blocs de commentaires est autorisée (voir paragraphe 7.6).

C.1 Organisation des sources SCL

Tableau C-1 Syntaxe des sources SCL

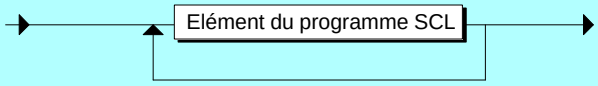
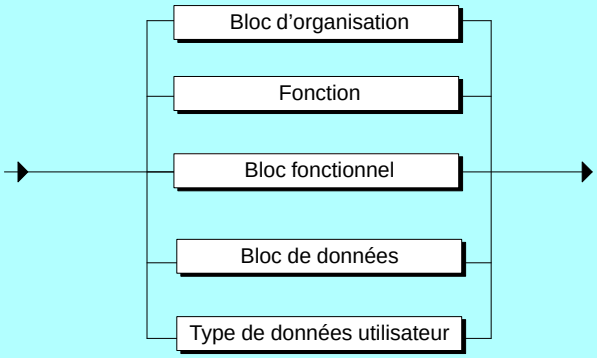
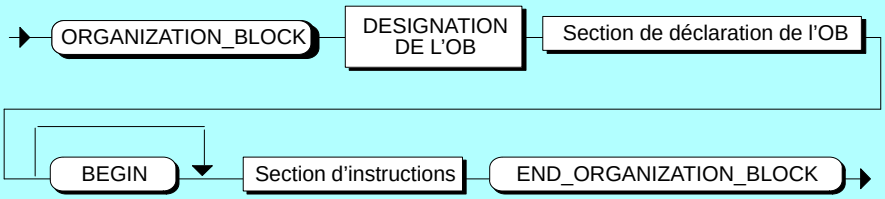
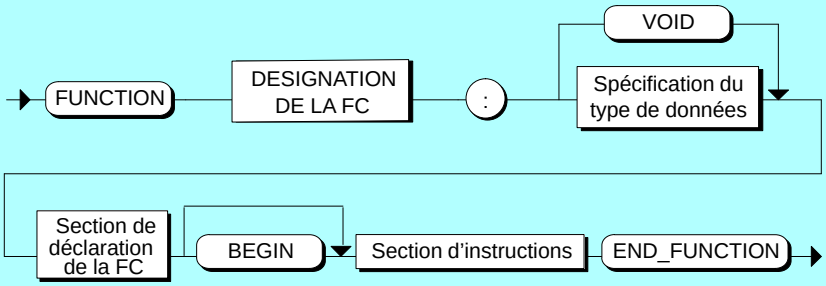
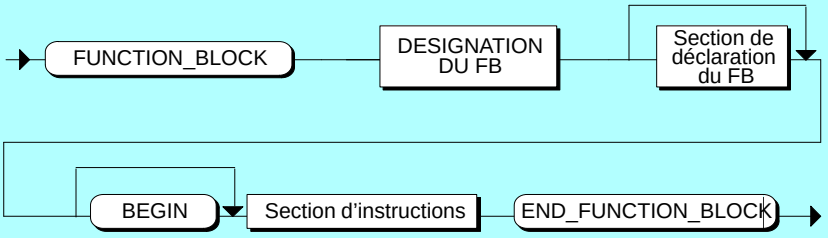
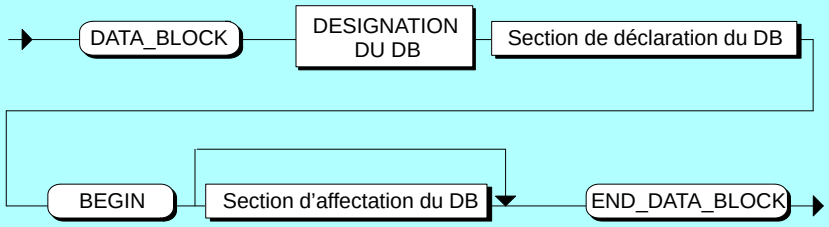
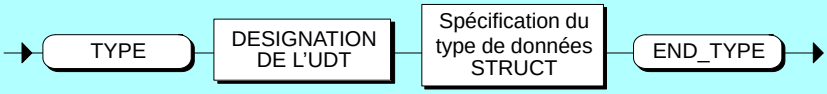
Règle	Diagramme syntaxique
Programme SCL	
Elément du programme SCL	
Bloc d'organisation	
Fonction N'oubliez pas que pour les fonctions dépourvues de l'attribut VOID, la valeur en retour doit être affectée au nom de la fonction dans la section des instructions !	
Bloc fonctionnel	

Tableau C-1 Syntaxe des sources SCL (suite)

Règle	Diagramme syntaxique
Bloc de données	
Type de données utilisateur	

C.2 Structure des sections de déclaration

Tableau C-2 Syntaxe de la section de déclaration

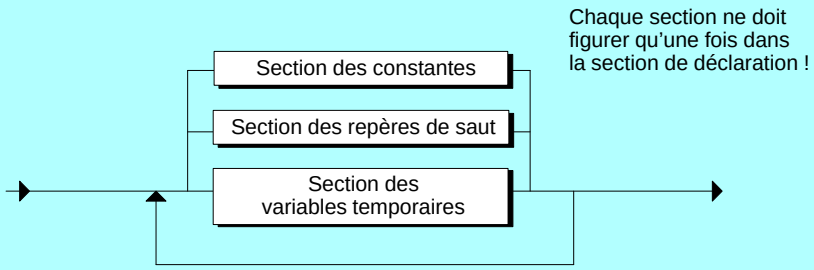
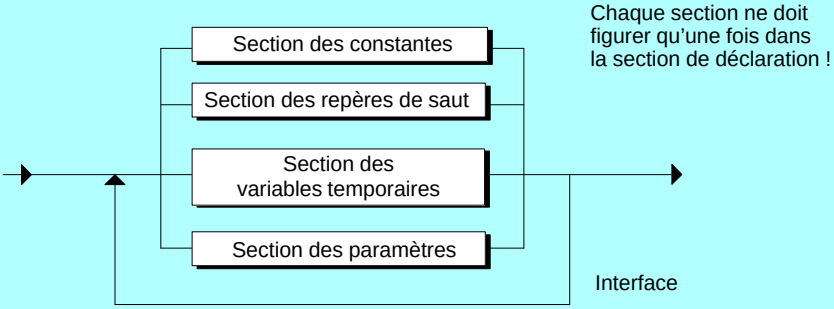
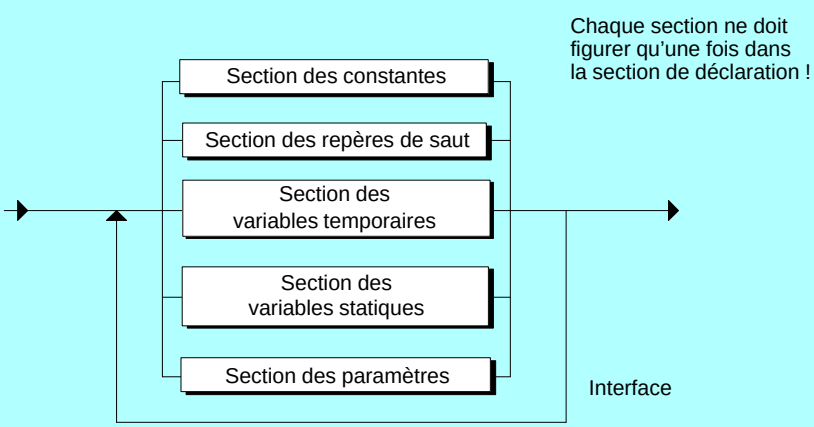
Règle	Diagramme syntaxique
Section de déclaration de l'OB	
Section de déclaration de la FC	
Section de déclaration du FB	
Section de déclaration du DB	

Tableau C-3 Syntaxe des sections de déclaration

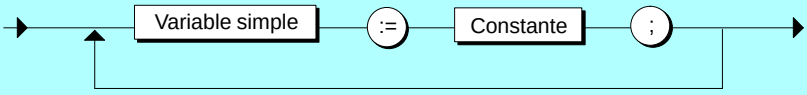
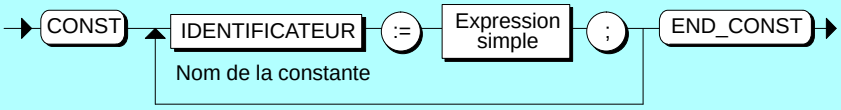
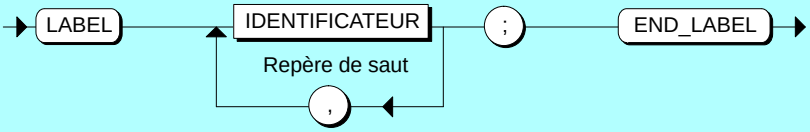
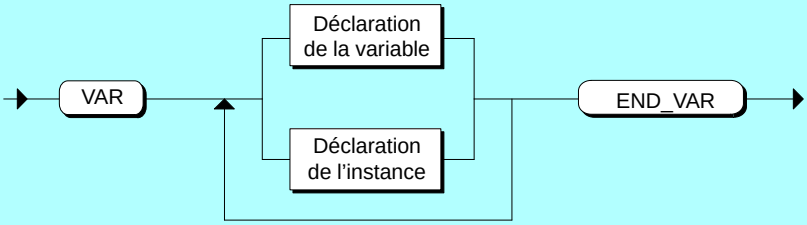
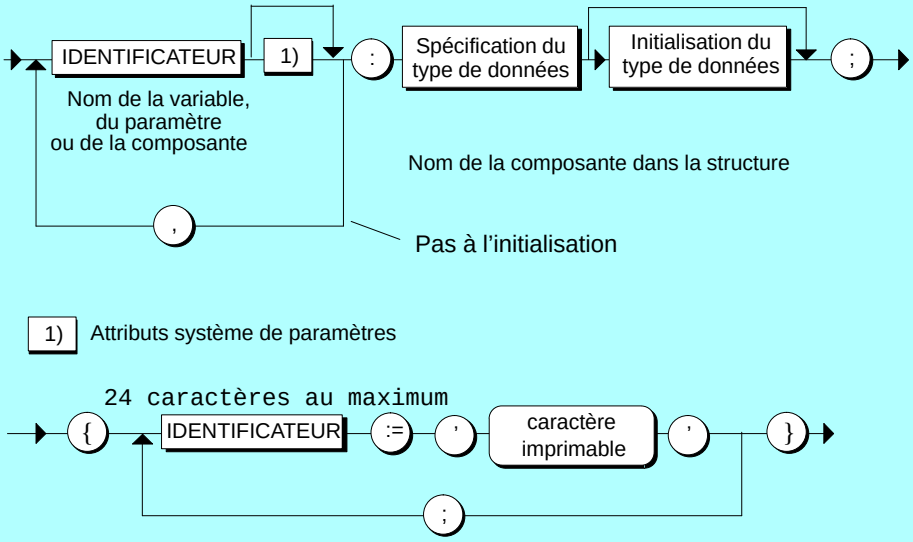
Règle	Diagramme syntaxique
Section d'affectation du DB	
Section des constantes	
Section des repères de saut	
Section des variables statiques	
Déclaration de variables	

Tableau C-3 Syntaxe des sections de déclaration (suite)

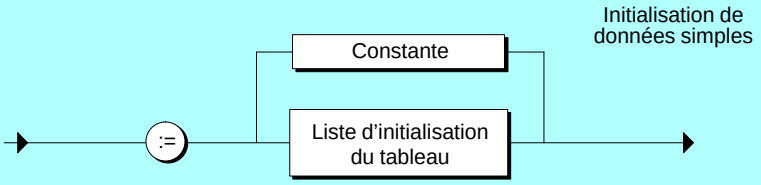
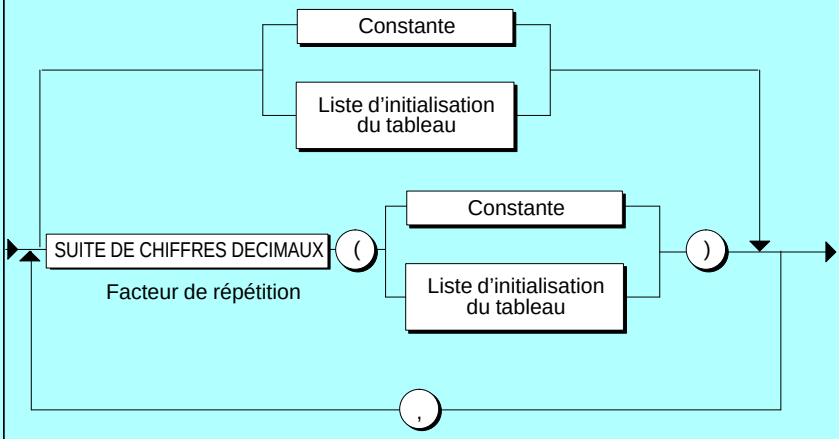
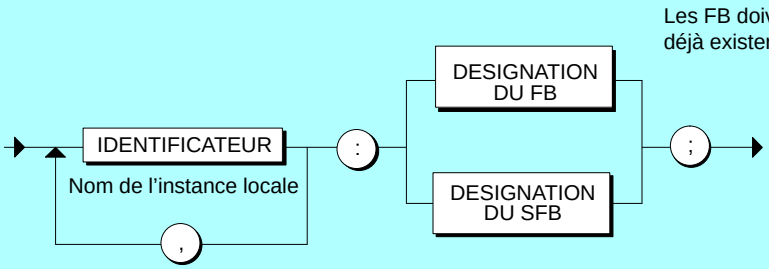
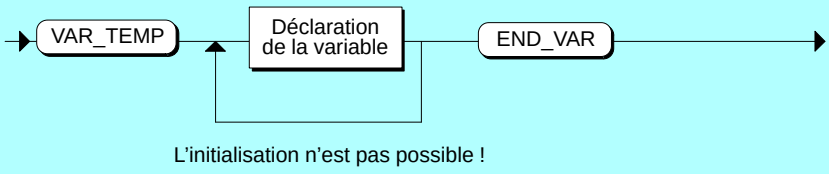
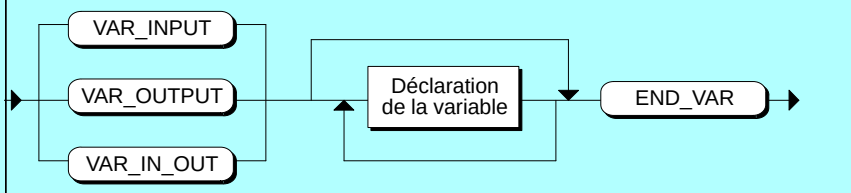
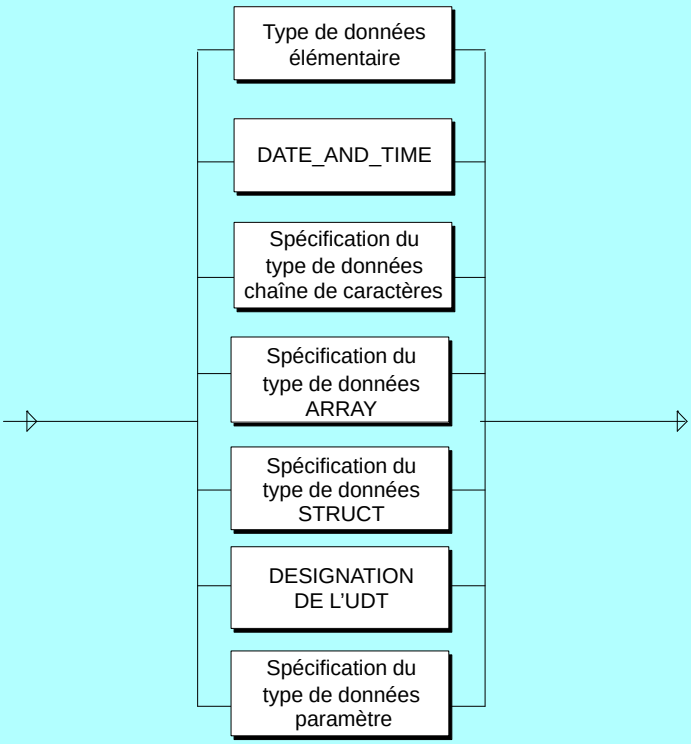
Règle	Diagramme syntaxique
Initialisation du type de données	
Liste d'initialisation du tableau	
Déclaration de l'instance	
Section des variables temporaires	

Tableau C-3 Syntaxe des sections de déclaration (suite)

Règle	Diagramme syntaxique
Section des paramètres	 L'initialisation n'est possible que pour VAR_INPUT et VAR_OUTPUT !
Spécification du type de données	

C.3 Types de données dans SCL

Tableau C-4 Syntaxe des types de données dans la section de déclaration

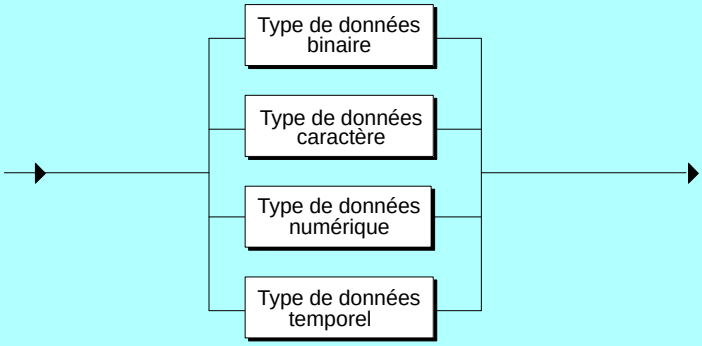
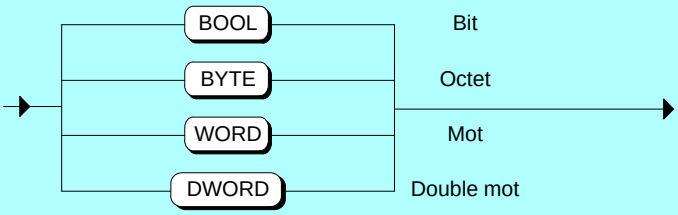
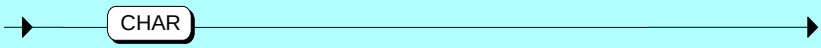
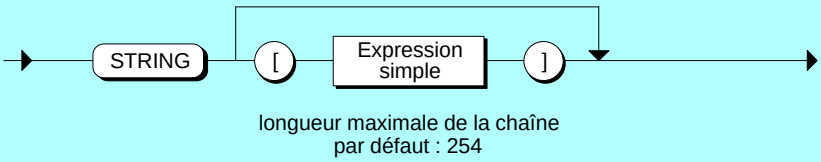
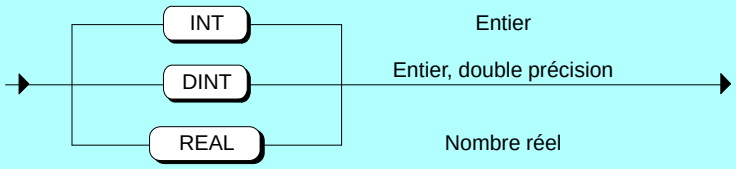
Règle	Diagramme de syntaxe
Type de données simple	
Type de données binaire	
Type de données caractère	
Spécification du type de données chaîne de caractères	
Type de données numérique	

Tableau C-4 Syntaxe des types de données dans la section de déclaration (suite)

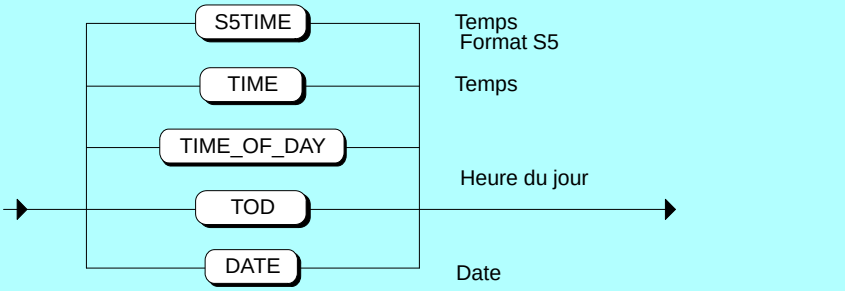
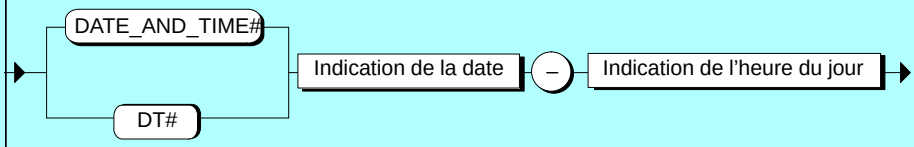
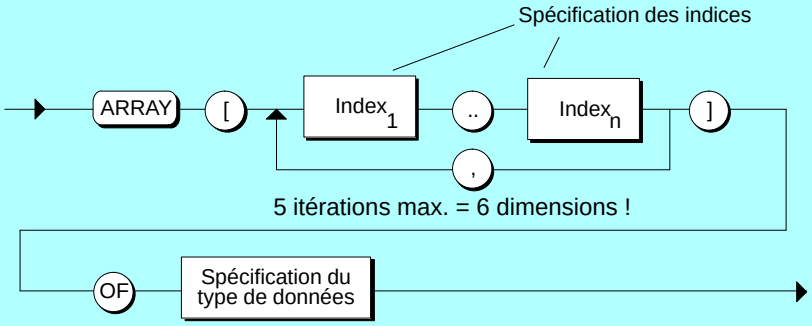
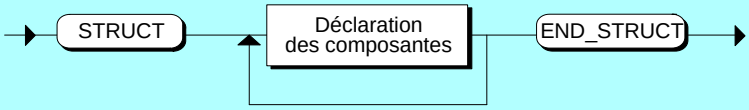
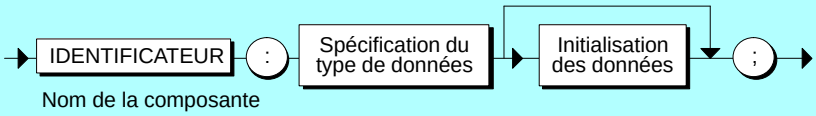
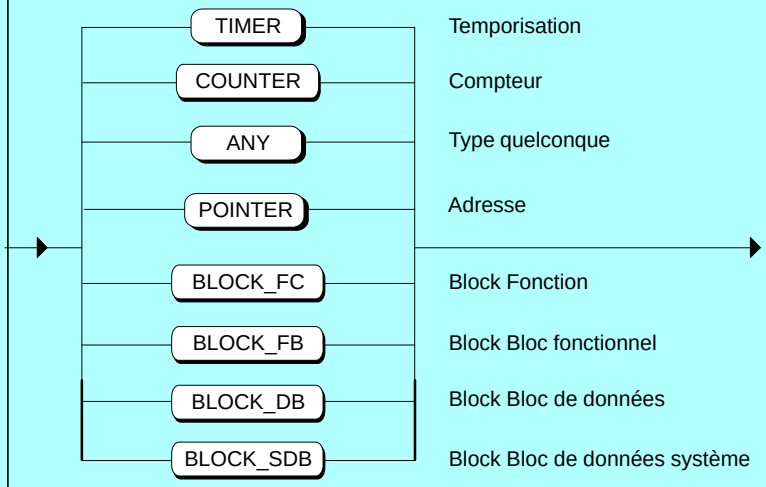
Règle	Diagramme de syntaxe
Type de temporisation	 Voir aussi le paragraphe B.1.1
DATE_ET_TIME	
Spécification du type de données ARRAY	
Spécification de type de données STRUCT N'oubliez pas de terminer le mot-clé END_STRUCT avec un point-virgule !	

Tableau C-4 Syntaxe des types de données dans la section de déclaration (suite)

Règle	Diagramme de syntaxe
Déclaration des composantes	
Spécification du type de données du paramètres	

C.4 Section des instructions

Tableau C-5 Syntaxe de la section des instructions

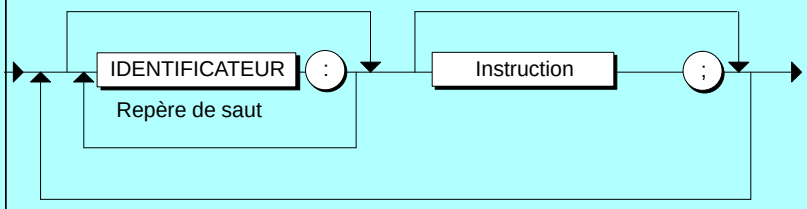
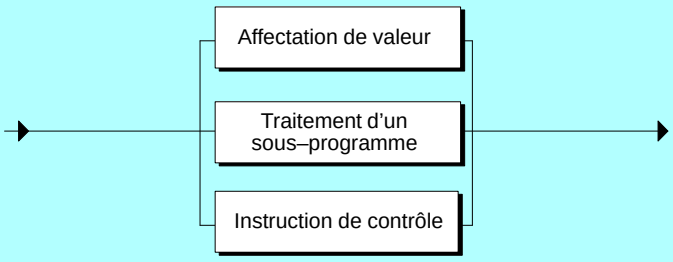
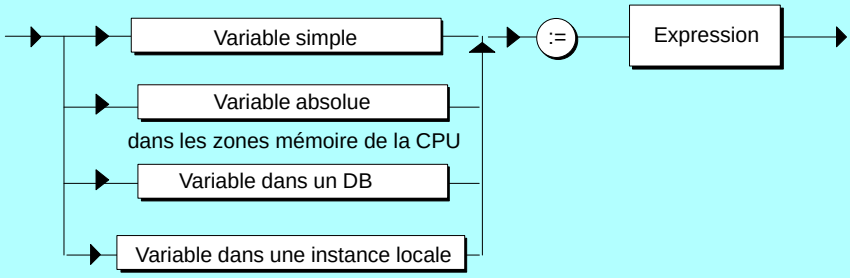
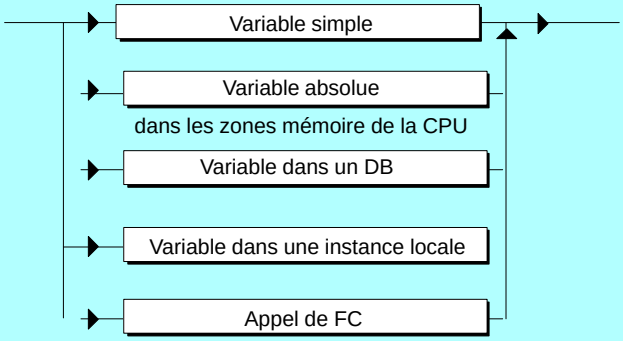
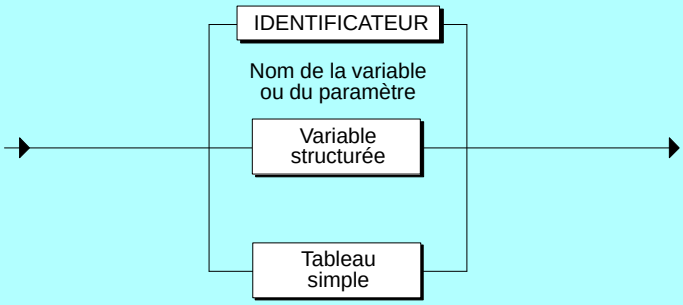
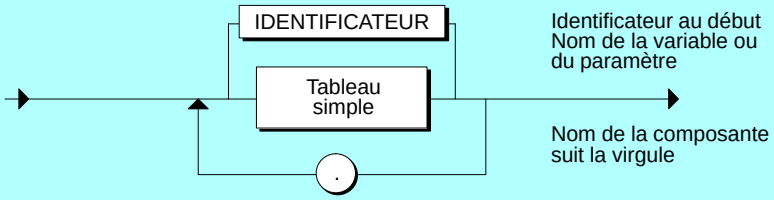
Règle	Diagramme syntaxique
Section des instructions	
Instruction	
Affectation de valeur	
Variable étendue	

Tableau C-5 Syntaxe de la section des instructions (suite)

Règle	Diagramme syntaxique
Variable simple	
Variable structurée	

C.5 Affectation de valeurs

Tableau C-6 Syntaxe de l'affectation de valeurs

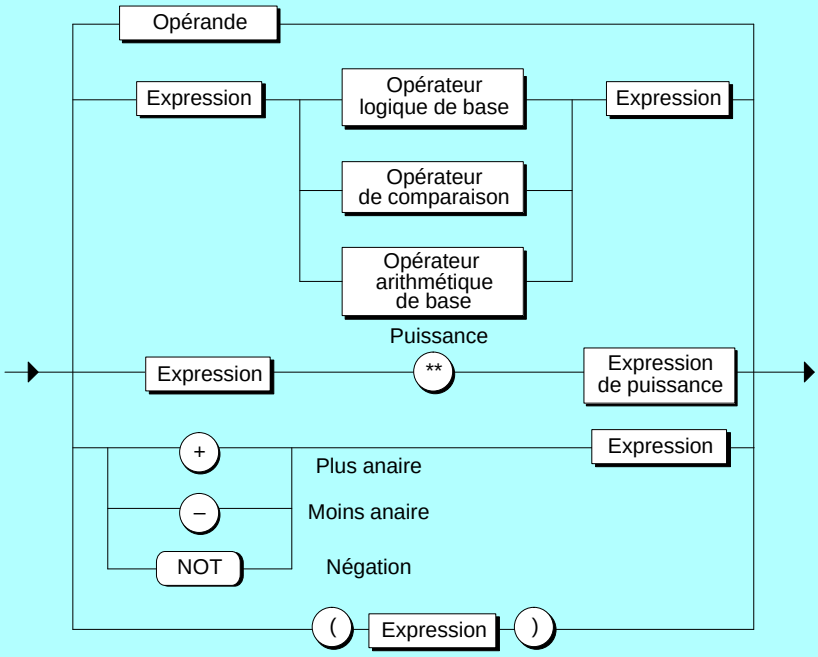
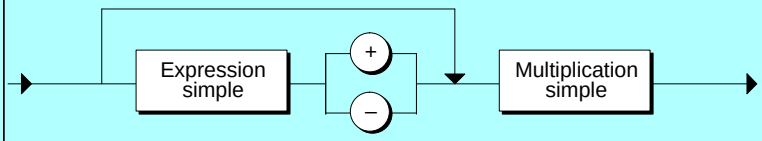
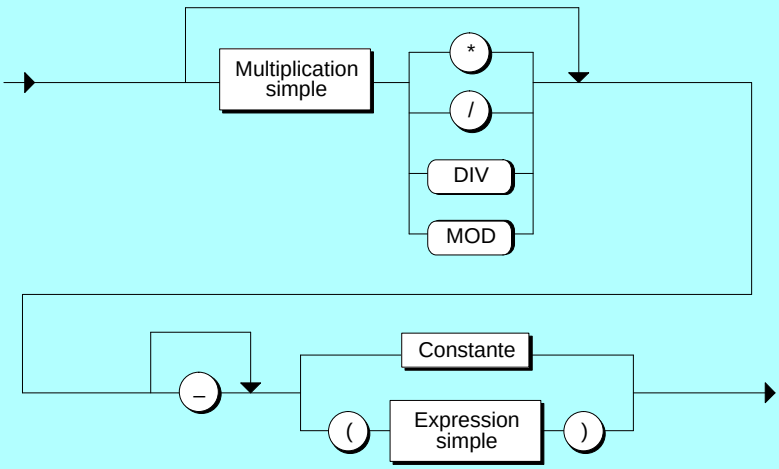
Règle	Diagramme syntaxique
Expression	
Expression simple	
Multiplication simple	

Tableau C-6 Syntaxe de l'affectation de valeurs (suite)

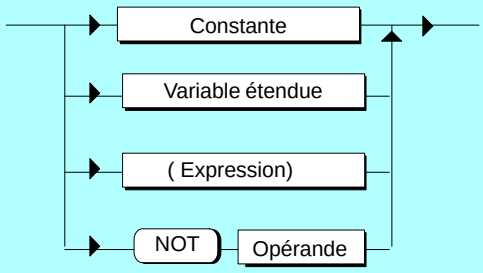
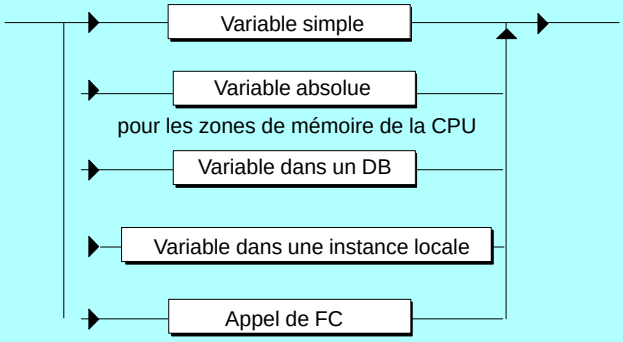
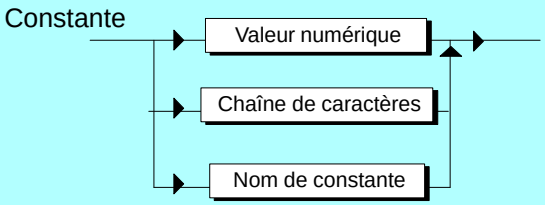
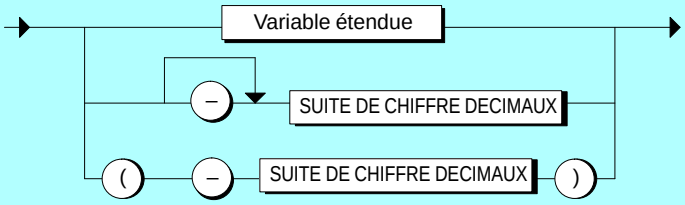
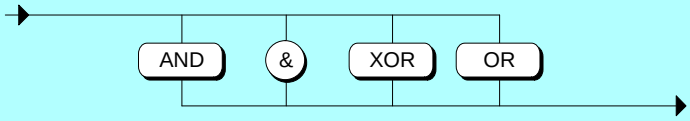
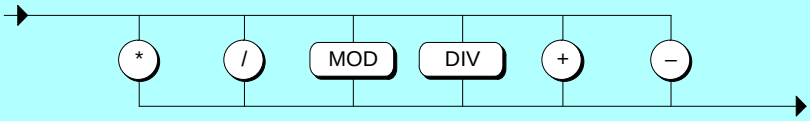
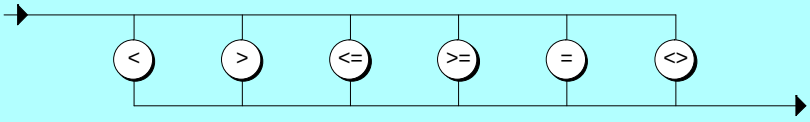
Règle	Diagramme syntaxique
Opérande	
Variable étendue	
Constante	
Exposant	
Opérateur logique de base	

Tableau C-6 Syntaxe de l’affectation de valeurs (suite)

Règle	Diagramme syntaxique
Opérateur arithmétique de base	
Opérateur de comparaison	

C.6 Appel de fonctions et de blocs fonctionnels

Tableau C-7 Syntaxe des appels

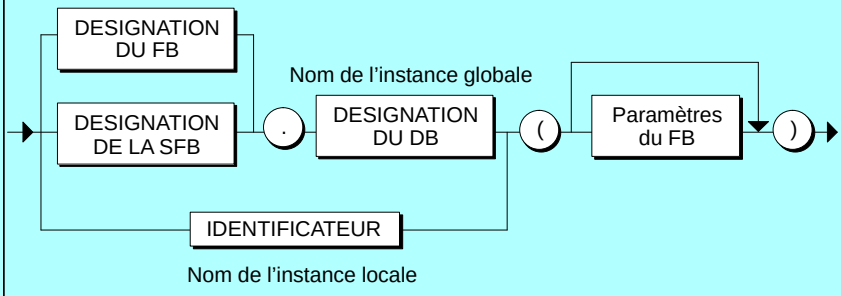
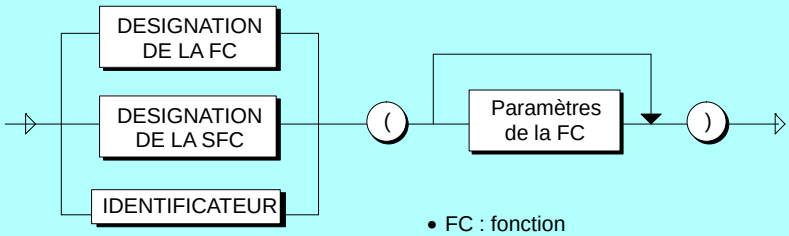
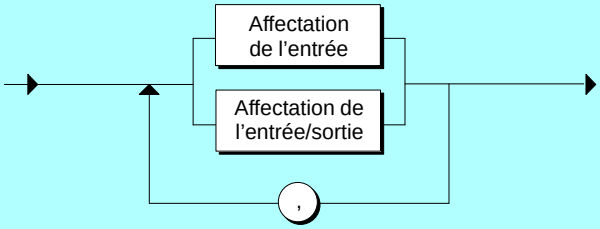
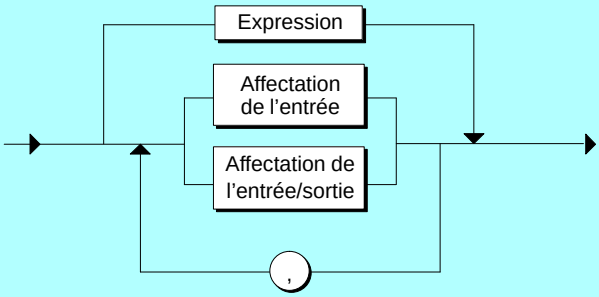
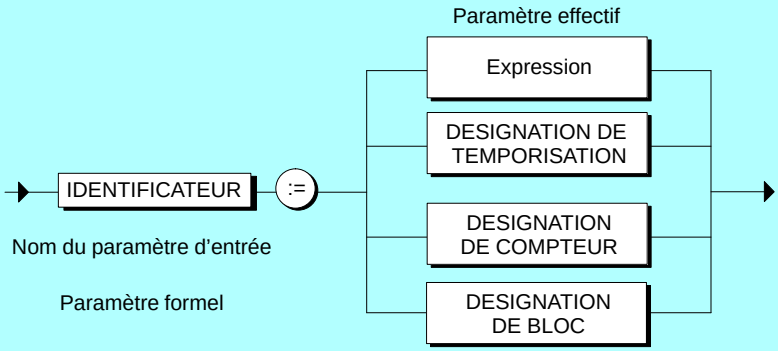
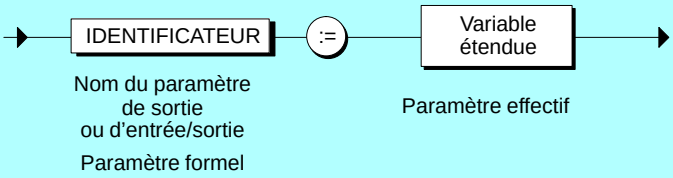
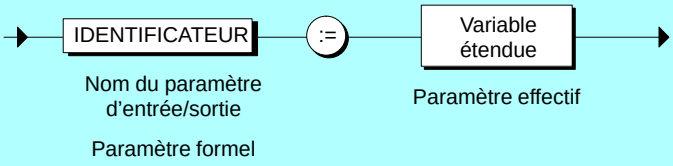
Règle	Diagramme syntaxique
Appel de FB	FB : bloc fonctionnel SFB: bloc fonctionnel système 
Appel de fonction	
Paramètres du FB	
Paramètres de la FC	

Tableau C-7 Syntaxe des appels (suite)

Règle	Diagramme syntaxique
Affectation de l'entrée	
Affectation de la sortie ou de l'entrée/sortie	
Affectation de l'entrée/ sortie	

C.7 Instructions de contrôle

Tableau C-8 Syntaxe des instructions de contrôle

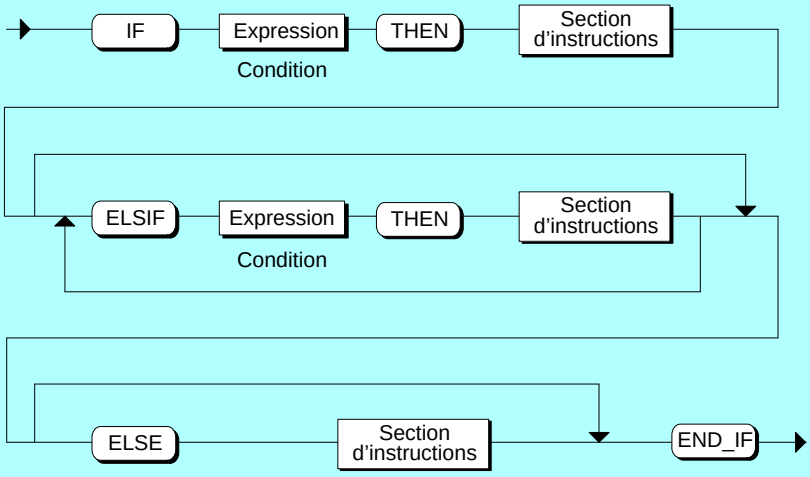
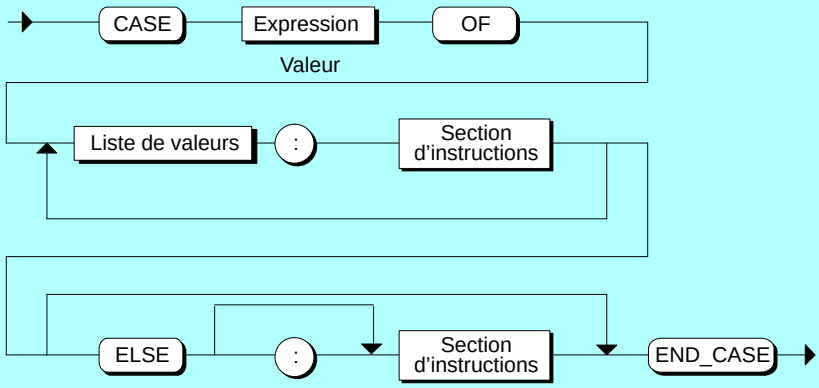
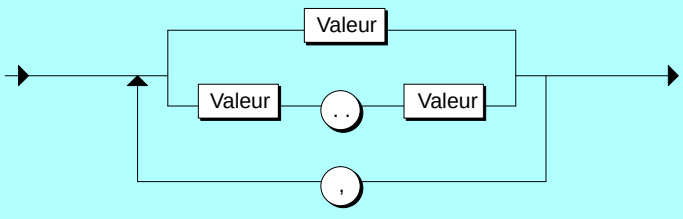
Règle	Diagramme syntaxique
Instruction IF N'oubliez pas de terminer le mot-clé END_IF avec un point-virgule !	
Instruction Case N'oubliez pas de terminer le mot-clé END_CASE avec un point-virgule !	
Liste de valeurs	

Tableau C-8 Syntaxe des instructions de contrôle (suite)

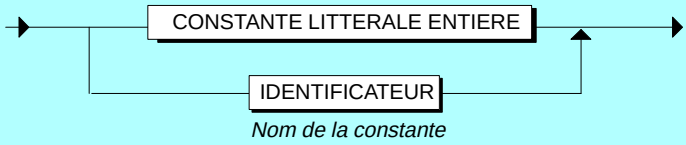
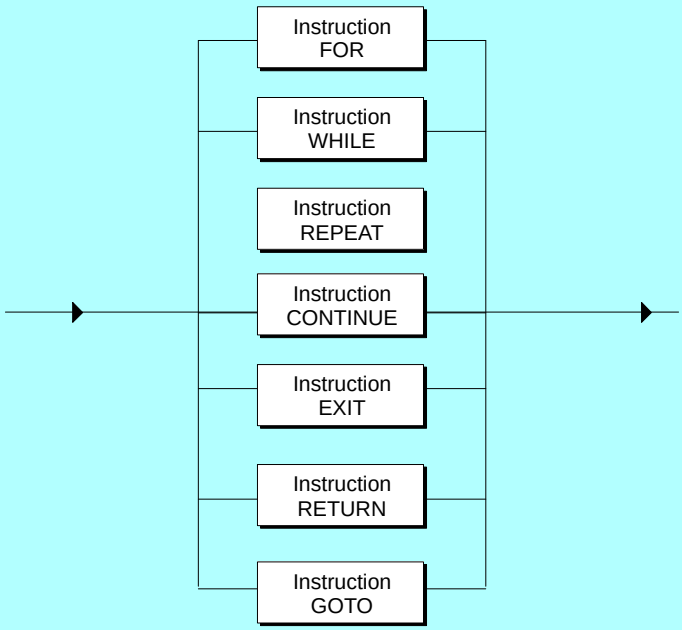
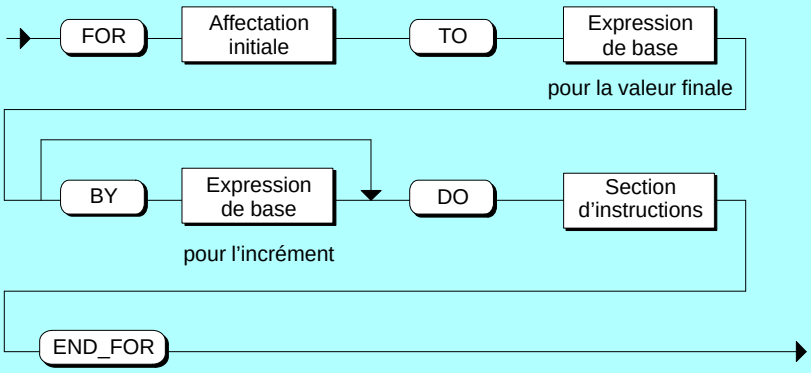
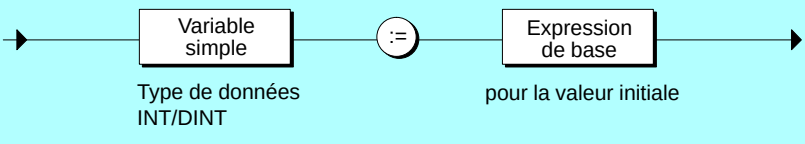
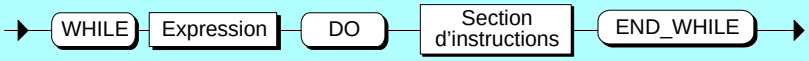
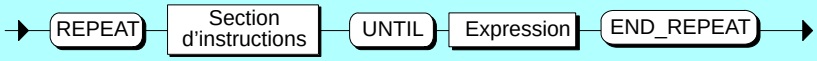
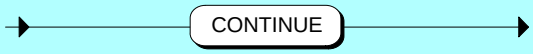
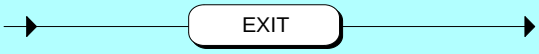
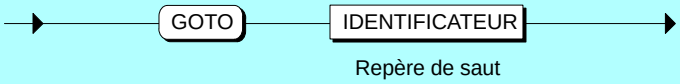
Règle	Diagramme syntaxique
Valeur	
Instructions d'itération et de saut	
Instruction FOR N'oubliez pas de terminer le mot-clé END_FOR avec un point-virgule !	
Affectation initiale	

Tableau C-8 Syntaxe des instructions de contrôle (suite)

Règle	Diagramme syntaxique
Instruction WHILE N'oubliez pas de terminer le mot-clé END_WHILE avec un point-virgule !	
Instruction REPEAT N'oubliez pas de terminer le mot-clé END_REPEAT avec un point-virgule !	
Instruction CONTINUE	
Instruction RETURN	
Instruction EXIT	
Saut dans le programme	

Bibliographie

D

/12/ Guide : *Automate programmable S7-300,*
Configuration et application

/13/ Guide : *Automate programmable S7-400,*
Configuration et application

/14/ Guide : *Calculateur industriel M7-300/400,*
Configuration et application

/20/ Guide : *Automates programmables S7-300/400,*
Programmation

/25/ Guide : *Calculateur industriel M7,*
Programmation

/30/ Petit manuel illustré : *Faites connaissance avec le S7-300...*

/70/ Manuel : *Automate programmable S7-300,*
Installation et configuration - Caractéristiques des CPU

/71/ Manuel de référence : *Systèmes d'automatisation S7-300, M7-300,*
Caractéristiques des modules

/72/ Liste des opérations : *Automate programmable S7-300*

/100/ Manuel de mise en œuvre : *Systèmes d'automatisation S7-400, M7-400,*
Installation et configuration

/230/ Manuel : *Logiciel de base pour SIMATIC S7,*
Pour une conversion facile de S5 à S7...

/231/ Guide de l'utilisateur : *Logiciel de base pour SIMATIC S7 et M7,*
STEP 7

/232/ Manuel : *Langage LIST pour SIMATIC S7-300/400,*
Programmation de blocs

/233/ Manuel : *Langage CONT pour SIMATIC S7-300/400,*
Programmation de blocs

/234/ Manuel de programmation : *Logiciel système pour SIMATIC S7-300/400*
Conception de programmes

/235/ Manuel de référence : *Logiciel système pour SIMATIC S7-300/400*
Fonctions standard et fonctions système

/236/ Manuel: *Langage LOG pour SIMATIC S7-300/400,*
Programmation de blocs

/237/ *Index général, STEP 7*

/251/ Manuel: *GRAPH pour SIMATIC S7-300/400,*
Programmation de commandes séquentielles

/254/ Manuel: *CFC pour SIMATIC S7 et M7,*
Comment câbler graphiquement des fonctions technologiques ?

/800/ *DOCPRO,*
Documentation normalisée d'un projet (uniquement sur CD-ROM)

/801/ *Téléservice pour automates programmables de type S7,C7 et M7,*
(uniquement sur CD-ROM)

/802/ *Simulateur PLC de STEP 7,*
(uniquement sur CD-ROM)

/803/ Manuel de référence : *Logiciel système pour S7-300/400,*
STEP 7 Fonctions standard, 2ème partie (uniquement sur CD-ROM)

Glossaire

A

Abréviations

Les abréviations représentent les opérandes et les opérations de programmation de manière abrégée (E, par exemple, désigne l'entrée). STEP 7 reconnaît les abréviations CEI (exprimées en anglais) et les abréviations SIMATIC (exprimées d'après les opérations et conventions d'adressage SIMATIC allemandes).

Activer les points d'arrêt

Cette fonction permet de mettre l'unité centrale (CPU) en attente à des endroits définis du programme. Lorsqu'un point d'arrêt est atteint, vous pouvez réaliser les fonctions de test, comme par exemple le traitement incrémental d'instructions ou la visualisation de l'état des variables.

Adressage absolu

L'adressage absolu consiste à indiquer l'adresse de l'opérande à traiter. Exemple : l'adresse A 4.0 désigne le bit 0 dans l'octet 4 de la mémoire image des sorties.

Adressage symbolique

L'adressage symbolique consiste à indiquer de manière symbolique l'opérande à traiter (au lieu d'une adresse).

Aide en ligne

STEP 7 vous offre la possibilité d'afficher à l'écran des textes d'aide contextuels pendant votre travail avec le logiciel de programmation.

Appel de bloc

L'exécution des blocs est démarrée de la manière suivante dans le programme utilisateur STEP 7 : les blocs d'organisation sont par principe toujours appelés par le système, tous les autres le sont par le programme utilisateur STEP 7.

Attente

L'état d'attente peut être obtenu à partir de l'état de marche sur sollicitation depuis la console de programmation. Cet état de fonctionnement permet de réaliser des fonctions de test particulières.

Attribut

Un attribut est une propriété qui caractérise par exemple une désignation de bloc ou un nom de variable. Dans SCL, il existe des attributs pour les indications suivantes : titre du bloc, version, protection du bloc, auteur, nom du bloc, famille de bloc.

B

Bloc

Les blocs constituent des parties délimitées d'un programme utilisateur par leur fonction, leur structure et leur objet. Dans STEP 7, il existe des blocs de code (FB, FC, OB, SFC, SFB), des blocs de données (DB, SDB) et des types de données utilisateur (UDT).

Bloc de code

Dans SIMATIC S7, un bloc de code est un bloc contenant une partie du programme utilisateur STEP 7, contrairement aux blocs de données qui ne contiennent que des données. Il existe, comme blocs de code, les blocs d'organisation (OB, les blocs fonctionnels (FB), les fonctions (FC), les blocs fonctionnels système (SFB) et les fonctions système (SFC).

Bloc de données (DB)

Les blocs de données (DB) sont des zones de mémoire dans le programme utilisateur qui contiennent des données utilisateur. Il existe des blocs de données globales auxquels tous les blocs de code peuvent accéder et des blocs de données d'instance associés à un appel de FB précis.

Bloc de données d'instance (DB d'instance)

Un bloc de données d'instance sauvegarde les paramètres formels et les données locales statiques de blocs fonctionnels. Un DB d'instance est associé à un appel de bloc fonctionnel ou à une hiérarchie d'appel de blocs fonctionnels. Il est généré automatiquement dans SCL.

Bloc de données système (SDB)

Les blocs de données système (SDB) sont des zones de données dans l'unité centrale qui contiennent des paramètres système et des paramètres de module. Vous pouvez les créer et les modifier avec le logiciel de base STEP 7.

Bloc d'organisation (OB)

Les blocs d'organisation constituent l'interface entre le système d'exploitation de la CPU et le programme utilisateur. L'ordre de traitement du programme utilisateur est défini dans les blocs d'organisation.

Bloc fonctionnel

Selon la norme CEI 1131-3, un bloc fonctionnel (FB) est un bloc de code avec données statiques. Un bloc fonctionnel permet la transmission de paramètres dans le programme utilisateur. Aussi, les blocs fonctionnels se prêtent-ils à la programmation de fonctions complexes se répétant souvent, comme par exemple les régulations et le choix du mode de fonctionnement. Puisqu'un FB possède une mémoire (bloc de données d'instance), ses paramètres (par exemple sorties) sont accessibles à tout instant à tout bloc du programme utilisateur.

Bloc fonctionnel système (SFB)

Un bloc fonctionnel système (SFB) est un bloc fonctionnel intégré au système d'exploitation de la CPU qu'il est possible d'appeler dans le programme utilisateur STEP 7.

C

Charger dans la PG

Des objets chargeables (par exemple des blocs de code) sont chargés depuis la mémoire de chargement d'un module programmable dans la console de programmation. Ceci peut être réalisé via une console de programmation directement connectée ou, par exemple, via PROFIBUS.

Charger dans le système cible

Des objets chargeables (par exemple des blocs de code) sont chargés depuis la console de programmation dans la mémoire de chargement d'un module programmable. Ceci peut être réalisé via une console de programmation directement connectée ou, par exemple, via PROFIBUS.

Classe de bloc

En fonction de leur contenu, les blocs sont subdivisés en deux classes, les blocs de code et les blocs de données.

Classeur

Classeur qu'il est possible d'ouvrir dans l'interface utilisateur de SIMATIC Manager et qui peut contenir d'autres classeurs et objets.

Commentaire de bloc

Les commentaires de bloc sont des informations supplémentaires sur un bloc (par exemple des explications sur le processus d'automatisation) qui ne sont pas chargées dans la mémoire de travail des automates programmables SIMATIC S7.

Compilateur de SCL

Le compilateur de SCL est un compilateur séquentiel qui permet de traduire le programme préalablement édité (source SCL) en code machine MC7. Les blocs ainsi créés sont enregistrés dans le programme S7 qui se trouve dans le classeur des blocs.

Compilation

Traduction d'une source en programme utilisateur exécutable.

Compilation source

Dans le mode de saisie source, d'éventuelles erreurs de saisie ne sont recherchées que durant la compilation. Le code exécutable n'est obtenu qu'en absence totale d'erreurs.

Compteur

Les compteurs font partie constituante de la mémoire système de la CPU. Leur contenu est actualisé de façon asynchrone au programme utilisateur par le système. Les instructions en STEP 7 permettent de définir la fonction exacte de la cellule de comptage (par exemple incrémentation) et d'en lancer l'exécution.

Constante (symbolique)

Les constantes aux noms symboliques servent à réserver la place pour les valeurs constantes dans les blocs de code. Elles permettent d'améliorer la lisibilité d'un programme.

Constante (littérale)

Ce sont des constantes dont la valeur et le type sont déterminés par la syntaxe. On distingue des constantes littérales de type numérique, caractère ou de temporisation.

Conversion explicite

La conversion explicite consiste à insérer une fonction de conversion dans le programme source. Lorsque l'utilisateur effectue la combinaison de deux opérandes de types de données différents, il doit réaliser une conversion explicite : lors du passage à une autre classe de types de données, comme par exemple d'un type de données binaire à un type de données numérique et lorsque le type de données cible est moins puissant que le type de données source – même au sein de la même classe.

Conversion implicite

La conversion implicite signifie que le compilateur réalise automatiquement une fonction de conversion. Il y a une conversion lors de la combinaison de deux opérandes de types différents : lorsqu'il n'y a pas de passage dans une autre classe de types de données et lorsque le type de données cible n'est pas moins puisssant que le type de données source.

D

DCB

DCB signifie « Décimal Codé binaire ». L'indication interne à la CPU des temporisations et des compteurs est uniquement réalisée dans le format DCB.

Débogueur de SCL

Le débogueur de SCL est un débogueur de langage évolué permettant de trouver des erreurs de programmation logiques dans le programme utilisateur créé avec SCL.

Déclaration des variables

La déclaration d'une variable consiste à indiquer son mnémonique, son type de données et éventuellement une valeur de présélection, une adresse et un commentaire.

Déclaration du type de données

Dans la déclaration du type de données, l'utilisateur peut déclarer des types de données utilisateur.

Décompilation

La décompilation en LIST permet de charger et d'afficher dans une PG ou un PC quelconques, le bloc chargé dans la CPU. Certaines parties du bloc, comme par exemple les mnémoniques et les commentaires peuvent toutefois manquer.

Données globales

Les données globales sont des données accessibles à chaque bloc de code (FB, FC, OB), à savoir des mémentos (M), des entrées (E), des sorties (A), des temporisations, des compteurs et des éléments de blocs de données. Il est possible d'y accéder par adressage absolu ou symbolique.

Données locales

Les données locales sont les données associées à un bloc de code qui sont déclarées dans la section de déclaration ou dans la déclaration des variables de ce bloc. Elles comprennent – selon le bloc – les paramètres formels, les données statiques, les données temporaires.

Données statiques

Les données statiques sont des données locales d'un bloc fonctionnel, sauvegardées dans le bloc de données d'instance et donc conservées jusqu'au traitement suivant du bloc fonctionnel.

Données temporaires

Les données temporaires sont des données locales d'un bloc de code qui sont rangées dans la pile L pendant l'exécution de ce bloc et ne sont plus disponibles une fois l'exécution achevée.

Données utiles

Les données utiles sont échangées par l'intermédiaire de la mémoire image ou par adressage direct entre une unité centrale et un module de signaux, un module fonctionnel et des modules de communication. Il peut s'agir des signaux d'entrée/sortie TOR et analogiques de modules de signaux, d'informations de commande et d'état de modules de fonctions.

E

Editeur de SCL

L'éditeur de SCL s'adapte tout particulièrement aux exigences de SCL pour créer la source SCL.

En ligne

Il s'agit de l'état de fonctionnement dans lequel aucune liaison (physique, logique) n'est établie entre la console de programmation et l'automate programmable.

Enable (EN)

Tout bloc dans STEP 7 dispose d'une entrée « Enable » (EN) qui réalise l'appel du bloc. Si le signal 1 arrive sur EN le bloc est exécuté, dans le cas d'un signal 0 il n'est pas appelé.

Enable out (ENO)

Tout bloc dans STEP 7 dispose d'une sortie « Enable Output » (ENO). L'utilisateur a la possibilité de combiner l'entrée « Enable » avec une valeur interne (UND) dans le bloc. Le résultat est affecté automatiquement à la sortie ENO. Dans l'imbrication d'appels de blocs, la sortie ENO permet de soumettre l'exécution du bloc suivant au déroulement correct du bloc précédent.

Entier (INT)

L'entier (INT) est l'un des types de données simples. On le représente comme nombre entier à 16 bits.

Etat de bloc

⇒ Visualisation continue

Expression

Une expression permet de traiter des données dans SCL. Il existe des expressions arithmétiques, logiques et de comparaison.

F

Fenêtre de résultat

Il s'agit dans le contexte du débogueur SCL de la fenêtre affichant les résultats des fonctions de test « Activer les points d'arrêt » et « Visualisation continue ».

Fonction (FC)

Selon la norme CEI 1131-3, une fonction (FC) est un bloc de code **sans** données statiques. Elle permet la transmission de paramètres dans le programme utilisateur. Aussi, les fonctions se prêtent-elles à la programmation de fonctions complexes se répétant souvant, comme les calculs.

Fonction système (SFC)

Une fonction système (SFC) est une fonction intégrée au système d'exploitation de la CPU qu'il est possible d'appeler dans le programme utilisateur STEP 7.

G

Globales (données)

Les données globales sont des zones de mémoire de la CPU, accessibles à chaque bloc de code du programme (par exemple aux mémentos).

H

Hiérarchie d'appel

Pour pouvoir être exécutés, tous les blocs doivent préalablement être appelés. On appelle hiérarchie d'appel, l'ordre et l'imbrication de ces appels.

Hors ligne

Il s'agit de l'état de fonctionnement dans lequel aucune liaison (physique, logique) n'est établie entre la console de programmation et l'automate programmable.

I

Identificateur

Les identificateurs servent à adresser des objets du langage SCL. Il existe les classes d'identificateurs suivantes : identificateurs standard, noms et mots-clés prédéfinis, identificateurs absolus (par exemple les identificateurs d'opérandes), noms quelconques de votre choix, par exemple pour les variables et les repères de saut, ou encore mnémoniques définis dans la table des mnémoniques.

Identificateur d'opérande

Un identificateur d'opérande est la partie de l'opérande d'une opération contenant des informations. La zone de mémoire dans laquelle l'opération trouve une valeur qu'elle va combiner ou la grandeur d'une valeur qu'elle va combiner en sont des exemples. Dans l'instruction « Valeur := EB10 », « EB » est l'identificateur d'opérande (« E » désigne la zone d'entrée de la mémoire, « B » désigne un octet dans cette zone).

Instance

On désigne par instance l'appel d'un bloc fonctionnel. Un bloc de données d'instance ou une instance locale lui sont associés. Si, par exemple, un bloc fonctionnel est appelé n fois avec des paramètres et des noms de blocs de données différents dans le programme utilisateur STEP 7, il existe n instances

```
FB13.DB3 (P3:=. . .), FB13.DB4 (P4:=. . .),
FB13.DB5 (P5:=. . .), ..., FB13.DBn (Pn:=. . .),
```

Instance locale Vous définissez l'instance locale dans la section des variables statiques d'un bloc fonctionnel. A la place d'un bloc de données d'instance global, seule une partie locale est utilisée comme zone de données pour le bloc fonctionnel. Vous l'appelez par le nom de l'instance locale.

Instruction Une instruction est la plus petite unité autonome d'un programme utilisateur créé dans un langage textuel. Il s'agit d'une tâche que doit exécuter le processeur.

Instruction CASE L'instruction CASE permet de réaliser un branchement. Elle sert à sélectionner 1 parmi n parties d'un programme en fonction de la valeur d'une expression de sélection.

Instruction CONTINUE Quitte une variable de contrôle et la reprend avec sa valeur suivante.

Instruction EXIT Abandon d'une boucle.

Instruction FOR L'instruction FOR permet de répéter une suite d'instructions tant que la variable de contrôle se trouve dans la plage des valeurs indiquée.

Instruction GOTO L'instruction GOTO réalise le saut immédiat à un repère de saut donné.

Instruction REPEAT L'instruction REPEAT permet de répéter une suite d'instructions jusqu'à une condition d'abandon.

Instruction RETURN L'instruction RETURN permet de quitter immédiatement le bloc en cours.

Interface d'appel L'interface d'appel est définie par les paramètres d'entrée, de sortie et d'entrée/sortie (paramètres formels) d'un bloc dans le programme utilisateur STEP 7. Ces paramètres sont remplacés par les paramètres effectifs lors de l'appel du bloc.

L

Liste d'instructions (LIST) La liste d'instructions (LIST) est un langage de programmation textuel proche du code machine.

M

Mémento (M) Un mémento est une zone de mémoire dans la mémoire système d'une CPU SIMATIC S7 qui peut être adressée en écriture et en lecture (par bit, octet, mot et double mot). L'utilisateur peut utiliser la zone de mémento pour y enregistrer des résultats intermédiaires.

Mémoire image

Les états de signaux des modules d'entrées et de sorties TOR sont rangés dans une mémoire image dans la CPU. On distingue la mémoire image des entrées (MIE) et la mémoire image des sorties (MIS).

Mémoire image des entrées (MIE)

Avant le traitement du programme utilisateur, le système d'exploitation lit la mémoire image des entrées dans les modules d'entrée.

Mémoire image des sorties (MIS)

A la fin du traitement du programme utilisateur, le système d'exploitation transmet la mémoire image des sorties aux modules de sortie.

Mémoire système (zone de mémoire)

La mémoire système est une mémoire vive intégrée à l'unité centrale, dans laquelle sont enregistrées les zones d'opérandes (par exemple temporisations, compteurs, mémentos) ainsi que les zones de données requises de manière interne par le système (par exemple mémoire pour la communication).

Mnémonique

Un mnémonique est un nom que l'utilisateur définit en respectant les règles de syntaxe imposées. Ce nom peut être utilisé en programmation et en contrôle-commande une fois son affectation déterminée (par exemple, variable, type de données, repère de saut, bloc). Exemple : adresse E 5.0, type de données : Bool, mnémonique : Bouton_arret_urg.

Mode de saisie source

Le mode de saisie source consiste à éditer les blocs ou le programme utilisateur entier dans un fichier de texte. La vérification syntaxique n'est réalisée que lors de la compilation. SCL autorise le mode de saisie source.

Mot-clé

Dans SCL, les mots-clés servent à identifier le début d'un bloc, à annoncer les sections de déclaration, à identifier des instructions, des commentaires et des attributs.

Mot d'état

Le mot d'état fait partie des registres de la CPU. Il contient des informations d'état et des informations d'erreur susceptibles de survenir avec le traitement d'instructions STEP 7. Les bits d'état peuvent être lus et décrits par l'utilisateur, les bits d'erreur peuvent uniquement être lus.

Multi-instance

Lorsque vous utilisez des multi-instances, le bloc de données d'instance contient les données pour plusieurs blocs fonctionnels d'une hiérarchie d'appel.

N

Nombre réel

Un nombre réel, ou nombre à virgule flottante est un nombre positif ou négatif représentant une valeur décimale comme par exemple 0.339 ou -11.1.

O

Opérande Un opérande est la partie d'une instruction indiquant l'objet que le processeur doit traiter. On peut y accéder par une adresse absolue ou symbolique.

Opération Une opération est la partie d'une instruction indiquant ce que le processeur doit faire.

P

Paramètre Dans SCL, il s'agit d'une variable d'un bloc de code (paramètre effectif, paramètre formel).

Paramètre d'entrée Seules les fonctions et les blocs fonctionnels possèdent des paramètres d'entrée. Ils permettent de transmettre des données à traiter au bloc appelé.

Paramètre d'entrée/sortie Les fonctions et les blocs fonctionnels possèdent des paramètres d'entrée/sortie. Ils permettent de transmettre des données au bloc appelé, où elles sont traitées, pour ensuite ranger les résultats dans la même variable.

Paramètre de sortie Le paramètre de sortie d'un bloc permet de transmettre des résultats au bloc appelant dans le programme utilisateur STEP 7.

Paramètre effectif Les paramètres effectifs remplacent les paramètres formels lors de l'appel d'un bloc fonctionnel (FB) ou d'une fonction (FC).

Exemple : le paramètre formel « Démarrer » est remplacé par le paramètre effectif « E.3.6 ».

Paramètre formel Un paramètre formel « réserve la place » pour le paramètre effectif dans les blocs de code paramétrables. Pour les FB et les FC, c'est l'utilisateur qui déclare les paramètres formels ; ils existent déjà pour les SFB et les SFC. A l'appel du bloc, un paramètre effectif est affecté au paramètre formel afin que le bloc appelé utilise les valeurs en cours. Les paramètres formels font partie des données locales du bloc et se répartissent en paramètres d'entrée, de sortie et d'entrée/sortie.

Pavé non terminal Un pavé non terminal est un élément complexe, décrit par une nouvelle règle lexicale ou syntaxique.

Pavé terminal Un pavé terminal est un élément de base d'une règle lexicale ou syntaxique dont la description ne nécessite pas de règle supplémentaire, mais une simple explication. Il peut par exemple s'agir d'un mot-clé ou d'un caractère unique.

Programmation structurée

La résolution de tâches d'automatisation complexes appelle la subdivision du programme utilisateur en parties délimitées de programmes (blocs). L'organisation du programme utilisateur peut être fonctionnelle ou dépendre de la structure technologique de l'installation.

Programmation symbolique

Le langage de programmation SCL permet de remplacer les opérandes par des chaînes de caractères symboliques : l'opérande A 1.1 peut par exemple être remplacé par « Vanne_17 ». La correspondance entre l'opérande et la chaîne de caractères qui lui est associée est réalisée dans la liste des symboles.

Programme utilisateur

Le programme utilisateur contient toutes les instructions et déclarations ainsi que toutes les données pour le traitement des signaux permettant de commander une installation ou un processus. Il est associé à un module programmable (par exemple CPU, FM) et peut être structuré en unités plus petites : les blocs.

Programme utilisateur S7

Le programme utilisateur S7 se trouve dans le classeur des blocs. Il contient les blocs qui vont être chargés dans un module programmable S7 (par exemple une CPU) pour y être exécutés afin de réaliser la commande d'une installation ou d'un processus.

Projet

Un projet est un classeur pour tous les objets d'une solution d'automatisation, indépendamment du nombre de stations, modules et de leur mise en réseau.

Protection du bloc

La protection du bloc désigne la possibilité de pouvoir protéger des blocs individuels contre la décompilation, en effectuant la compilation de la source avec le mot-clé « KNOW_HOW_PROTECTED ».

R

Règle lexicale

Les règles lexicales constituent le niveau inférieur dans la description du langage SCL. Elles n'autorisent pas l'utilisation d'un format libre, ce qui signifie qu'elles ne peuvent être complétées ni par des caractères d'espacement, ni par des caractères de commande.

Règle syntaxique

Les règles syntaxiques constituent le niveau supérieur de description du langage SCL. Elles autorisent l'utilisation d'un format libre, ce qui signifie qu'elles peuvent être complétées par des caractères d'espacement et par des caractères de commande.

RUN

A l'état de fonctionnement RUN (marche), le programme utilisateur est traité, la mémoire image est actualisée cycliquement et toutes les entrées TOR sont validées.

RUN-P

L'état de fonctionnement RUN-P se distingue de l'état de fonctionnement RUN, en ce qu'il autorise l'application sans restrictions de nombreuses fonctions de la console de programmation.

S

Saisie source
La programmation dans SCL permet le mode de saisie source d'un programme STEP 7, avec un éditeur de texte quelconque. Ce n'est que durant la compilation que le code de programme réel est créé et que d'éventuelles erreurs sont décelées. Ce mode de saisie se prête à la création symbolique de programmes standard.

SCL
SCL est l'abréviation de « Structured Control Language ». Il s'agit d'un langage évolué proche du PASCAL, conforme à la norme DIN EN 61131-3 (CEI 1131-3), adapté à la programmation de tâches complexes d'automatisation, comme par exemple des algorithmes ou du traitement de données.

Section de déclaration
C'est dans la section de déclaration que vous déclarez les données locales d'un bloc de code.

Single Step
⇒ Test incrémental

Source
Une source (fichier de texte) contient le code source (texte ASCII) qui peut être écrit dans un éditeur de texte quelconque. Elle est compilée en un programme utilisateur exécutable par un compilateur (LIST, SCL). Une source est enregistrée dans le classeur des sources, sous le programme S7.

Source SCL
La source SCL est le fichier SCL dans lequel vous créez le programme qui va ensuite être compilé avec le compilateur de SCL.

Structure (STRUCT)
Une structure est un type de données complexe composé d'éléments de données de types différents. Il peut s'agir d'éléments de données simples ou complexes.

T

Tableau
Un tableau (ARRAY) est un type de données complexe formé d'éléments de données de même type, pouvant eux-mêmes être des types de données simples ou complexes.

Table des mnémoniques
Cette table contient l'affectation de mnémoniques (ou noms symboliques) à des adresses pour les données globales et les blocs. Exemple : Arret_urg (mnémonique), E 1.7 (adresse), Régulateur (mnémonique), SFB24 (bloc).

Table des variables
La table des variables regroupe les variables que vous voulez visualiser et forcer, avec les indications correspondantes sur le format.

Temporisation

Les temporisations font partie constituante de la mémoire système de la CPU. Leur contenu est actualisé de façon asynchrone au programme utilisateur par le système. Les instructions en STEP 7 permettent de définir la fonction exacte de la cellule de temporisation (par exemple retard à la montée) et d'en lancer l'exécution.

Temps de cycle

Le temps de cycle est le temps que met la CPU à traiter une seule fois le programme utilisateur.

Temps de surveillance du cycle

Si le temps d'exécution du programme utilisateur excède le temps de surveillance du cycle choisi, le système d'exploitation émet un message d'erreur et la CPU passe à l'état d'arrêt.

Test incrémental

Le mode de test incrémental est une fonction du débogueur de SCL. Il vous permet d'exécuter le programme instruction par instruction et de le visualiser dans une fenêtre de résultat.

Type de bloc

L'architecture des blocs de STEP 7 distingue les types de blocs suivants : bloc d'organisation, bloc fonctionnel, fonction, bloc de données de même que bloc fonctionnel système, fonction système et bloc de données système ⇒ Bloc.

Type de déclaration

On définit, à l'aide du type de déclaration, comment un paramètre ou une variable locale doivent être utilisés par le bloc. Il existe les paramètres d'entrée, les paramètres de sortie et les paramètres d'entrée/sortie, de même que les variables statiques et temporaires.

Type de données

On définit, à l'aide du type de données, comment la valeur d'une variable ou d'une constante doit être utilisée dans le programme utilisateur. Dans SCL, l'utilisateur en a trois sortes à sa disposition :

- les types de données simples,
- les types de données complexes,
- les types de données utilisateur (UDT).

Type de données complexe

On distingue les structures et les tableaux. Les structures sont composées de divers types de données différents (par exemple des types de données simples), alors que les tableaux comportent plusieurs éléments d'un même type de données. Les types de données STRING et DATE_AND_TIME sont également des types de données complexes.

Type de données simple

Les types de données simples sont des types de données prédéfinis selon CEI 1131-3. Exemples : le type de données BOOL définit une variable binaire (bit) et le type de données INT une variable entière de 16 bits.

Type de données utilisateur

Les types de données utilisateur (UDT) sont créés par l'utilisateur avec la déclaration de type de données. Ils ont un nom en propre et sont donc réutilisables. Un type de données utilisateur peut, par exemple, servir à la création de plusieurs blocs de données de même organisation (par exemple, régulateur).

Type de paramètre Un type de paramètre est un type de données spécial réservé aux temporisations, aux compteurs et aux blocs. Vous pouvez l'utiliser pour les paramètres d'entrée de blocs fonctionnels **et** de fonctions, pour les paramètres d'entrée/sortie uniquement de blocs fonctionnels afin de transmettre des temporisations, des compteurs et des blocs au bloc appelé.

U

UDT ⇒ Type de données utilisateur

V

Variable Une variable définit une donnée de contenu variable pouvant être utilisée dans le programme utilisateur STEP 7. Elle comprend un opérande (par exemple M 3.1) et un type de données (par exemple BOOL) et peut être identifiée par un mnémonique. Vous la déclarez dans la section de déclaration.

Variable OK La variable OK permet de mentionner l'exécution correcte ou incorrecte d'un bloc. Elle a généralement le type de données BOOL.

Visualisation Voir fenêtre de résultat

Visualisation continue Il s'agit de l'un des modes de test dans SCL. La visualisation continue d'un programme vous permet de tester une suite d'instructions, également appelée domaine de visualisation.

Z

Zone de mémoire Une unité centrale dans SIMATIC S7 possède trois zones de mémoire, à savoir la mémoire de chargement, la mémoire de travail et la zone système.

Index

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W

Z

Siemens AG
A&D AS E46

Östliche Rheinbrückenstr. 50
D-76181 Karlsruhe
République Fédérale d’Allemagne

Expéditeur :

Vos . Nom : _

Fonction : _

Entreprise : _

Rue : _

Code postal : _

Ville : _

Pays : _

Téléphone : _

Indiquez votre secteur industriel :

- ❐ Industrie automobile
- ❐ Industrie chimique
- ❐ Industrie électrique
- ❐ Industrie alimentaire
- ❐ Contrôle/commande
- ❐ Construction mécanique
- ❐ Pétrochimie
- ❐ Industrie pharmaceutique
- ❐ Traitement des matières plastiques
- ❐ Industrie du papier
- ❐ Industrie textile
- ❐ Transports
- ❐ Autres _ _ _ _ _ _ _ _ _ _ _

Remarques / suggestions

Vos remarques et suggestions nous permettent d'améliorer la qualité générale de notre documentation. C'est pourquoi nous vous serions reconnaissants de compléter et de renvoyer ces formulaires à Siemens.

Répondez aux questions suivantes en attribuant une note comprise entre 1 pour très bien et 5 pour très mauvais.

1. Le contenu du manuel répond-il à votre attente ? ☐
2. Les informations requises peuvent-elles facilement être trouvées ? ☐
3. Le texte est-il compréhensible ? ☐
4. Le niveau des détails techniques répond-il à votre attente ? ☐
5. Quelle évaluation attribuez-vous aux figures et tableaux ? ☐

Vos remarques et suggestions: