

**RAPPORT
TECHNIQUE
TECHNICAL
REPORT**

**CEI
IEC**

TR 61131-8

Première édition
First edition
2000-01

Automates programmables –

Partie 8:

**Lignes directrices pour l'application et la mise
en oeuvre des langages de programmation**

Programmable controllers –

Part 8:

**Guidelines for the application and implementation
of programming languages**



Numéro de référence
Reference number
IEC/TR 61131-8:2000

Numéros des publications

Depuis le 1er janvier 1997, les publications de la CEI sont numérotées à partir de 60000.

Publications consolidées

Les versions consolidées de certaines publications de la CEI incorporant les amendements sont disponibles. Par exemple, les numéros d'édition 1.0, 1.1 et 1.2 indiquent respectivement la publication de base, la publication de base incorporant l'amendement 1, et la publication de base incorporant les amendements 1 et 2.

Validité de la présente publication

Le contenu technique des publications de la CEI est constamment revu par la CEI afin qu'il reflète l'état actuel de la technique.

Des renseignements relatifs à la date de reconfirmation de la publication sont disponibles dans le Catalogue de la CEI.

Les renseignements relatifs à des questions à l'étude et des travaux en cours entrepris par le comité technique qui a établi cette publication, ainsi que la liste des publications établies, se trouvent dans les documents ci-dessous:

- **«Site web» de la CEI***
- **Catalogue des publications de la CEI**
Publié annuellement et mis à jour régulièrement
(Catalogue en ligne)*
- **Bulletin de la CEI**
Disponible à la fois au «site web» de la CEI* et comme périodique imprimé

Terminologie, symboles graphiques et littéraux

En ce qui concerne la terminologie générale, le lecteur se reportera à la CEI 60050: *Vocabulaire Electrotechnique International* (VEI).

Pour les symboles graphiques, les symboles littéraux et les signes d'usage général approuvés par la CEI, le lecteur consultera la CEI 60027: *Symboles littéraux à utiliser en électrotechnique*, la CEI 60417: *Symboles graphiques utilisables sur le matériel. Index, relevé et compilation des feuilles individuelles*, et la CEI 60617: *Symboles graphiques pour schémas*.

* Voir adresse «site web» sur la page de titre.

Numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series.

Consolidated publications

Consolidated versions of some IEC publications including amendments are available. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Validity of this publication

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology.

Information relating to the date of the reconfirmation of the publication is available in the IEC catalogue.

Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is to be found at the following IEC sources:

- **IEC web site***
- **Catalogue of IEC publications**
Published yearly with regular updates
(On-line catalogue)*
- **IEC Bulletin**
Available both at the IEC web site* and as a printed periodical

Terminology, graphical and letter symbols

For general terminology, readers are referred to IEC 60050: *International Electrotechnical Vocabulary* (IEV).

For graphical symbols, and letter symbols and signs approved by the IEC for general use, readers are referred to publications IEC 60027: *Letter symbols to be used in electrical technology*, IEC 60417: *Graphical symbols for use on equipment. Index, survey and compilation of the single sheets* and IEC 60617: *Graphical symbols for diagrams*.

* See web site address on title page.

**RAPPORT
TECHNIQUE
TECHNICAL
REPORT**

**CEI
IEC**

TR 61131-8

Première édition
First edition
2000-01

Automates programmables –

Partie 8:

**Lignes directrices pour l'application et la mise
en oeuvre des langages de programmation**

Programmable controllers –

Part 8:

**Guidelines for the application and implementation
of programming languages**

© IEC 2000 Droits de reproduction réservés — Copyright - all rights reserved

Aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photo-copie et les microfilms, sans l'accord écrit de l'éditeur.

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

International Electrotechnical Commission
Telefax: +41 22 919 0300

3, rue de Varembé Geneva, Switzerland
e-mail: inmail@iec.ch IEC web site <http://www.iec.ch>



Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

CODE PRIX
PRICE CODE **XC**

*Pour prix, voir catalogue en vigueur
For price, see current catalogue*

SOMMAIRE

	Pages
AVANT-PROPOS	10
Articles	
1 Généralités	12
1.1 Domaine d'application	12
1.2 Documents de références	12
1.3 Généralités	14
2 Introduction à la CEI 61131-3	16
2.1 Considérations générales	16
2.2 Limitations historiques	20
2.3 Nouveautés de la CEI 61131-3	22
2.4 Considérations sur l'ingénierie des logiciels	24
2.4.1 Mesurage de la qualité des logiciels	24
2.4.2 Application des principes d'ingénierie des logiciels	26
2.4.3 Portabilité	30
3 Lignes directrices pour les applications	32
3.1 Utilisation des types de données	32
3.1.1 Initialisation des variables versus initialisation des types	32
3.1.2 Utilisation des types à liste de valeurs ou à sous-plage de valeurs	34
3.1.3 Utilisation de données BCD	34
3.1.4 Utilisation des types REAL	38
3.1.5 Utilisation des données de types chaîne de caractères	38
3.1.6 Utilisation des types date et heure	40
3.1.7 Utilisation des variables multi-éléments	42
3.2 Importation et exportation des données	42
3.2.1 Variables globales et variables externes	44
3.2.2 Variables en entrée/sortie (VAR_IN_OUT)	44
3.3 Utilisation des blocs fonctionnels	48
3.3.1 Types et instances de blocs fonctionnels	48
3.3.2 Portée des données dans un bloc fonctionnel	50
3.3.3 Accès aux blocs fonctionnels et invocation des blocs fonctionnels	52
3.4 Différence entre instance de bloc fonctionnel et fonction	54
3.5 Utilisation d'instances de blocs fonctionnels indirectement référencées	54
3.5.1 Mise en place d'une référence indirecte d'instance de bloc fonctionnel	56
3.5.2 Accès aux instances de blocs fonctionnels référencées indirectement	60
3.5.3 Invocation des instances de blocs fonctionnels référencées indirectement	60
3.5.4 Récursivité des instances de blocs fonctionnels référencées indirectement	66
3.5.5 Contrôle de l'exécution des instances de blocs fonctionnels référencées indirectement	66
3.5.6 Utilisation des instances de blocs fonctionnels référencées indirectement dans les fonctions	66
3.6 Récursivité dans les langages de programmation des automates programmables	66
3.7 Invocations simples ou multiples	68

CONTENTS

	Page
FOREWORD	11
Clause	
1 General.....	13
1.1 Scope	13
1.2 Reference documents	13
1.3 Overview.....	15
2 Introduction to IEC 61131-3.....	17
2.1 General considerations	17
2.2 Historical limitations	21
2.3 New features in IEC 61131-3.....	23
2.4 Software engineering considerations	25
2.4.1 Software quality measures.....	25
2.4.2 Application of software engineering principles.....	27
2.4.3 Portability.....	31
3 Application guidelines.....	33
3.1 Use of data types	33
3.1.1 Type vs. variable initialization	33
3.1.2 Use of enumerated and subrange types.....	35
3.1.3 Use of BCD data	35
3.1.4 Use of REAL data types	39
3.1.5 Use of character string data types	39
3.1.6 Use of time data types.....	41
3.1.7 Use of multi-element variables.....	43
3.2 Data import and export.....	43
3.2.1 Global and external variables	45
3.2.2 Input/output (VAR_IN_OUT) variables	45
3.3 Use of function blocks.....	49
3.3.1 Function block types and instances	49
3.3.2 Scope of data within function blocks	51
3.3.3 Function block access and invocation	53
3.4 Differences between function block instances and functions	55
3.5 Use of indirectly referenced function block instances	55
3.5.1 Establishing an indirect function block instance reference	57
3.5.2 Access to indirectly referenced function block instances	61
3.5.3 Invocation of indirectly referenced function block instances.....	61
3.5.4 Recursion of indirectly referenced function block instances.....	67
3.5.5 Execution control of indirectly referenced function block instances.....	67
3.5.6 Use of indirectly referenced function block instances in functions.....	67
3.6 Recursion within programmable controller programming languages	67
3.7 Single and multiple invocation	69

Articles	Pages
3.8 Dispositifs spécifiques aux langages	70
3.8.1 Fonctionnalités déclenchées par des fronts	70
3.8.2 Utilisation des blocs fonctionnels EN/ENO dans les fonctions et les blocs fonctionnels.....	74
3.8.3 Utilisation de langages non définis dans la CEI 61131-3	76
3.9 Utilisation des éléments SFC.....	76
3.9.1 Commande des actions	78
3.9.2 Actions booléennes	80
3.9.3 Actions non-SFC	90
3.9.4 Actions SFC	92
3.9.5 Blocs fonctionnels SFC	96
3.9.6 Variables «indicateur»	96
3.10 Mécanismes d'ordonnancement, de concurrence et de synchronisation	98
3.10.1 Problèmes des systèmes d'exploitation.....	98
3.10.2 Ordonnancement des tâches	102
3.10.3 Sémaphores.....	104
3.10.4 Messagerie	106
3.10.5 Tampons horaires	108
3.11 Facilités de communications dans l'ISO/CEI 9506-5 et la CEI 61131-5.....	108
3.11.1 Canaux de communication	110
3.11.2 Lire et écrire des variables	110
3.11.3 Blocs fonctionnels de communication	110
3.12 Pratiques de programmation recommandées.....	112
3.12.1 Variables globales.....	112
3.12.2 Sauts dans le langage FBD (blocs fonctionnels)	114
3.12.3 Invocations multiples d'instances de blocs fonctionnels en FBD	114
3.12.4 Couplage de réseaux SFC (diagramme fonctionnel en séquence)	114
3.12.5 Modification dynamique de la priorité des tâches	116
3.12.6 Commande de l'exécution des instances de blocs fonctionnels par les tâches	116
3.12.7 Utilisation des blocs fonctionnels RTC (real time clock = horloge temps réel).....	116
4 Lignes directrices pour la mise en oeuvre	118
4.1 Allocation des ressources.....	118
4.2 Mise en oeuvre des types de données	118
4.2.1 Types de données REAL et LREAL.....	118
4.2.2 Chaînes de caractères	118
4.2.3 Données de type temps	120
4.2.4 Variables multi-éléments	120
4.3 Exécution des fonctions et des blocs fonctionnels.....	120
4.3.1 Fonctions	122
4.3.2 Blocs fonctionnels	122
4.4 Mise en oeuvre des SFC (diagramme fonctionnel en séquence)	124
4.5 Ordonnancement des tâches.....	124
4.5.1 Classification des tâches	126
4.5.2 Priorités des tâches.....	126
4.6 Traitement des erreurs.....	128
4.6.1 Mécanismes de traitement des erreurs	128
4.6.2 Procédures de traitement des erreurs d'exécution.....	132

Clause	Page
3.8 Language specific features.....	71
3.8.1 Edge triggered functionality	71
3.8.2 Use of EN/ENO in functions and function blocks	75
3.8.3 Use of non-IEC 61131-3 languages	77
3.9 Use of SFC elements	77
3.9.1 Action control	79
3.9.2 Boolean actions.....	81
3.9.3 Non-SFC actions	91
3.9.4 SFC actions	93
3.9.5 SFC function blocks	97
3.9.6 “Indicator” variables.....	97
3.10 Scheduling, concurrency and synchronization mechanisms.....	99
3.10.1 Operating system issues	99
3.10.2 Task scheduling	103
3.10.3 Semaphores.....	105
3.10.4 Messaging.....	107
3.10.5 Time stamping	109
3.11 Communication facilities in ISO/IEC 9506-5 and IEC 61131-5.....	109
3.11.1 Communication channels.....	111
3.11.2 Reading and writing variables.....	111
3.11.3 Communication function blocks.....	111
3.12 Recommended programming practices.....	113
3.12.1 Global variables	113
3.12.2 Jumps in function block diagram (FBD) language.....	115
3.12.3 Multiple invocations of function block instances in FBD	115
3.12.4 Coupling of sequential function chart (SFC) networks	115
3.12.5 Dynamic modification of task priorities.....	117
3.12.6 Execution control of function block instances by tasks	117
3.12.7 Use of RTC (real time clock) function blocks.....	117
4 Implementation guidelines.....	119
4.1 Resource allocation.....	119
4.2 Implementation of data types.....	119
4.2.1 REAL and LREAL data types	119
4.2.2 Character strings.....	119
4.2.3 Time data types.....	121
4.2.4 Multi-element variables.....	121
4.3 Execution of functions and function blocks	121
4.3.1 Functions	123
4.3.2 Function blocks	123
4.4 Implementation of sequential function charts (SFCs)	125
4.5 Task scheduling	125
4.5.1 Classification of tasks.....	127
4.5.2 Task priorities	127
4.6 Error handling	129
4.6.1 Error handling mechanisms	129
4.6.2 Run-time error handling procedures.....	133

Articles	Pages
4.7 Interface système.....	136
4.8 Conformité.....	136
4.8.1 Déclaration de conformité.....	136
4.8.2 Jeux d'instructions d'automates.....	138
4.8.3 Essais de conformité.....	138
4.9 Compatibilité avec la CEI 60617-12, la CEI 60617-13 et la CEI 60848.....	138
5 Exigences pour PSE (environnement de support de programmation).....	138
5.1 Interface utilisateur.....	138
5.2 Ecriture des programmes, des fonctions et des blocs fonctionnels.....	140
5.3 Conception et configuration des applications.....	142
5.4 Compilation séparée.....	142
5.5 Séparation entre l'interface et le corps.....	146
5.5.1 Invocation d'une fonction à partir d'une unité de programmation.....	146
5.5.2 Déclaration et invocation d'une instance de bloc fonctionnel.....	146
5.6 Etablissement des liens entre les éléments de configuration et les programmes ..	148
5.7 Gestion de bibliothèques.....	154
5.8 Outils d'analyse.....	156
5.8.1 Simulation et débogage.....	156
5.8.2 Estimation des performances.....	156
5.8.3 Analyse des boucles d'asservissement.....	156
5.8.4 Analyse des SFC.....	158
5.9 Exigences pour la documentation.....	164
5.10 Sécurité des données et des programmes.....	164
5.11 Facilités en ligne.....	164
Bibliographie.....	166
Index.....	168
Figure 1 – Application distribuée.....	16
Figure 2 – Applications autonomes.....	18
Figure 3 – Balayage d'un programme cyclique.....	20
Figure 4 – Bloc fonctionnel BCD_DIF a) Interface externe, b) Corps.....	36
Figure 5 – Bloc fonctionnel SBCD_DIFF a) Définition du type de donnée structurée SBCD_BYTE, b) Interface externe, c) Corps.....	38
Figure 6 – Exemple en littéral structuré (ST) d'utilisation de données de type heure.....	40
Figure 7 – Exemple d'utilisation de tableau.....	42
Figure 8 – Exemples d'utilisation de VAR_IN_OUT.....	46
Figure 9 – Masquage des instances du bloc fonctionnel a) Déclaration du type de bloc fonctionnel contenu b) Déclaration du type de programme conteneur c) Visibilité des instances de blocs fonctionnels.....	52
Figure 10 – Mise en place d'une instance de bloc fonctionnel référencée indirectement a) Mise en place en tant que variable en entrée b) Mise en place en tant que variable d'entrée/sortie c) Mise en place en tant que variable externe.....	58
Figure 11 – Accès à une instance d'un bloc fonctionnel référencée indirectement.....	60
Figure 12 – Invocation d'une instance de bloc fonctionnel référencée indirectement a) Déclaration et invocation littérale; b) Invocation graphique; c) Passage littéral d'un nom d'instance; d) Passage graphique du nom d'instance.....	64
Figure 13 – Estimation des durées des fonctionnalités déclenchées par des fronts a) Exemple de réseau b) Estimation du cas le plus défavorable.....	72

Clause	Page
4.7 System interface	137
4.8 Compliance.....	137
4.8.1 Compliance statement.....	137
4.8.2 Controller instruction sets	139
4.8.3 Compliance testing.....	139
4.9 Compatibility with IEC 60617-12, IEC 60617-13 and IEC 60848	139
5 Programming support environment (PSE) requirements	139
5.1 User interface	139
5.2 Programming of programs, functions and function blocks	141
5.3 Application design and configuration	143
5.4 Separate compilation.....	143
5.5 Interface/body separation	147
5.5.1 Invocation of a function from a programming unit.....	147
5.5.2 Declaration and invocation of a function block instance.....	147
5.6 Linking of configuration elements with programs.....	149
5.7 Library management	155
5.8 Analysis tools.....	157
5.8.1 Simulation and debugging	157
5.8.2 Performance estimation.....	157
5.8.3 Feedback loop analysis	157
5.8.4 SFC analysis.....	159
5.9 Documentation requirements.....	165
5.10 Security of data and programs.....	165
5.11 On-line facilities.....	165
Bibliography	167
Index.....	169
Figure 1 – Distributed application.....	17
Figure 2 – Stand-alone applications	19
Figure 3 – Scanning a cyclic program	21
Figure 4 – Function block BCD_DIFF a) External interface b) Body	37
Figure 5 – Function block SBCD_DIFF a) Definition of structured data type SBCD_BYTE b) External interface c) Body.....	39
Figure 6 – Structured text (ST) example of time data type usage.....	41
Figure 7 – Example of array usage.....	43
Figure 8 – Examples of VAR_IN_OUT usage	47
Figure 9 – Hiding of function block instances a) Declaration of contained function block type b) Declaration of containing program type c) Visibility of function block instances.....	53
Figure 10 – Establishing an indirectly referenced function block instance a) Establishment as input variable b) Establishment as input/output variable c) Establishment as external variable.....	59
Figure 11 – Access to an indirectly referenced function block instance	61
Figure 12 – Invocation of an indirectly referenced function block instance a) Textual declaration and invocation b) Graphical invocation c) Textual passing of instance name d) Graphical passing of instance name	65
Figure 13 – Timing of edge triggered functionality a) Example network b) Worst-case timing	73

Figure 14 – Exemple de contrôle d'exécution (voir article F.4 de la CEI 61131-3)	
a) Langage ST sans EN/ENO b) Langage FBD avec EN/ENO	76
Figure 15 – Exemple de bloc action	78
Figure 16 – Chronométrage des actions booléennes a) Action N (non stockée) b) Action P (pulsée) c) Action S/R ((actionnée/stoppée)	82
Figure 16 – Chronométrage des actions booléennes (<i>suite</i>) d) Action L (limitée dans le temps) e) Action D (différée)	84
Figure 16 – Chronométrage des actions booléennes (<i>suite</i>) f) Action SD (stockée et différée) g) Action DS (différée et stockée)	86
Figure 16 – Chronométrage des actions booléennes (<i>suite</i>) h) SL (stockée et limitée dans le temps).....	88
Figure 17 – Exemple d'action programmée non booléenne	90
Figure 18 – Utilisation du qualificateur P (pulsion) a) Fragment en SFC; b) Corps incorrect pour l'action A15 (langage ST) c) Corps correct pour l'action A15 (langage ST) d) Corps correct pour l'action A15 (langage FBD).....	92
Figure 19 – Exemple d'action SFC activée par SFC	94
Figure 20 – Bloc fonctionnel SFC a) Interface externe b) Corps.....	98
Figure 21 – Phases essentielles de la création d'une unité d'organisation de programmes (POU).....	140
Figure 22 – Phases essentielles de la création d'une application.....	142
Figure 23 – Compilation séparée de fonctions et de blocs fonctionnels sans références externes	144
Figure 24 – Compilation d'un programme avec accès à des variables externes ou représentées directement	144
Figure 25 – Compilation d'une fonction dont le corps contient une invocation d'une autre fonction a) Déclaration d'un exemple de fonction b) Compilation d'un exemple de fonction .	146
Figure 26 – Compilation d'un programme contenant des instances locales de blocs fonctionnels b) Compilation d'un exemple de programme a) Programme d'exemples de déclaration	148
Figure 27 – Exemple de compilation séparée a) Esquisse d'unités de programmation à compiler b) Compilation séparée.....	152
Figure 28 – Processus de configuration a) Déclaration de la configuration b) production d'une application avec liens.....	154
Figure 29 – Etapes de réduction	158
Figure 30 – Réduction des SFC a) Pas de réduction demandée selon la règle de terminaison 1 b) Pas de réduction demandée selon la règle de terminaison 2 c) SFC irréductible (pas sûr); d) SFC irréductible (inaccessible)	162
Tableau 1 – Eléments de la CEI 61131-3 traitant de l'encapsulation et du masquage	26
Tableau 2 – Différences entre systèmes multi-utilisateurs et systèmes en temps réel	102
Tableau 3 – Mécanismes de traitement d'erreurs recommandés.....	130

Figure 14 – Execution control example (see clause F.4 of IEC 61131-3)	
a) ST language without EN/ENO b) FBD language with EN/ENO	77
Figure 15 – Example of an action block.....	79
Figure 16 – Timing of Boolean actions a) N (non-stored) action b) P (pulse) action	
c) S/R (set/reset) action.....	83
Figure 16 – Timing of Boolean actions d) L (time limited) action e) D (time delayed) action....	85
Figure 16 – Timing of Boolean actions f) SD (stored and time delayed) g) DS (time delayed and stored)	87
Figure 16 – Timing of Boolean actions h) SL (stored and time limited) action.....	89
Figure 17 – Example of a programmed non-Boolean action	91
Figure 18 – Use of pulse (P) qualifier a) SFC fragment b) Incorrect body for action A15 (ST language) c) Correct body for action A15 (ST language) d) Correct body for action A15 (FBD language)	93
Figure 19 – Example of an SFC action enabled by an SFC.....	95
Figure 20 – An SFC-function block a) External interface, b) Body.....	99
Figure 21 – Essential phases of program organization unit (POU) creation.....	141
Figure 22 – Essential phases of application creation	143
Figure 23 – Separate compilation of functions and function blocks without external reference.....	145
Figure 24 – Compilation of a program which access to external or directly represented variables	145
Figure 25 – Compilation of a function whose body contains an invocation of another function a) Declaration of example function b) Compilation of example function.....	147
Figure 26 – Compilation of a program containing local instances of function blocks a) Declaration of example program b) Compilation of example program	149
Figure 27 – Separate compilation example a) Sketch of programming units to be compiled b) Separate compilation.....	153
Figure 28 – Configuration process a) Declaration of the configuration b) Production of the linked application	155
Figure 29 – Reduction steps	159
Figure 30 – Reduction of SFCs a) No reduction required as per termination rule 1 b) No reduction required as per termination rule 2 c) An irreducible (unsafe) SFC d) An irreducible (unreachable) SFC	163
Table 1 – IEC 61131-3 elements supporting encapsulation and hiding.....	27
Table 2 – Differences between multi-user and real-time systems.....	103
Table 3 – Recommended error handling mechanisms	131

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

AUTOMATES PROGRAMMABLES –

Partie 8: Lignes directrices pour l'application et la mise en oeuvre des langages de programmation

AVANT-PROPOS

- 1) La CEI (Commission Électrotechnique Internationale) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI, entre autres activités, publie des Normes internationales. Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
- 2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux intéressés sont représentés dans chaque comité d'études.
- 3) Les documents produits se présentent sous la forme de recommandations internationales. Ils sont publiés comme normes, spécifications techniques, rapports techniques ou guides et agréés comme tels par les Comités nationaux.
- 4) Dans le but d'encourager l'unification internationale, les Comités nationaux de la CEI s'engagent à appliquer de façon transparente, dans toute la mesure possible, les Normes internationales de la CEI dans leurs normes nationales et régionales. Toute divergence entre la norme de la CEI et la norme nationale ou régionale correspondante doit être indiquée en termes clairs dans cette dernière.
- 5) La CEI n'a fixé aucune procédure concernant le marquage comme indication d'approbation et sa responsabilité n'est pas engagée quand un matériel est déclaré conforme à l'une de ses normes.
- 6) L'attention est attirée sur le fait que certains des éléments du présent rapport technique peuvent faire l'objet de droits de propriété intellectuelle ou de droits analogues. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de propriété et de ne pas avoir signalé leur existence.

La tâche principale des comités d'études de la CEI est l'élaboration des Normes internationales. Toutefois, un comité d'études peut proposer la publication d'un rapport technique lorsqu'il a réuni des données de nature différente de celles qui sont normalement publiées comme Normes internationales, cela pouvant comprendre, par exemple, des informations sur l'état de la technique.

Un rapport technique ne doit pas nécessairement être révisé avant que les données qu'il contient ne soient plus jugées valables ou utiles par le groupe de maintenance.

La CEI 61131-8, qui est un rapport technique, a été établie par le sous-comité 65B: Dispositifs, du comité d'études 65 de la CEI: Mesure et commande dans les processus industriels.

Le texte de ce rapport technique est issu des documents suivants:

Projet d'enquête	Rapport de vote
65B/284/CDV	65B/336/RVC

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de ce rapport technique.

Ce document, purement informatif, ne doit pas être considéré comme une Norme internationale.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 3.

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PROGRAMMABLE CONTROLLERS –

**Part 8: Guidelines for the application
and implementation of programming languages**

FOREWORD

- 1) The IEC (International Electrotechnical Commission) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of the IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, the IEC publishes International Standards. Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. The IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
- 2) The formal decisions or agreements of the IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested National Committees.
- 3) The documents produced have the form of recommendations for international use and are published in the form of standards, technical specifications, technical reports or guides and they are accepted by the National Committees in that sense.
- 4) In order to promote international unification, IEC National Committees undertake to apply IEC International Standards transparently to the maximum extent possible in their national and regional standards. Any divergence between the IEC Standard and the corresponding national or regional standard shall be clearly indicated in the latter.
- 5) The IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with one of its standards.
- 6) Attention is drawn to the possibility that some of the elements of this technical report may be the subject of patent rights. The IEC shall not be held responsible for identifying any or all such patent rights.

The main task of IEC technical committees is to prepare International Standards. However, a technical committee may propose the publication of a technical report when it has collected data of a different kind from that which is normally published as an International Standard, for example "state of the art".

Technical reports do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful by the maintenance team.

IEC 61131-8, which is a technical report, has been prepared by subcommittee 65B: Devices, of IEC technical committee 65: Industrial-process measurement and control.

The text of this technical report is based on the following documents:

Enquiry draft	Report on voting
65B/284/CDV	65B/336/RVC

Full information on the voting for the approval of this technical report can be found in the report on voting indicated in the above table.

This document which is purely informative is not to be regarded as an International Standard.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 3.

AUTOMATES PROGRAMMABLES –

Partie 8: Lignes directrices pour l'application et la mise en oeuvre des langages de programmation

1 Généralités

1.1 Domaine d'application

Le présent rapport technique s'applique à la programmation des systèmes d'automates programmables utilisant les langages de programmation définis dans la CEI 61131-3. Il fournit aussi des lignes directrices pour la mise en oeuvre de ces langages dans les systèmes d'automates programmables et leur environnement de support de programmation (PSE).

Il convient de se rapporter à la CEI 61131-4 pour tous les autres aspects des applications des systèmes d'automates programmables.

NOTE Ni la CEI 61131-3 ni le présent rapport technique ne traitent des problèmes de sécurité des systèmes d'automates programmables ou de leurs logiciels associés. Il convient, pour ces considérations, de consulter la CEI 61508-1 et la CEI 61508-7.

1.2 Documents de références

CEI 60559:1989, *Arithmétique binaire en virgule flottante pour les systèmes à micro processeurs*

CEI 60617-12:1991, *Symboles graphiques pour schémas – Partie 12: Opérateurs logiques binaires*

CEI 60617-13:1993, *Symboles graphiques pour schémas – Partie 13: Opérateurs analogiques*

CEI 60848:1988, *Etablissement des diagrammes fonctionnels pour systèmes de commande*

CEI 61131-1:1992, *Automates programmables – Partie 1: Informations générales*

CEI 61131-2:1992, *Automates programmables – Partie 2: Spécifications et essais des équipements*

CEI 61131-3:1993, *Automates programmables – Partie 3: Langages de programmation*

CEI 61131-4:1995, *Automates programmables – Partie 4: Guide pour l'utilisateur*

CEI 61131-5:1998, *Automates programmables – Partie 5: Communications*

CEI 61508-1:1998, *Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité – Partie 1: prescriptions générales*

PROGRAMMABLE CONTROLLERS –

Part 8: Guidelines for the application and implementation of programming languages

1 General

1.1 Scope

This technical report applies to the programming of programmable controller systems using the programming languages defined in IEC 61131-3. It also provides guidelines for the implementation of these languages in programmable controller systems and their programming support environments (PSEs).

IEC 61131-4 should be consulted for other aspects of the application of programmable controller systems.

NOTE Neither IEC 61131-3 nor this technical report explicitly addresses safety issues of programmable controller systems or their associated software. IEC 61508-1 and IEC 61508-7 should be consulted for such considerations.

1.2 Reference documents

IEC 60559:1989, *Binary floating-point arithmetic for microprocessor systems*

IEC 60617-12:1991, *Graphical symbols for diagrams – Part 12: Binary logic elements*

IEC 60617-13:1993, *Graphical symbols for diagrams – Part 13: Analogue elements*

IEC 60848:1988, *Preparation of functional charts for control systems*

IEC 61131-1:1992, *Programmable controllers – Part 1: General information*

IEC 61131-2:1992, *Programmable controllers – Part 2: Equipment requirements and tests*

IEC 61131-3:1993, *Programmable controllers – Part 3: Programming languages*

IEC 61131-4:1995, *Programmable controllers – Part 4: User guidelines*

IEC 61131-5:1998, *Programmable controllers – Part 5: Communications*

IEC 61508-1:1998, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 1: General requirements*

CEI 61508-2, — *Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité – Partie 2: Prescriptions pour les systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité* ¹⁾

CEI 61508-3:1998, *Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité – Partie 3: Prescriptions concernant les logiciels*

CEI 61508-4:1998, *Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité – Partie 4: Définitions et abréviations*

CEI 61508-5:1998, *Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité – Partie 5: Exemples de méthodes de détermination concernant les logiciels*

CEI 61508-6, — *Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité – Partie 6: Lignes directrices pour l'application de la CEI 1508-3* ¹⁾

CEI 61508-7, — *Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité – Partie 7; Présentation de techniques de mesures* ¹⁾

ISO/CEI 646:1991, *Technologies de l'information – Jeu ISO de caractères codés à 7 éléments pour l'échange d'informations*

ISO/CEI 9506-5:1999, *Système d'automatisation industrielle, Spécification de messagerie industrielle – Partie 5: Norme d'accompagnement pour les automates programmables* ¹⁾

ANSI/IEEE Std 754: 1985, *Binary floating point arithmetic*

NOTE Les dispositions techniques et normatives de la CEI 60559 et de l'ANSI/IEEE Std 754 sont équivalentes.

1.3 Généralités

L'audience prévue pour le présent rapport technique comprend:

- les utilisateurs, tels qu'ils sont définis dans la CEI 61131-3, des systèmes d'automates programmables, qui doivent programmer, configurer, installer et maintenir des automates programmables faisant partie des processus industriels de mesurage et de commande;
- les implémenteurs de langages de programmation pour les systèmes d'automates programmables, tels qu'ils sont définis dans la CEI 61131-3. Ce groupe peut inclure des vendeurs de *logiciel* et de *matériel* pour la préparation et la maintenance de *programmes* pour ces systèmes, ainsi que les vendeurs des systèmes d'automates programmables eux-mêmes.

L'article 2 du présent rapport technique fournit une introduction générale à la CEI 61131-3 sur les systèmes d'automates programmables, alors que l'article 3 fournit des informations supplémentaires concernant l'application de certains éléments de langages de programmation spécifiés par la CEI 61131-3. L'article 4 fournit des informations relatives à la mise en oeuvre prévue de certains éléments de ces langages de programmation, alors que l'article 5 donne des informations générales sur les besoins en matériel et en logiciel pour le développement et la maintenance de programmes. Il est donc prévu que les utilisateurs d'automates programmables trouveront très utiles les articles 2 et 3 du présent rapport technique, alors que les implémenteurs de langages de programmation trouveront plus d'utilité aux articles 4 et 5, en se référant, si nécessaire, aux informations de base des articles 2 et 3.

¹⁾ A publier.

IEC 61508-2, — *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems* ¹⁾

IEC 61508-3:1998, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements*

IEC 61508-4:1998, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 4: Definitions and abbreviations of terms*

IEC 61508-5:1998, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 5: Examples of methods for the determination of safety integrity levels*

IEC 61508-6, — *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 6: Guidelines on the application of IEC 61508-2 and IEC 61508-3* ¹⁾

IEC 61508-7, — *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 7: Overview of techniques and measures* ¹⁾

ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange*

ISO/IEC 9506-5:1999, *Industrial automation systems – Manufacturing message specification – Part 5: Companion standard for programmable controllers*

ANSI/IEEE Std 754:1985, *Binary floating-point arithmetic*

NOTE The technical and normative provisions of IEC 60559 and ANSI/IEEE Std 754 are equivalent.

1.3 Overview

The intended audience for this technical report consists of:

- *users of programmable controller systems* as defined in IEC 61131-3, who must program, configure, install and maintain programmable controllers as part of industrial process measurement and control systems;
- *implementors of programming languages*, as defined in IEC 61131-3, for programmable controller systems. This may include vendors of *software* and *hardware* for the preparation and maintenance of *programs* for these systems, as well as vendors of the programmable controller systems themselves.

Clause 2 of this technical report provides a general introduction to IEC 61131-3, whilst clause 3 provides complementary information about the application of some of the programming language elements specified in IEC 61131-3. Clause 4 provides information about the intended implementation of some of these programming language elements, whilst clause 5 provides general information about requirements for hardware and software for program development and maintenance. Hence, it is expected that users of programmable controllers will find clauses 2 and 3 of this technical report most useful, whilst programming language implementors will find clauses 4 and 5 more useful, referring to the background material in clauses 2 and 3 as necessary.

¹⁾ To be published.

2 Introduction à la CEI 61131-3

2.1 Considérations générales

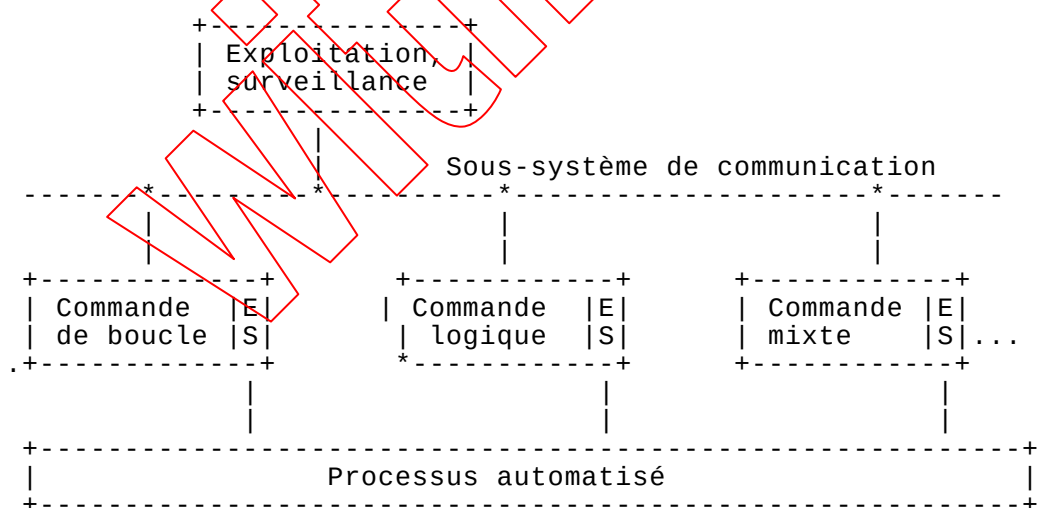
Les possibilités limitées des composants de matériels coûteux ont, dans le passé, imposé de sévères contraintes au processus d'étude des systèmes de commande industriels, de mesurage et d'automatisation. L'étude et la mise en oeuvre du logiciel étaient étroitement dépendantes du matériel choisi. Les spécialistes nécessaires étaient hautement qualifiés, à la fois pour la résolution des problèmes de processus d'automatisation et pour la manipulation d'ensembles de programmation, compliqués et dépendants souvent du matériel.

Avec les innovations rapides dans la micro-électronique et les technologies qui s'y rattachent, le rapport prix/performance des matériels a considérablement décru. Un petit automate programmable peut maintenant coûter bien moins cher que ne coûte sa programmation.

Le coût rapidement décroissant du matériel a entraîné une tendance à remplacer de grands ordinateurs centraux de processus ou d'autres automates autonomes de taille relativement grande par des systèmes distribués spatialement et fonctionnellement.

Comme le montre la figure 1, le sous-système de communications, qui fournit les mécanismes d'échange d'informations entre les dispositifs d'automatisation, constitue le fondement de ces systèmes. Il y a, connectés à ce sous-système, des dispositifs tels que des automates programmables qui fournissent la puissance de traitement distribué du système. Chaque dispositif, contrôlé par son propre logiciel, exécute une sous-tâche qui lui est affectée afin de réaliser la totalité des fonctionnalités du système. On choisit la taille et la puissance de chaque dispositif de façon à satisfaire les demandes de cette sous-tâche particulière.

Historiquement, comme le montre la figure 2, on a utilisé les automates programmables pour des applications autonomes. Les utilisateurs de ces applications bénéficient également de l'évolution esquissée plus haut. A cause des faibles coûts actuels, de nombreuses nouvelles tâches d'automatisation, relativement petites, peuvent être résolues avec souplesse et profit par des automates programmables.



IEC 1614/99

Figure 1 – Application distribuée

2 Introduction to IEC 61131-3

2.1 General considerations

In the past, the limited capabilities of expensive hardware components imposed severe constraints on the design process for industrial process control, measurement and automation systems. Software design and implementation were tightly tailored to the selected hardware. This required specialists who were highly skilled, both in solving process automation problems and in dealing with complicated, often hardware-specific computer programming constructs.

With the rapid innovation in microelectronics and related technologies, the cost/performance ratio of system hardware has decreased dramatically. At present, a small programmable controller may cost many times less than the cost of programming it.

Driven by rapidly decreasing hardware cost, a trend has become established of replacing large, centrally installed process computers or other comparatively large, isolated controllers by systems with spatially and functionally distributed parts.

As illustrated in figure 1, the essential backbone of such systems is the communication subsystem, which provides the mechanism for information exchange between the distributed automating devices. Connected to this backbone are the devices, such as programmable controllers, which deliver the distributed processing power of the system. Each device, under the control of its own software, performs a dedicated subtask to achieve the required overall system functionality. Each device is chosen with the size and performance required to meet the demands of its particular subtask.

Historically, programmable controllers have been used in stand-alone applications as illustrated in figure 2. Users of these applications also stand to gain by the evolution outlined above. Due to the present low cost of hardware components, many of the new relatively small automation tasks can be solved profitably and flexibly by programmable controllers.

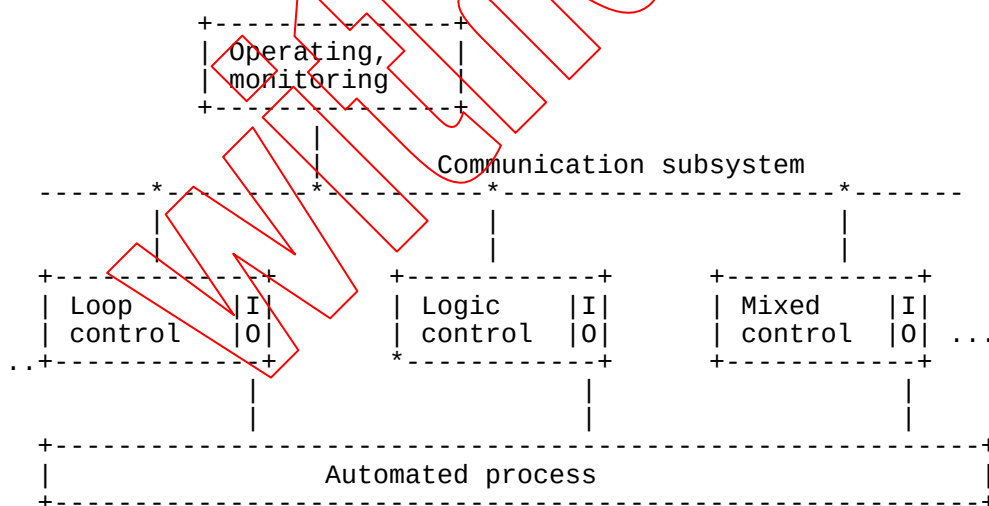


Figure 1 – Distributed application

IEC 1614/99



IEC 1615/99

Figure 2 – Applications autonomes

S'ajoutant au faible coût du matériel, la simplicité de leurs principes de programmation et de leurs système d'exploitation, qui sont facilement compris et appliqués par le personnel de base impliqué dans la programmation, l'exploitation et la maintenance, fait également progresser l'usage intensif des automates programmables pour la résolution des problèmes des tâches d'automatisation.

Les automates programmables utilisent typiquement, comme le montre la figure 3, le principe d'une exécution cyclique des programmes. Ce principe est bien connu et appliqué pour l'exploitation des systèmes de traitement des signaux numériques pour simuler le fonctionnement continu d'un système analogique ou électromécanique. Les valeurs du processus sont lues dans les dispositifs et écrites par le processus, comme des échantillons discrets prélevés de façon aléatoire ou à des intervalles de temps fixes, en fonction de la tâche de surveillance qui doit être accomplie.

Ces principes d'exploitation permettent l'élaboration de programmes pour les automates programmables qui utilisent des éléments très proches des principes des logiques câblées ou des circuits de commande continus utilisés auparavant dans ce cas.

Les principes d'exploitation des automates programmables permettent ainsi des langages de programmation spécifiques aux applications. Associés à l'interface homme/système appropriée, ces langages permettent à l'ingénieur chargé de la surveillance de se consacrer à la solution des tâches de surveillance sans formation intensive en génie logiciel. Les spécifications technologiques de l'ingénierie de surveillance peuvent directement correspondre aux éléments du langage.

Un des avantages particuliers des langages de programmation graphiques est qu'ils proposent une représentation qui ne sert pas uniquement à la saisie et à la documentation des programmes, mais aussi aux essais en ligne et au diagnostic. Les environnements de support de programmation (PSE) des automates programmables sont donc capables de fournir une représentation et une documentation graphique qui sont déjà connues de l'ingénieur d'application et du personnel de base.



IEC 1615/99

Figure 2 – Stand-alone applications

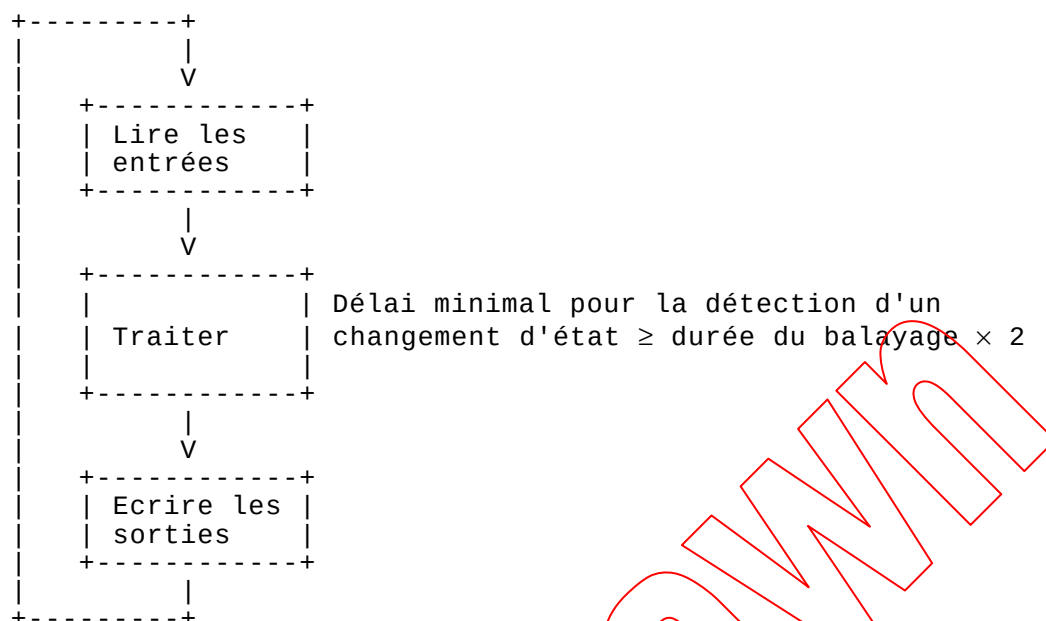
In addition to their low hardware price, the intensive use of programmable controllers in solving automation tasks is also advanced by their straightforward operating and programming principles, which are easily understood and applied by the shop floor personnel involved in programming, operation and maintenance.

Programmable controllers typically employ the principle of cyclic program execution illustrated in figure 3. This principle is well known and applied in the operation of digital signal processing systems to simulate the operation of continuously operating analogue or electromechanical systems. Process values are read into the device and written out to the process as discrete samples at random or equidistant points of time, depending on the control task that has to be fulfilled.

These operating principles allow the construction of programs for programmable controllers using elements closely related to the principles of hard-wired logic or continuous control circuits previously used for the same purpose.

The operating principles of programmable controllers thus enable the provision of application-specific programming languages. Combined with appropriate man-machine interfaces, these languages enable the control engineer to concentrate on solving the problems of the application, without extensive training in software engineering. The control engineer's technological specifications can be mapped directly to the corresponding language elements.

A particular advantage of graphical programming languages is that the representation they offer can be used not only for program input and documentation, but for on-line test and diagnosis as well. Thus, programming support environments (PSEs) for programmable controllers are able to provide the graphically oriented representation and documentation that are already familiar to the application engineer and shop-floor personnel.



IEC 1616/99

Figure 3 – Balayage d'un programme cyclique

2.2 Limitations historiques

On demande souvent à ceux qui conçoivent les systèmes d'automatisation d'utiliser des automates programmables fournis par différents fabricants dans différents systèmes ou même dans le même système. Les automates programmables de constructeurs différents peuvent, cependant, ne pas avoir grand chose de commun sur le plan matériel. Historiquement, il en a résulté des différences significatives dans les éléments et les méthodes de programmation des logiciels, ce qui a conduit au développement d'outils de programmation et de débogage propres à chaque fabricant, qui contiennent souvent des logiciels spécialisés pour la programmation, les essais et la maintenance d'une «famille» particulière d'automates programmables.

Passer d'une famille d'automates à une autre exige souvent, de la part des concepteurs, la lecture de gros manuels relatifs au matériel et au logiciel de la nouvelle famille. Il faut souvent examiner plusieurs fois les manuels afin d'en comprendre la signification exacte et pour pouvoir se servir de la nouvelle famille d'automates de façon correcte. A cause de la concentration sur la tâche ennuyeuse nécessaire à la lecture et à la compréhension du nouveau matériel spécifique à un fournisseur, peu de gens le font. Pour cette raison, beaucoup de gens considèrent l'étude et la programmation de ces automates comme une espèce de magie noire à éviter. Le savoir-faire pour l'utilisation de ces systèmes est en conséquence effectivement concentré entre les mains d'un ou de quelques spécialistes, et ne peut pas être réellement transmis à ceux qui ont la responsabilité de l'exploitation, de la maintenance et de l'amélioration du système.

L'objectif majeur de la CEI 61131-3 est de supprimer les obstacles à la compréhension et à l'utilisation des automates programmables. En conséquence, la CEI 61131-3 introduit de nombreuses facilités pour renforcer les avantages des automates programmables décrits en 2.1, même quand les automates concernés proviennent de différents fournisseurs. On espère que l'extension des domaines d'application des automates programmables qui en résultera compensera les coûts encourus par les fournisseurs pour rendre les systèmes conformes à la norme.

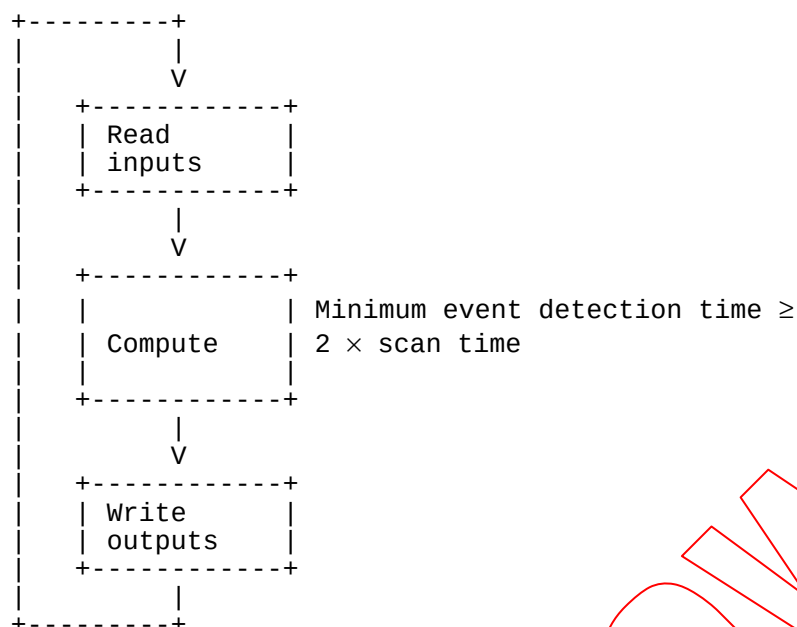


Figure 3 – Scanning a cyclic program

IEC 1616/99

2.2 Historical limitations

Automation system designers are often required to use programmable controllers from various manufacturers in different systems or even in the same system. However, the hardware of programmable controllers from different manufacturers may have very little in common. Historically, this has resulted in significant differences in the elements and methods of programming the software as well. This has led to the development of manufacturer-specific programming and debugging tools, which often carry very specialized software for programming, testing and maintaining particular controller “families”.

Changing from one controller family to another often requires the designer to read large manuals for both the hardware and software of the new family. Often, the manual must be reviewed several times in order to understand the exact meaning and to use the new controller family in an appropriate way. Due to the concentrated, tedious work necessary to read and understand the new, vendor-specific material, few people do it. For this reason, many people regard the design and the programming of such controllers as some black magic to be avoided. Thus, the knowledge of how to use such systems effectively is concentrated in one or a few specialists and cannot be transferred effectively to those responsible for system operation, maintenance and upgrade.

A major goal of IEC 61131-3 is to remove such barriers to the understanding and application of programmable controllers. Thus, IEC 61131-3 introduces numerous facilities to support the advantages of programmable controllers described in 2.1, even if controllers of different vendors are concerned. It is hoped that the resulting expansion of the application domains of programmable controllers will offset the costs that vendors will incur in making their systems compliant to the standard.

2.3 Nouveautés de la CEI 61131-3

Du point de vue de l'ingénieur d'application et de celui du configurateur du système, on peut résumer les nouveautés les plus importantes de la CEI 61131-3 de la façon suivante.

- 1) l'élaboration bien structurée de programmes ascendants ou descendants est facilitée par des structures de langage pour la définition d'objets fonctionnels typés (program organisation unit, unité d'organisation du programme), tels que fonctions, blocs fonctionnels ou programmes.
- 2) l'utilisation de données clairement typées n'est pas seulement encouragée, mais fondamentalement exigée, éliminant une importante cause d'erreurs de programmation.
- 3) un jeu suffisant de dispositifs pour le suivi de l'exécution des unités d'organisation de programmes est inclus: ces dispositifs, associés aux blocs d'étapes, de transition et d'actions, offrent un excellent moyen de représenter, sous une forme concise, des solutions de suivi séquentielles compliquées.
- 4) les fonctionnalités nécessaires à la conception de la communication entre programmes d'application sont prévues. Des fonctions de communication identiques peuvent être utilisées entre deux programmes, qu'ils soient ou non sur le même appareil, ce qui facilite la réutilisation du logiciel dans différents environnements.
- 5) possibilité pour le concepteur de choisir deux langages graphiques et deux langages littéraux, en fonction des besoins de l'application. Ces langages, avec un jeu de «common elements» (éléments communs) graphiques ou littéraux, fournissent un support logiciel aux méthodologies de conception basées sur des modèles bien compris:
 - a) le langage graphique à contacts (LD = *ladder diagram*) modélise des réseaux d'éléments électromécaniques tels que relais à contacts et à bobines, temporisateurs, compteurs, etc., fonctionnant simultanément;
 - b) le langage graphique à blocs fonctionnels (FBD = *function block diagram*) modélise un réseau d'éléments électroniques tels qu'additionneurs, multiplicateurs, registres à décalages, portes, etc., fonctionnant simultanément;
 - c) le langage littéral structuré (ST = *structured text*) modélise les informations typiques des tâches de traitement telles que des algorithmes numériques utilisant des structures que l'on trouve dans les langages de haut niveau à usage général, comme Pascal;
 - d) le langage liste d'instructions (IL = *instruction list*) modélise un bas niveau de programmation des systèmes de surveillance en langage assembleur;
 - e) un jeu de common elements graphiques et littéraux fournit des règles pour la définition des valeurs et des variables, des dispositifs pour la configuration des logiciels et les déclarations d'objets. Les common elements contiennent des éléments graphiques et littéraux pour la construction de:
 - f) sequencial function charts (SFC = *diagramme fonctionnel en séquence*), qui modélisent des dispositifs et des algorithmes de commande séquentiels basés sur le temps ou les changements d'états.
- 6) La souplesse dans le choix des langages convenant à la programmation de fonctionnalités spécifiques aux applications augmentera la réutilisation de solutions logicielles à des problèmes de surveillance de processus.
- 7) Chaque spécialiste d'application au sein d'une équipe projet peut utiliser le style et le langage qui convient à l'application spécifique considérée, avec la garantie que les résultats du travail de tous les spécialistes seront facilement intégrés tous ensemble.

En résumé, l'objectif principal de la CEI 61131-3 est d'introduire tous les concepts et les structures de langages normalisés nécessaires à la solution des problèmes technologiques de chaque application, et de fournir les principes de développement d'éléments logiciels indépendants du constructeur. Cela rend plus facile la réutilisation des conceptions de logiciels de surveillance de différents types d'automates, même s'il faut encore faire quelque efforts pour passer un programme de surveillance d'un automate à un autre.

2.3 New features in IEC 61131-3

From the point of view of the application engineer and the control systems configurator, the most important new features introduced by IEC 61131-3 can be summarized as follows.

- 1) Well-structured 'top-down' or 'bottom-up' program development is facilitated by language constructs for the definition of typed functional objects (program organization units), such as functions, function blocks and programs.
- 2) Strong data typing is not only supported, but inherently required, thus eliminating a major source of programming errors.
- 3) A sufficient set of features for the execution control of program organization units is included; those features associated with steps, transitions and action blocks offer excellent means to represent complicated sequential control solutions in a concise form.
- 4) The necessary functionalities for designing the communication between application programs are provided. Independent of the mapping of programs onto a single device or different devices, identical communication features can be used between two programs. This facilitates the reuse of software in different environments.
- 5) Two graphical languages and two textual languages may be chosen by the designer, according to the requirements of the application. These languages, plus a set of textual and graphical common elements, support software design methodologies based on well-understood models:
 - a) the graphical ladder diagram (LD) language models networks of simultaneously functioning electromechanical elements such as relay contacts and coils, timers, counters, etc.;
 - b) the graphical function block diagram (FBD) language models networks of simultaneously functioning electronic elements such as adders, multipliers, shift registers, gates, etc.;
 - c) the structured text (ST) language models typical information processing tasks such as numerical algorithms using constructs found in general-purpose high level languages such as Pascal;
 - d) the instruction list (IL) language models the low-level programming of control systems in assembly language;
 - e) a set of graphical and textual common elements provides rules for defining values and variables, features for software configuration and object declaration. The common elements include graphical and textual elements for the construction of:
 - f) sequential function charts (SFCs), which model time- and event-driven sequential control devices and algorithms.
- 6) Flexibility in selection of languages suited for programming application-specific functionalities will increase the reuse of software solutions to process control problems.
- 7) Each application specialist on a project team can use a programming style and language suited for the particular functionality in question, with the assurance that the results of the work of the individual specialists will integrate smoothly together.

In summary, the principal goal of IEC 61131-3 is to introduce all the necessary standardized language concepts and constructs to solve the technological problems of each application and to provide principles for the construction of vendor-independent software elements. This facilitates the reusability of control software designs for different controller types, even though some effort will almost always be required in order to move control programs from one controller family to another.

2.4 Considérations sur l'ingénierie des logiciels

2.4.1 Mesurage de la qualité des logiciels

La qualité des systèmes pilotés par des logiciels se mesure à l'aide d'un certain nombre d'attributs [1]¹⁾. Les plus importantes parmi les mesures de la qualité des systèmes de processus industriels de surveillance et d'automatisation sont:

- 1) *capacité*: limite jusqu'à laquelle le système peut assurer les fonctions pour lesquelles il est prévu. Le mesurage de la capacité comprend:
 - a) *temps de réponse*: temps qu'il faut au système pour produire les réponses appropriées à une combinaison spécifiée de changements d'états externes;
 - b) *capacité de traitement*: limite jusqu'à laquelle le système peut respecter les délais planifiés avec un ensemble spécifié de conditions;
 - c) *capacité de stockage*: limite jusqu'à laquelle le système peut garder en mémoire tous les programmes et les données nécessaires avec un ensemble spécifié de conditions.
- 2) *disponibilité*: proportion de temps que le système est capable de consacrer à l'exécution des fonctions pour lesquelles il est prévu. La disponibilité varie en fonction de facteurs tels que:
 - a) *fiabilité*: capacité du système à continuer à remplir les fonctions pour lesquelles il est prévu, pendant une période de temps et avec un ensemble de conditions spécifiés. Une autre mesure de fiabilité est le temps moyen entre pannes (MTBF = mean time between failure);
 - b) *maintenabilité*: facilité avec laquelle on peut remettre le système dans son état de marche intégral après une ou plusieurs pannes dans un domaine défini. Une autre mesure de la maintenabilité est le temps moyen de remise en état (MTTR = mean time to repair); la maintenabilité est souvent définie par l'expression $MTBF/(MTBF+MTTR)$;
 - c) *intégrité*: degré jusqu'où le système peut continuer à remplir les fonctions pour lesquelles il est prévu malgré un nombre défini d'atteintes, y compris les actions involontaires des utilisateurs, les actions volontairement nuisibles, ainsi que les applications potentiellement dangereuses et les changements d'états du système.
- 3) *facilité d'utilisation*: facilité avec laquelle un ensemble défini d'utilisateurs peut acquérir et exercer la possibilité de coopérer avec le système pour remplir les fonctions pour lesquelles ce dernier est prévu. La capacité d'utilisation dépend des facteurs suivants:
 - a) *niveau exigé*: niveau de formation formel ou informel nécessaire avant qu'un utilisateur puisse apprendre à coopérer avec le système (par exemple niveau de formation, expérience de l'utilisation des systèmes d'exploitation, des systèmes de fenêtrage, etc.).
 - b) *besoins en formation*: formation nécessaire pour qu'un utilisateur d'un niveau donné puisse coopérer avec le système pour remplir un ensemble défini de fonctions du système;
 - c) *productivité de l'utilisateur*: nombre d'opérations liées au système effectuées par unité de temps par un utilisateur d'un niveau donné de formation et d'expérience;
 - d) *convivialité*: degré jusqu'où un utilisateur préfère l'utilisation du logiciel du système pour effectuer les fonctions prévues du système à l'utilisation d'activités alternatives qui pourraient ne pas être liées au système.
- 4) *adaptabilité*: facilité avec laquelle la fonctionnalité du système peut être changée de diverses façons comme:
 - a) *faculté d'amélioration*: facilité avec laquelle la *capacité*, la *disponibilité* et/ou les *facilités d'utilisation* du système existant peuvent être augmentées sans changement fondamental des fonctionnalités du système;
 - b) *extensibilité*: facilité avec laquelle on peut ajouter des fonctionnalités au système;

1) Les chiffres entre crochets se rapportent à la bibliographie

2.4 Software engineering considerations

2.4.1 Software quality measures

The quality of software-driven systems is measured by a number of attributes [1] ¹⁾ Among the most important quality measures of industrial process measurement, control and automation systems are;

- 1) *capability*: the extent to which the system performs its intended function. Typical measures of capability include:
 - a) *responsiveness*: the time required for the system to produce appropriate responses to specified combinations of external events;
 - b) *processing capacity*: the extent to which the system can meet scheduling deadlines under specified sets of conditions;
 - c) *storage capacity*: The extent to which the system can retain in memory all the required programs and data under specified sets of conditions.
- 2) *availability*: the proportion of the total process operating time the system is capable of performing its intended function. Availability is affected by factors such as:
 - a) *reliability*: the ability of the system to continue to perform all its intended functions over a specified period of time and range of conditions. An inverse measure of reliability is mean time between failures (MTBF);
 - b) *maintainability*: the ease with which the system can be restored to full capability after the occurrence of one or more faults from a specified set. An inverse measure of maintainability is mean time to repair (MTTR); availability is often defined by the expression $MTBF/(MTBF+MTTR)$;
 - c) *integrity*: the degree to which the system can continue to perform all its intended functions over a specified range of threats, including both unintended user actions, intentionally hostile actions and potentially hazardous application and system events.
- 3) *usability*: The ease with which a specified set of users can acquire and exercise the ability to interact with the system in order to perform its intended functions. Usability is affected by factors such as:
 - a) *entry requirements*: the level of formal and informal training required before the user can learn to interact with the system (e.g. educational level, training in the use of operating systems, windowing systems, etc.);
 - b) *learning requirements*: the training required for a user meeting a specified set of entry requirements to learn to interact with the system to perform a specified set of system functions;
 - c) *user productivity*: the number of system-related operations per unit time which can be performed by a user with a specified level of training and experience;
 - d) *congeniality*: the extent to which a user prefers to utilize the system software to perform the intended system functions, with respect to alternative activities that may or may not be system-related.
- 4) *adaptability*: the ease with which the functionality of the system may be changed in various ways such as:
 - a) *improvability*: the ease with which the existing system *capability*, *availability* and/or *usability* can be upgraded without basic changes in the system's functionality;
 - b) *extensibility*: the ease with which functionality can be added to the system;

¹⁾ Figures in square brackets refer to the bibliography.

- c) *portabilité*: facilité avec laquelle les fonctionnalités d'un système peuvent être transférées d'un système à un autre;
- d) *réutilisabilité*: facilité avec laquelle on peut utiliser les possibilités fonctionnelles d'un élément de logiciel existant pour augmenter les capacités d'un système, qu'il soit nouveau ou non.

2.4.2 Application des principes d'ingénierie des logiciels

On a utilisé différents principes d'ingénierie des logiciels lors de l'élaboration de la CEI 61131-3 afin de promouvoir une meilleure qualité du logiciel. Quelques-uns des plus importants de ces principes, leur contribution à la qualité du logiciel, ainsi que leur intégration dans la CEI 61131-3 sont discutés ci-dessous.

2.4.2.1 Encapsulage et masquage

L'encapsulage est le «conditionnement» de données et/ou de procédures reliées fonctionnellement dans une seule entité logicielle. L'encapsulage contribue à la fiabilité, à la maintenabilité, à la facilité d'utilisation et à l'adaptabilité des logiciels.

On associe à l'encapsulage la notion de masquage de procédures et de données, pour laquelle la seule connaissance que l'utilisateur a d'une entité logicielle est son interface externe ainsi que ses fonctionnalités spécifiées. Les détails de la structure interne des données et de la mise en oeuvre des procédures sont cachés volontairement. Le masquage contribue à la maintenabilité, l'intégrité, la facilité d'utilisation, la portabilité et la réutilisabilité du logiciel.

Les éléments de la CEI 61131-3 qui traitent de l'encapsulage et du masquage, ainsi que les principaux paragraphes s'y rapportant dans la CEI 61131-3 sont énumérés au tableau 1.

Tableau 1 – Eléments de la CEI 61131-3 traitant de l'encapsulage et du masquage

Elément (paragraphe)	Encapsulage		Masquage	
	Données	Procédures	Données	Procédures
Structure (2.3.3)	Oui	Non	Non	Non
Fonction (2.5.1)	Non	Oui	Oui	Oui
Bloc fonctionnel (2.5.2)	Oui	Oui	Oui	Oui
Programme (2.5.3)	Oui	Oui	Oui	Oui
Action (2.6.4)	Non	Oui	Non	Non
Chemin d'accès (2.7.1)	Oui	Non	Oui	Non

2.4.2.2 Représentation explicite d'état

Les éléments du SFC (sequential function chart = diagramme fonctionnel en séquence) définis en 2.6 de la CEI 61131-3 permettent à tout instant de déterminer l'état du système de surveillance par un ensemble d'étapes et d'actions actives. Sans cette représentation, il faut déduire l'état du système à partir de données telles que les entrées et les sorties du système ainsi que de certaines variables d'état (booléennes). L'utilisation des éléments SFC contribue donc à la maintenabilité, à la facilité d'utilisation et à la portabilité du logiciel. Les temps de réponse et les capacités de traitements sont en outre améliorés en ne traitant que la partie des algorithmes qui dépend de l'état présent.

2.4.2.3 Correspondance avec le domaine d'application

On a déjà remarqué, en 2.1 et 2.3 du présent rapport technique, la correspondance directe qui existe entre les éléments de la CEI 61131-3 et des concepts bien connus pour le mesurage, l'automatisation et la surveillance des processus industriels. Cette caractéristique contribue à la maintenabilité, à la facilité d'utilisation et à l'adaptabilité du logiciel.

- c) *portability*: the ease with which the system functionality can be moved from one system to another;
- d) *reusability*: the ease with which the functional capabilities of an existing software element can be used to add capability to a new or existing system.

2.4.2 Application of software engineering principles

A number of software engineering principles were employed in the development of IEC 61131-3 in order to promote increased software quality. A few of the more important principles, their contributors to software quality and their embodiment in IEC 61131-3 are discussed below.

2.4.2.1 Encapsulation and hiding

Encapsulation is the “packaging” of functionally related data and/or procedures in a single software entity. Encapsulation contributes to software reliability, maintainability, usability and adaptability.

Associated with encapsulation is the notion of *hiding* of procedures and data, in which the only knowledge that the user has of a software entity is its external interface and specified functionality. Details of internal data structure and procedure implementation are intentionally hidden. Hiding contributes to software maintainability, integrity, usability, portability and reusability.

The elements of IEC 61131-3 that support encapsulation and hiding and their main subclauses in IEC 61131-3, are listed in table 1.

Table 1 – IEC 61131-3 elements supporting encapsulation and hiding

Elements (subclause)	Encapsulation		Hiding	
	Data	Procedures	Data	Procedures
Structure (2.3.3)	Yes	No	No	No
Function (2.5.1)	No	Yes	Yes	Yes
Function block (2.5.2)	Yes	Yes	Yes	Yes
Program (2.5.3)	Yes	Yes	Yes	Yes
Action (2.6.4)	No	Yes	No	No
Access path (2.7.1)	Yes	No	Yes	No

2.4.2.2 Explicit representation of state

The sequential function chart (SFC) elements defined in 2.6 of IEC 61131-3 enable the state of the control system to be determined at any point in time as the set of active steps and actions. Without this representation, the state of the system must be inferred from data such as system inputs, outputs and some set of state (Boolean) variables. The use of SFC elements thus contributes to software maintainability, usability and portability. Furthermore, system responsiveness and processing capacity are enhanced by performing only those portions of the software algorithm relevant to the current state.

2.4.2.3 Mapping to the application domain

The direct mapping of the elements of IEC 61131-3 to well-understood concepts in industrial process measurement, automation and control has already been noted in 2.1 and 2.3. This characteristic contributes to software maintainability, usability and adaptability.

2.4.2.4 Correspondance entre la conception et la mise en oeuvre

La CEI 61131-3 préconise un style de réalisation de système connu sous le nom de «conception descendante» (ou «conception par décomposition fonctionnelle») suivi d'une «mise en oeuvre ascendante» (ou «mise en oeuvre par assemblage fonctionnel»). Cela contribue à la fiabilité, la maintenabilité, la facilité d'utilisation et à l'adaptabilité du logiciel.

Ce style de conception et de mise en oeuvre du système se caractérise par la séquence d'étapes ci-dessous.

- 1) Les fonctionnalités désirées du système ainsi que les interfaces externes sont spécifiées. Par exemple, les fonctionnalités de base d'une cellule d'usinage peuvent devoir accepter une pièce brute d'un système de manipulation de matériels, produire une pièce finie à partir de l'ébauche brute, mesurer la pièce finie, de la rendre au système de manipulation de matériels, et rendre compte des résultats de l'opération au système d'informations de la fabrication. Les interfaces externes de cette cellule incluent les interfaces avec le système de manipulation de matériels et avec le système d'information. On peut, à ce stade, utiliser les éléments de configuration décrits en 2.7 de la CEI 61131-3.
- 2) On peut procéder à une première subdivision de la conception du système en allouant les fonctionnalités exigées à un ou plusieurs éléments, typiquement des programmes (voir 2.5.3 de la CEI 61131-3). Les interfaces entre les programmes et entre les programmes et les interfaces externes du système sont définies ainsi que la fonction affectée à chaque programme. Cette subdivision suit souvent la partition physique du système; par exemple, dans la cellule d'usinage décrite plus haut, on pourrait définir des programmes séparés pour la station d'usinage, la station de mesurage et, le cas échéant, pour le robot de manipulation des matériels de la cellule.
- 3) Chacun des éléments définis lors de l'étape précédente est ensuite subdivisé en unités plus élémentaires. Si la fonction de l'élément est essentiellement séquentielle, la première phase de la subdivision pourrait être la formulation d'un SFC (diagramme fonctionnel en séquence = sequential function chart) (voir 2.6 de la CEI 61131-3) exprimant la séquence des opérations à effectuer et les conditions pour répéter le cycle des opérations. Chaque action (voir 2.6.4 de la CEI 61131-3) du SFC est alors encore subdivisée, typiquement en *blocs fonctionnels* (voir 2.5.2 de la CEI 61131-3), c'est-à-dire en FBD (*diagramme de blocs fonctionnels* = function block diagram) (voir 4.3 de la CEI 61131-3). Le programme principal pour la station d'usinage dans la cellule décrite plus haut pourrait, par exemple, être un SFC décrivant la séquence des opérations d'usinage à effectuer, alors que les actions pourraient contenir des blocs fonctionnels effectuant les fonctions de surveillance des mouvements nécessaires.
- 4) Ce processus de subdivision fonctionnelle est réalisé récursivement jusqu'à ce que toutes les fonctionnalités puissent être identifiées comme appartenant à des éléments existant dans une bibliothèque (voir 1.4.3 de la CEI 61131-3), ou qu'elles puissent être exprimées de façon algorithmique à l'aide d'un des langages littéraux ou graphiques de la CEI 61131-3, c'est-à-dire le langage *littéral structuré (ST)* (voir 3.2 de la CEI 61131-3), *Liste d'instructions (IL)* (voir 3.1 de la CEI 61131-3), à *contact (LD)* (voir 4.2 de la CEI 61131-3), ou à *blocs fonctionnels (FBD)* (voir 4.3 de la CEI 61131-3).
- 5) Le système est ensuite mis en oeuvre par un assemblage fonctionnel «ascendant», c'est-à-dire en compilant et en ajoutant à la bibliothèque les éléments nouvellement définis dans l'ordre inverse de l'ordre dans lequel ils ont été définis dans les étapes précédentes. Avec un peu d'attention lors de la conception pour la réutilisation, beaucoup de ces nouveaux éléments de la bibliothèque pourront servir lors de la conception de systèmes futurs.
- 6) Finalement, l'affectation des programmes aux ressources, et des ressources aux configurations est terminée, les tâches d'exécution des programmes définies et les chemins d'accès établis pour la communication avec le système d'informations à l'aide des éléments de configuration définis en 2.7 de la CEI 61131-3.

2.4.2.4 Mapping of design to implementation

IEC 61131-3 supports a style of system realization known as “top-down” (or design by functional decomposition) followed by “bottom-up implementation” (or “implementation by functional composition”). This contributes to software reliability, maintainability, usability and adaptability.

This style of system design and implementation is characterized by the following sequence of steps.

- 1) The system's desired functionality and external interface are specified. For instance, the basic functionality of a machining cell might be to accept a rough part from a material handling system, procure a finished part from the rough blank, measure the finished part, pass it back to the material handling system and report the results of the operation to a manufacturing information system. External interfaces in this cell would include the material handling and information system interfaces. The *configuration elements* described in 2.7 of IEC 61131-3 can be used in this step.
- 2) A first decomposition of the system design is made by allocating the required functionality into one or more elements, typically *programs* (see 2.5.3 of IEC 61131-3). The interfaces among the programs and between the programs and the system's external interfaces, are defined and the functionality allocated to each program is defined. Such a decomposition will often follow the physical portioning of the system; for instance, in the machining cell described above, separate programs might be defined for the machining station, the measuring station and for the cell's material handling robot, if any.
- 3) Each element defined in the preceding step is further decomposed into more basic functional units. If the functionality of the element is essentially sequential, the first step in the decomposition may be the formulation of a *sequential function chart* (SFC) (see 2.6 of IEC 61131-3) expressing the sequence of operations to be performed and the conditions for repeating the cycle of operations. Each *action* (see 2.6.4 of IEC 61131-3) of the SFC is then further decomposed, typically into interconnected *function blocks* (see 2.5.2 of IEC 61131-3), i.e. a *function block diagram* (FBD) (see 4.3 of IEC 61131-3). For instance, the main program for the machining station in the cell described above may be a SFC describing the sequence of machining operations to be performed, whilst the actions might contain function blocks performing the required motion control functions.
- 4) This functional decomposition process is performed recursively until all functionality can be identified as belonging to existing library elements (see 1.4.3 of IEC 61131-3) or can be expressed algorithmically in one of the textual or graphic languages in IEC 61131-3, i.e. *structured text* (ST) (see 3.2 of IEC 61131-3), *instruction list* (IL) (see 3.1 of IEC 61131-3), *ladder diagram* (LD) (see 4.2 of IEC 61131-3) or *function block diagram* (FBD) (see 4.3 of IEC 61131-3).
- 5) The system is then implemented by “bottom up” functional composition, i.e. by compiling and adding to the library the newly defined elements in the reverse order in which they have been defined in the preceding steps. With some attention to design for reusability, many of the new library elements may be usable in future system designs.
- 6) Finally, the allocation of *programs* to *resources* and *resources* to *configurations* is completed, program execution tasks are set up and *access paths* are established for communication with information systems, using the configuration elements defined in 2.7 of IEC 61131-3.

2.4.2.5 Programmation structurée

Les techniques de programmation structurée contribuent, c'est bien connu, à la fiabilité, à la maintenabilité et à l'adaptabilité du logiciel. Le langage littéral structuré (ST), défini en 3.3 de la CEI 61131-3, fournit un ensemble complet de structures pour supporter ce style de programmation, tout en restant compatible avec les autres langages graphiques et littéraux ainsi qu'avec les éléments de la CEI 61131-3.

2.4.2.6 Réutilisation du logiciel

Le modèle de programmation décrit en 1.4.3 de la CEI 61131-3 et illustré à la figure 3 de la CEI 61131-3, supporte la réutilisation des éléments de logiciels, ce qui peut être développé par l'utilisateur dans le cadre du processus ascendant de mise en oeuvre décrit en 2.4.2.4, ou peut être fourni sous forme de bibliothèques par les fournisseurs de logiciel. Cette méthode de construction d'un système peut ne pas être familière aux utilisateurs qui ont, dans le passé, développé des applications pour des systèmes d'automatisation sous la forme d'un unique grand contact. Une certaine formation peut donc s'avérer nécessaire pour réaliser les importants gains potentiels de qualité du logiciel et de productivité procurés par la réutilisation du logiciel présentée dans la CEI 61131-3.

Comme on le décrit en 1.4.3 et à l'article B.0 de la CEI 61131-3, les éléments de logiciel que l'on peut mettre dans une bibliothèque pour être réutilisés comprennent, dans un ordre de complexité et de fonctionnalité croissant:

- des types de données;
- des fonctions;
- des blocs fonctionnels;
- des programmes;
- des configurations.

2.4.3 Portabilité

Comme on le note en 2.4.1 ci-dessus, la *portabilité* est définie comme étant la facilité avec laquelle on transporte une fonctionnalité de système d'un système à un autre. Ce concept peut être considéré de deux points de vue:

- la *portabilité entre langages*, c'est-à-dire la facilité avec laquelle on convertit une spécification de type d'unité d'organisation de programme d'un langage à un autre, ou
- la *portabilité entre systèmes*, c'est-à-dire la facilité avec laquelle on convertit une spécification de type d'unité d'organisation de programme d'un environnement d'aide à la programmation (PSE) vers un autre.

2.4.3.1 Portabilité entre langages

Comme on le note en 2.4.2.1, les facilités d'*encapsulation* et de *masquage* de la CEI 61131-3 fournissent un degré très élevé de *réutilisabilité* des fonctions, des blocs fonctionnels et des types de données dans tous les langages définis. Cependant, comme on le note au point 5 de 2.3, chaque langage de la CEI 61131-3 est, jusqu'à un certain point, spécialisé pour un modèle particulier du domaine de problèmes. Cela limite la facilité avec laquelle un algorithme écrit dans un des langages CEI peut être traduit dans un autre langage de programmation. Par exemple:

- les structures de sélections et les itérations du langage ST sont difficiles à traduire efficacement dans les langages FBD ou LD;
- les entrées EN et les sorties ENO des fonctions peuvent être représentées directement dans les langages LD et FBD, mais pas dans les langages ST ou IL;

2.4.2.5 Structured programming

The contribution of structured programming techniques to software reliability, maintainability and adaptability is well known. The structured text (ST) language defined in 3.3 of IEC 61131-3 provides a full set of constructs to support this style of programming, whilst retaining full compatibility with the other graphical and textual languages and elements in IEC 61131-3.

2.4.2.6 Software reuse

The programming model described in 1.4.3 of IEC 61131-3 and shown in figure 3 of IEC 61131-3 strongly supports the reuse of software elements. These may be developed by the user in the “bottom-up” implementation process described in 2.4.2.4 above, or may be supplied as “libraries” by software vendors. This method of system construction may be unfamiliar to users who have previously developed automation system applications as a single, large ladder diagram. Hence, some training may be required in order to realize the large potential gains in software quality and productivity presented by the new approach to software reuse presented in IEC 61131-3.

As described in 1.4.3 and clause B.0 of IEC 61131-3, the software elements that may be placed in libraries for reuse include, in order of increasing complexity and functionality:

- data types;
- functions;
- function blocks;
- programs;
- configurations.

2.4.3 Portability

As noted in 2.4.1 above, *portability* is defined as the ease with which system functionality can be moved from one system to another. This may be considered from the point of view of:

- *inter-language portability*, i.e. the ease of converting a program organization unit type specification from one language to another, or
- *inter-system portability*, i.e. the ease of converting a program organization unit type specification from one programming support environment (PSE) to another.

2.4.3.1 Inter-language portability

As noted in 2.4.2.1, the *encapsulation* and *hiding* facilities of IEC 61131-3 provide a high degree of *reusability* of functions, function blocks and data types among all the defined languages. However, as noted in item 5) of 2.3, each of the IEC 61131-3 languages is specialized to some extent for a particular model of the problem domain. This limits the ease with which an algorithm written in one of the IEC languages can be translated into another programming language. For instance:

- the selection and iteration constructs of the ST language are difficult to translate efficiently into FBD or LD;
- the EN input and ENO output of functions can be represented directly in the LD and FBD languages, but not in the ST or IL language;

- les expressions littérales ne peuvent être utilisées que dans le langage ST, pas dans les langages LD ou FBD.

L'utilisateur peut donc choisir le langage le plus approprié au type d'algorithme à développer au sein d'une fonction ou d'un bloc fonctionnel.

2.4.3.2 Portabilité entre systèmes

La CEI 61131-3 ne définit ni n'exige de définition de type de format commune pour l'échange d'unités d'organisation de programme (POU). En conséquence, les POU conformes, telles que définies en 1.5.2 de la CEI 61131-3, ne seront facilement portables que si elles sont écrites dans un langage littéral (ST ou IL). Les POU conformes ne seront portables que si les dispositifs du système cible sont égaux à ceux du système origine ou en constituent un sur-ensemble.

NOTE Un format d'échange commun est à l'étude en tant qu'extension de la CEI 61131-3.

3 Lignes directrices pour les applications

3.1 Utilisation des types de données

Le paragraphe 2.3.1 de la CEI 61131-3 offre vingt types élémentaires de données. L'utilisateur peut aussi définir de nouveaux types de données, comme il est décrit en 2.3.3 de la CEI 61131-3, quand c'est nécessaire pour satisfaire aux besoins de l'application. Tous les types de données, y compris les données définies par l'utilisateur, sont disponibles pour être utilisées dans une bibliothèque de types de données, comme décrit en 1.4.3 de la CEI 61131-3. L'utilisateur déclare alors le type de données à utiliser pour chaque variable.

Il convient que le choix d'un type pour une variable soit adapté à la plage de valeurs possibles et aux opérations à réaliser sur la variable. Par exemple:

- si une variable ne peut prendre que les valeurs 0 ou 1, et si elle n'est manipulée que par des opérations booléennes, le type BOOL sera alors choisi;
- si un programme d'automate programmable doit compter quelque chose et que les quantités attendues sont dans une plage de 0 à 1 000, on ne peut pas utiliser une variable de type SINT ou USINT, car leur plage de valeurs va de –128 à +127 pour SINT et de 0 à 255 pour USINT. Un type de données raisonnable dans ce cas sera UINT. Ce type fournit une plage de valeurs suffisante et l'utilisation d'un nombre entier sans signe indique clairement qu'on n'attend pas de valeurs négatives.

3.1.1 Initialisation des variables versus initialisation des types

Toutes les variables d'un programme conforme à la CEI 61131-3 doivent être initialisées, soit explicitement par la programmation, soit implicitement par des mécanismes par défaut définis dans la norme. Il ne doit, en principe, jamais y avoir de variables non initialisées. Pour rendre plus facile la déclaration des variables, tous les types élémentaires ont des valeurs initiales par défaut spécifiées dans la norme. Si l'utilisateur ne spécifie pas d'initialisation d'une variable, cette variable aura la valeur initiale par défaut. La plupart des valeurs initiales par défaut sont la représentation de la valeur zéro pour ce type de données.

La CEI 61131-3 permet aussi à l'utilisateur de spécifier la valeur initiale par défaut pour les types définis par l'utilisateur. Par exemple le type déclaré par:

```
Type TempLimit: REAL:= 250.0; END_TYPE
```

Toute variable déclarée de ce nouveau type «TempLimit» sera initialisée avec une valeur par défaut de 250.0 au lieu de 0.0 comme il aurait été normal pour une donnée REAL (nombre réel). De cette façon, dans les déclarations ci-dessous, la variable «Boiler MaxTemperature» prend la valeur initiale 250,0 alors que la variable «PipeMaxTemperature» prend la valeur

- textual expressions can only be used in the ST language, not the LD or FBD language.

Therefore, the user should choose the language most appropriate to the type of algorithm to be developed in the body of a function or function block.

2.4.3.2 Inter-system portability

IEC 61131-3 neither defines nor requires a common exchange format for interchange or program organization unit (POU) type definitions. Consequently, compliant POU, as defined in 1.5.2 of IEC 61131-3, will only be easily portable if they are written in a textual language (ST or IL). Also, compliant POU will not be portable unless the set of features supported in the target system is equal to or is a superset of the features supported in the source system.

NOTE A common exchange format is under consideration as an extension to IEC 61131-3.

3 Application guidelines

3.1 Use of data types

Subclause 2.3.1 of IEC 61131-3 offers twenty elementary data types. The user can also define new data types, as described in 2.3.3 of IEC 61131-3, as necessary to meet the data representation needs of the application. All data types, including user defined types, are made available for use in a library of data types as described in 1.4.3 of IEC 61131-3. The user then declares the data type to be used for each variable.

The selection of a type for a variable should be appropriate to the range of values and operations to be performed on the variable. For instance:

- if a variable can only hold the values 0 or 1 and is only to be operated on by Boolean operations, then the elementary type BOOL should be chosen;
- if a programmable controller program has to count something and the counts are expected to be in the range from zero to one thousand, a variable of type SINT or USINT cannot be used, since their value ranges only extend from –128 to +127 for SINT and from 0 to 255 for USINT. A reasonable data type for this purpose would be UINT. This has a sufficient value range and the usage of an unsigned integer type also makes it clear that negative values are not expected.

3.1.1 Type vs. variable initialization

In a program that complies with IEC 61131-3, each variable has to be initialized either explicitly by programming or implicitly by the default mechanism defined in the standard. Uninitialized values should never occur. To ease the declaration of variables, all elementary types have default initial values specified in the standard. If no initialization of a variable is specified by the user, then that variable will have the default initial value. Most default initial values are defined as the representation of the value of zero for the type.

IEC 61131-3 also allows the user to specify default initial values for user-defined types. For instance, consider a type declared by:

```
TYPE TempLimit: REAL:=250.0; END_TYPE
```

Any declared variable of this new type “TempLimit” is initialized with the default value of 250.0 instead of 0.0 as would be the normal case for all REAL data. Thus, in the following declaration, the variable “BoilerMaxTemperature” is initialized to 250.0, whilst the variable “PipeMaxTemperature” is initialized to 0.0. If the value of zero is not a reasonable maximum

initiale 0.0. Si la valeur 0 n'est pas une température maximale raisonnable pour le tuyau, sa valeur correcte doit être établie avant la première utilisation de cette variable. Omettre cela poserait des problèmes. Dans le présent exemple, la température maximale de la chaudière est initialisée avec la valeur adéquate dans sa valeur initiale par défaut. Il est inutile de fixer cette valeur avant la première utilisation, ce qui simplifie grandement les programmes des automates programmables et augmente la fiabilité du logiciel.

```
VAR_GLOBAL
    BoilerMaxTemperature: TempLimit;
    PipeMaxTemperature: REAL;
END_VAR
```

3.1.2 Utilisation des types à liste de valeurs ou à sous-plage de valeurs

Une donnée de type liste limite les valeurs des variables de ce type à un ensemble de valeurs identifiées par des identificateurs définis par l'utilisateur. Considérons, à titre d'exemple

```
TYPE Color: (Red, Green, Blue); END_TYPE
...
VAR_GLOBAL brickColor: Color; END_VAR
```

On a ici défini un nouveau type «Color». Il ne peut prendre que trois valeurs: Red, Green ou Blue (rouge, vert ou bleu). La norme ne dit pas quelles valeurs numériques ces valeurs énumérées prendront. Il n'existe pas non plus de fonctions de conversion entre les types à listes de valeurs et le type intégral. Les valeurs doivent simplement être distinctes et reproductibles. L'affectation d'une valeur à la variable «brickColor» n'est possible que si on utilise une des valeurs de couleur définies. Toutes les autres valeurs sont signalées comme étant incorrectes.

La CEI 61131-3 fournit des mécanismes pour définir les types à liste de valeurs ou à sous-plage de valeurs. Ces types n'ajoutent pas de nouvelles fonctionnalités à un programme, mais leur utilisation peut au moins rendre un programme plus lisible et est ainsi une aide à la documentation. Dans certains systèmes il peut aussi y avoir une sorte de vérification des valeurs pour des variables de ce type pendant toute la durée du programme. Ce comportement est en général connu sous le nom de «vérification des plages de valeurs».

3.1.3 Utilisation des données BCD

Il convient que les utilisateurs soient avertis du fait que «BCD» n'est pas un type de données dans la CEI 61131-3. BCD représente plutôt une option de codage pour les chaînes de bits dans les types BYTE (demi-mot), WORD (mot), DWORD (mot double) et LWORD (mot long), où ces types de données sont respectivement représentés par 2, 4, 8, ou 16 chiffres BCD. C'est la reconnaissance du fait que BCD est rarement utilisé dans les systèmes modernes, sauf pour le transfert de données dans une chaîne de bits à partir de dispositifs extérieurs ou vers ces dispositifs tels qu'une unité d'affichage multi-segments ou des commutateurs à mollettes.

Comme les systèmes conformes ne sont pas obligés de supporter l'arithmétique BCD, les données codées en BCD doivent être converties dans un des types entiers (SINT, INT, LINT, USINT, UINT, UDINT ou ULINT) en utilisant une des fonctions de conversion BCD_TO_** définies en 2.5.1.5.1 de la CEI 61131-3, afin de pouvoir être manipulées par des opérations arithmétiques. De façon similaire, les fonctions de conversion **_TO_BCD sont fournies pour convertir des données entières sous la forme de données BCD pour le transfert vers des dispositifs extérieurs.

Il convient que les utilisateurs soient avertis des erreurs potentielles qui peuvent être provoquées pendant le codage des données BCD.

temperature for the pipeline, its correct value has to be set before the first usage of the variable. Forgetting this will cause problems. In the present example, the maximum temperature for the boiler is initialized with a proper default initial value. There is no need for a setup before the first usage, which greatly simplifies a programmable controller program and increases software reliability.

```
VAR_GLOBAL  
    BoilerMaxTemperature: TempLimit;  
    PipeMaxTemperature: REAL;  
END_VAR
```

3.1.2 Use of enumerated and subrange types

An *enumerated* data type restricts the values of variables of the type to a user-defined set of identifiers. As an example, consider

```
TYPE Color: (Red, Green, Blue); END_TYPE  
...  
VAR_GLOBAL brickColor: Color; END_VAR
```

Here a new type “Color” is defined. It may only have three values: Red, Green or Blue. The standard does not state which numerical values these enumerated values should have. There is also no conversion function to and from enumerated types to integral types. The values only have to be distinct and reproducible. An assignment of a value to the variable “brickColor” is possible only if one of the defined color values is used. All other values are flagged as errors.

IEC 61131-3 provides mechanisms for the definition of *enumerated* and *subrange* types. These types do not add any new functionality to a program, but their use will at least make a program more readable and thus is a documentation aid. On some systems there might also be some kind of value checking for variables of these types throughout the lifetime of a program. This behaviour is usually referred to as “range checking.”

3.1.3 Use of BCD data

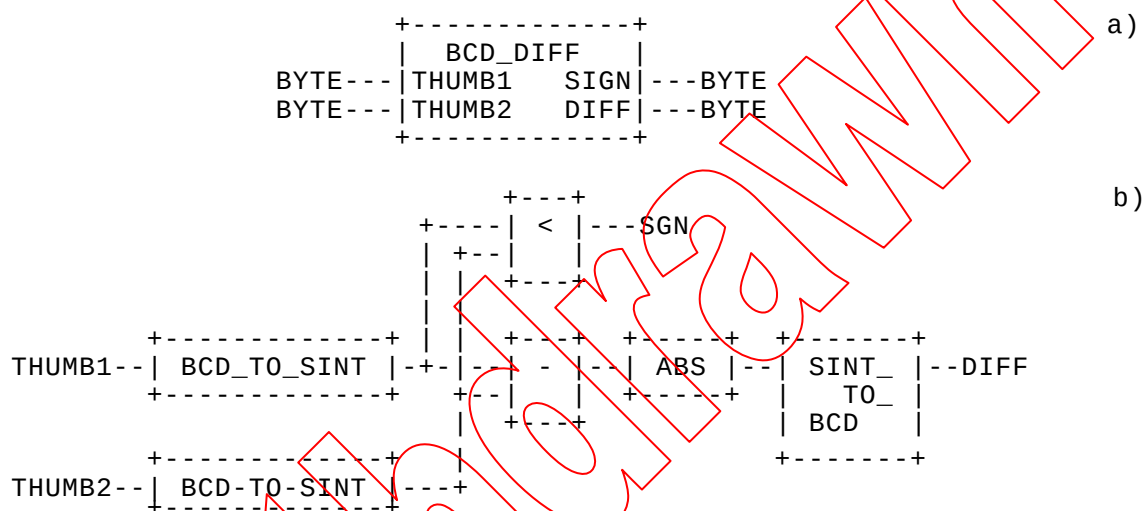
Users should be aware of the fact that “BCD” is not a data type in IEC 61131-3. Rather, it represents an encoding option for the bit string types BYTE, WORD, DWORD and LWORD, where data of these types might be encoded as 2, 4, 8 or 16 BCD digits, respectively. This is in recognition of the fact that BCD is rarely used in modern systems, except for transfer of data in bit-string form to and from external devices such as multi-segment displays and thumbwheel switches.

Since compliant systems are not required to support BCD arithmetic, BCD encoded data must be converted to one of the integer types (SINT, INT, DINT, LINT, USINT, UINT, UDINT or ULINT) using one of the BCD_TO_** conversion functions defined in 2.5.1.5.1 of IEC 61131-3, in order to be manipulated arithmetically. Similarly, the **_TO_BCD functions are provided to convert integer data to BCD encoded form for transfer to external devices.

Users should be aware of the potential errors that may be caused in the encoding of BCD data.

- 1) Comme il n'existe pas de codage BCD normalisé pour le signe «moins», l'utilisation de nombres négatifs en entrée pour une conversion ****_TO_BCD** peut provoquer une erreur de conversion.
- 2) La plage des valeurs d'une variable entière est plus grande que la plage d'une variable chaîne de bits codée en BCD avec le même nombre de bits. Par exemple, la plage d'une variable de type SINT est (–128.. 127), alors que la plage des nombres que l'on peut coder dans une chaîne de bits de type STRING est seulement (0..99).

Les exemples de la figure 4 montrent les précautions qui peuvent être nécessaires pour éviter ces erreurs. Dans cet exemple, le bloc fonctionnel **BCD_DIFF** fait la différence entre deux entrées codées sur deux bits THUMB1 et THUMB2 et le résultat sorti est une chaîne de bits de deux chiffres BCD plus un signe booléen qui peut servir, par exemple, à amener l'affichage BCD d'un signe plus sur deux chiffres.



IEC 1617/99

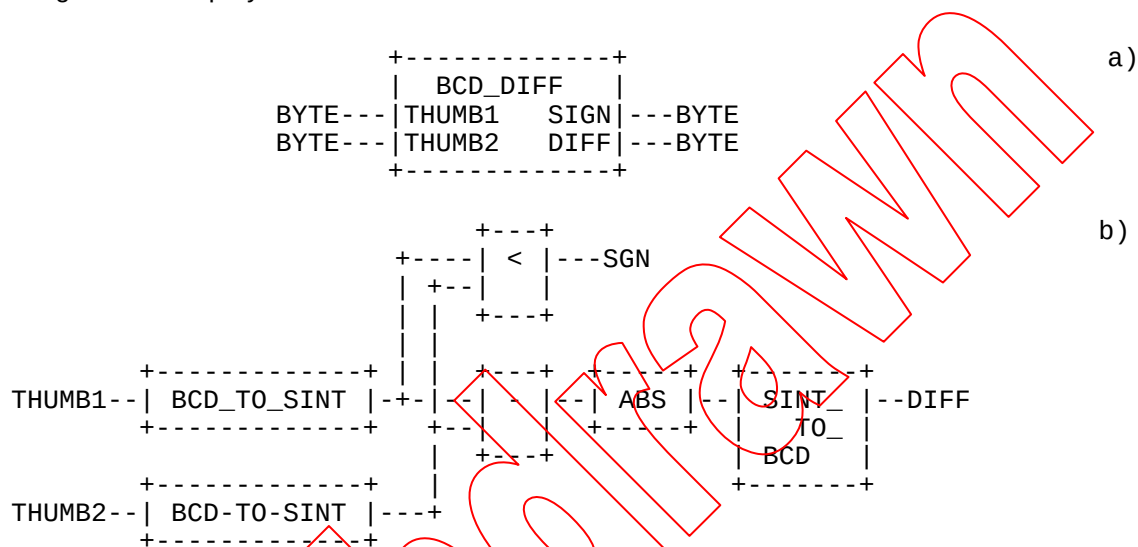
Figure 4 – Bloc fonctionnel BCD_DIFF

a) Interface externe,
b) Corps

Une autre solution à ce problème serait de définir un nouveau type de données structuré contenant le signe et la donnée codée en BCD, tel que **SBCD_BYTE** illustré à la figure 5a). Des fonctions, telles que **SBCD_DIFF** illustrées aux figures 5b) et 5c), pourraient alors être définies pour fournir les valeurs de ce nouveau type comme résultat de leur exécution.

- 1) Since no standard BCD encoding is defined for the “minus” sign, the use of a negative number as an input to a ****_TO_BCD** conversion may cause a conversion error.
- 2) The range of an integer variable is larger than the range of a BCD-encoded bit string variable with the same number of bits. For instance, the range of a variable of type **SINT** is (-128..127), whilst the range of numbers that can be encoded in a bit string of type **BYTE** is only (0..99).

The example shown in figure 4 illustrates precautions that may be necessary to avoid these errors. In this example, the function block **BCD_DIFF** takes the difference between two 2-digit BCD-encoded inputs **THUMB1** and **THUMB2** and outputs the result as a two-digit BCD encoded bit string plus a Boolean sign, which could be used, for example, to drive a “two digit plus sign” BCD display.



IEC 1617/99

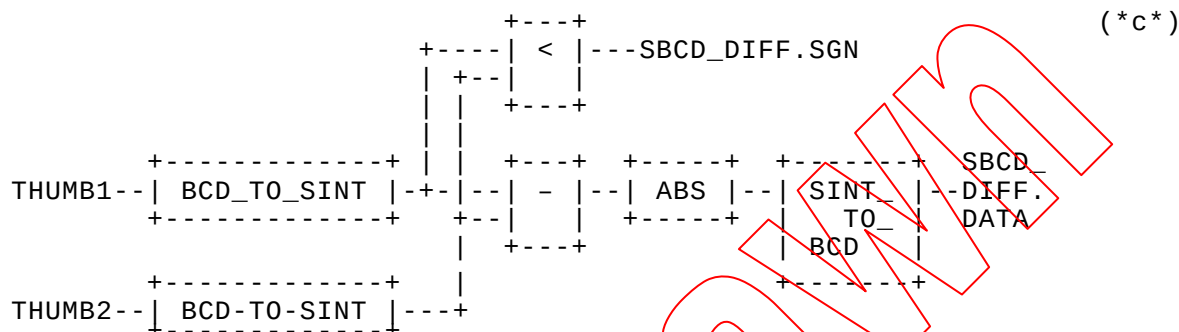
Figure 4 – Function block BCD_DIFF
a) External interface,
b) Body

A different solution to this problem would be to define a new structured data type containing both the sign and BCD-encoded data, such as type **SBCD_BYTE** shown in figure 5a). Functions such as **SBCD_DIFF**, as shown in Figures 5b) and 5c), could then be defined which produce values of this new type as the result of their execution.

```

+-----+
|      BCD_DIFF      |
| THUMB1  SIGN | sBCD_BYTE
| THUMB2  DIFF |
+-----+

```



IEC 1618/99

Figure 5 – Bloc fonctionnel SBCD_DIFF

a) Définition du type de données structurées SBCD_BYTE,
b) Interface externe,
c) Corps

Le type REAL (nombre réel) décrit en 2.3.1 de la CEI 61131-3 peut servir pour exprimer la majorité des valeurs décimales telles que les valeurs de consigne de commande de boucle et les valeurs de processus. Le type de données REAL offre une très grande plage de valeurs de $\pm 10^{\pm 38}$, avec une précision de 1 pour 2^{23} , soit 1 pour 8388608.

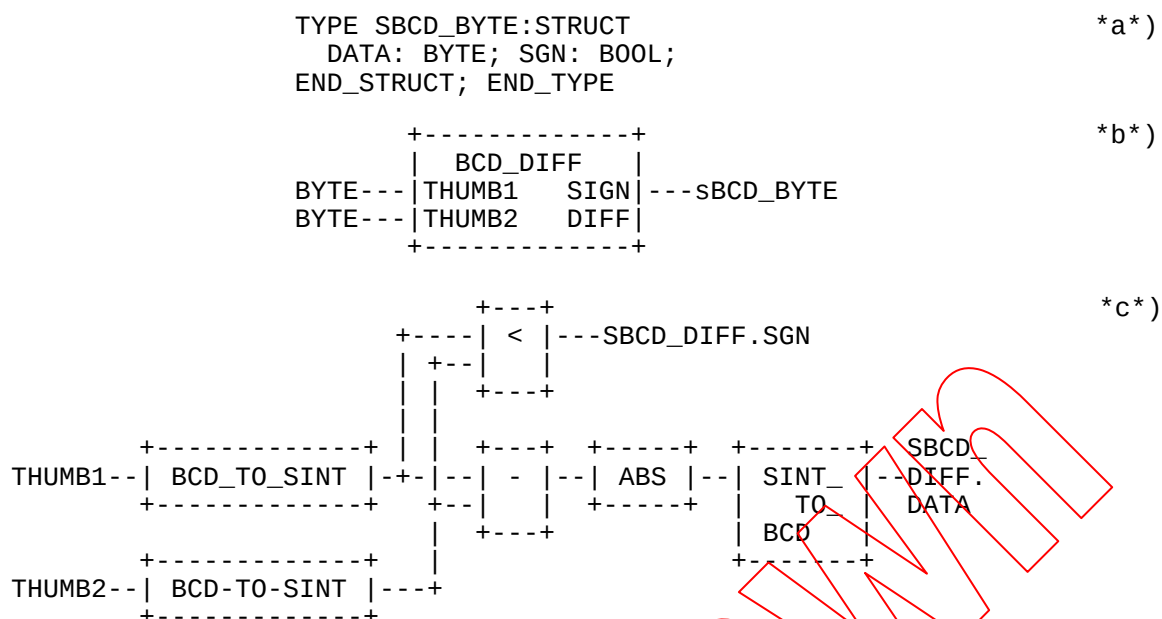
Quand une plage de valeurs plus large ou une meilleure précision sont nécessaires, on doit utiliser le format de 64 bits (long) LREAL avec une plage de $\pm 10^{\pm 308}$ et une précision de 1 pour 252.

NOTE 1 Dans certains algorithmes, les erreurs d'arrondis peuvent être amplifiées par les calculs effectués. On peut avoir besoin, pour éviter ces erreurs, de types de données dont la précision est plus grande qu'il semblait a priori.

NOTE 2 Voir 4.2.1 pour des considérations supplémentaires lors de la mise en oeuvre de données de type REAL.

3.1.5 Utilisation des données de type chaîne de caractères

Le type de données `STRING` (chaîne) fournit un stockage pour des données littérales de longueur variable, ce qui est nécessaire dans la majorité des programmes, par exemple pour conserver les identificateurs des processus, les noms des formules, les codes de sécurité des opérateurs.



IEC 1618/99

Figure 5 – Function block SBCD_DIFF
 a) Definition of structured data type SBCD_BYTE
 b) External interface
 c) Body

3.1.4 Use of REAL data types

The 32 bit REAL data type described in 2.3.1 of IEC 61131-3 can be used for holding the majority of decimal values such as control loop set points and process values. The REAL data type supports a wide range of values within the range $\pm 10^{\pm 38}$, with precision of 1 part in 2^{23} , i.e. 1 part in 8388608.

Where a higher value range or higher precision is required, the 64 bit (long) format LREAL can be used with a range of $\pm 10^{\pm 308}$ and precision of 1 part in 2^{52} .

NOTE 1 In some algorithms, rounding errors may be magnified by the calculations performed. Data types with higher precision than initially apparent may be required in order to avoid such errors.

NOTE 2 See 4.2.1 for additional considerations in the implementation of REAL types.

3.1.5 Use of character string data types

The STRING data type provides storage for variable length textual data, which is required in the majority of application programs, for example for holding processes batch identifiers, recipe names, operator security codes.

La CEI 61131-3 donne aussi des dispositions pour définir des caractères non imprimables dans une chaîne de caractères. Cela est souvent nécessaire quand on construit un message pour un dispositif externe. Par exemple, pour mettre en forme un compte rendu, il peut être nécessaire d'inclure des «alimentations papier» et d'autres caractères de commande similaires dans un message à soumettre à l'impression.

3.1.6 Utilisation des types date et heure

La CEI 61131-3 fournit un certain nombre de types de données pour gérer l'heure du jour, la date et la durée. Enregistrer l'heure à laquelle les activités surviennent, mesurer la durée d'un processus et déclencher des actions à l'instant voulu pendant la journée ou à une date particulière sont des éléments typiques et, dans certains cas, essentiels de la plupart des programmes d'application des processus de production et de fabrication.

L'usage du temps inclut typiquement ce qui suit.

- a) Une définition adéquate de la durée d'une phase d'un processus, par exemple pour un traitement thermique, où le temps de cuisson de certains matériaux est critique.
 - b) L'enregistrement de la date et de l'heure des conditions d'alertes pour procéder à des audits et aussi à des fins de maintenance.
 - c) Une mise en marche contrôlée d'un processus selon l'heure du jour, par exemple pour initialiser le préchauffage d'une cuve de réacteur avant le premier poste de la semaine.
 - d) L'enregistrement de la date de calibrage des données analogiques critiques en entrée, de façon que le système puisse prévenir quand un recalibrage est nécessaire.
 - e) L'enregistrement de l'heure de coupures d'alimentation en énergie et de celle de la reprise, afin de calculer la durée totale des pannes. Cela peut aussi servir à une application pour définir une stratégie contre les défauts de l'alimentation en énergie. Par exemple, si la coupure ne dure que quelques minutes, l'application peut être capable de continuer car les cuves sont toujours chaudes, alors qu'à la suite d'une longue coupure, l'application doit arrêter le processus et faire le nécessaire pour mettre l'usine dans un état de sécurité.
- NOTE Dans cet exemple, on suppose que l'automate programmable est capable de retenir la date et l'heure des coupures d'alimentation en énergie dans une mémoire non effaçable.
- f) La définition de délais pour l'exécution de certaines opérations. Par exemple, si la communication d'une transaction avec un dispositif série n'est pas terminée au bout d'un certain temps, on considère que l'opération n'est pas réussie.

Les types de données TIME, TIME_OF_DAY, DATE et DATE_AND_TIME, peuvent être utilisés dans des expressions avec les opérateurs numériques ADD (+), SUB (–), MUL (*), DIV (/) et aussi avec la fonction de concaténation CONCAT. Dans la figure 6, on donne un exemple d'un tel usage.

```
ProcessDuration := phaseDuration * phaseCount;
endTime         := startTime + ProcessDuration;
endDateAndTime  := CONCAT (todayDate, endTime);
```

IEC 1619/99

Figure 6 – Exemple en littéral structuré (ST) d'utilisation de données de type heure

Il y a également des fonctions de conversion de type pour toutes les manipulations nécessaires entre les dates, les heures du jour et les durées. Par exemple, il est possible de déduire TIME_OF_DAY à partir d'une variable DATE_AND_TIME.

IEC 61131-3 also has provision for defining non-printable characters within a character string. This is often required when constructing messages for external devices. For example, in order to format a report, it may be necessary to embed “form feed” and similar control characters in messages sent to a printer.

3.1.6 Use of time data types

IEC 61131-3 provides a number of data types for holding time of day, date and duration. Recording the times activities occur, measuring process durations and triggering actions at prescribed times of day or on particular dates are typical and in some cases, essential features of most process, production and manufacturing application programs.

Typical usage of time includes the following.

- a) Accurate definition of the duration of a process phase, for example, in heat treatment where the annealing time of some materials is critical.
- b) Recording the date and time of alarm conditions for process audit and maintenance purposes.
- c) Controlled switch-on of a process according to the time of day, e.g. to initiate pre-heating of a reactor vessel before the first shift of the week.
- d) Recording the calibration date of critical analogue inputs so that the system can warn when recalibration is required.
- e) Recording the times of power failure and power resumption, to calculate the down-time duration. This can be used with an application to define a power-fail strategy. For example, if power fails for a few minutes, the application may be able to continue because the process vessels are still warm, whereas, after a long power-failure, the application should abort the process and take whatever action is necessary to put the plant into a safe state.

NOTE This example assumes that the programmable controller is able to retain the date and time of power failure in non-volatile memory.

- f) Defining time-outs for certain operations to complete. For example, if a communications transaction with a serial device is not completed by a certain time, the operation is assumed to have failed.

The time data types TIME, TIME_OF_DAY, DATE and DATE_AND_TIME, can be used in expressions with the numeric operators ADD (+), SUB (-), MUL (*), DIV (/) and also with the concatenation function CONCAT. An example of such usage is given in figure 6.

```
ProcessDuration := phaseDuration * phaseCount;
endTime         := startTime + ProcessDuration;
endDateAndTime  := CONCAT (todayDate, endTime);
```

IEC 1619/99

Figure 6 – Structured text (ST) example of time data type usage

There are also type conversion functions to support all the required manipulation between dates, time of day and duration. For example, it is possible to extract the TIME_OF_DAY from DATE_AND_TIME variable.

3.1.7 Utilisation des variables multi-éléments

Il existe des dispositions de la CEI 61131-3 pour des variables multi-éléments («agrégats»), incluant des tableaux et des structures. Les tableaux sont utilisés dans une grande variété de programmes; leur utilisation peut éviter des répétitions de codes et rend dans bien des cas le programme plus facile à comprendre. La figure 7 donne un exemple de l'utilisation d'une variable tableau. Dans cet exemple, «speeds» est un tableau des vitesses de lignes permises, «lineState» est un nombre entier (INT) donnant l'état de la ligne tel que 0 (zéro) pour une ligne coupée, et de 1 à 3 pour des vitesses graduellement croissantes.

```
VAR_IN lineState: INT; END_VAR;
VBAR_OUT LineSpeed: REAL; END_VAR;
VAR speeds: REAL [0..3]:= (0.0, 1.0, 3.0, 9.0); END_VAR;
lineSpeed:= speeds [lineState];
```

IEC 1620/99

Figure 7 – Exemple d'utilisation de tableau

3.2 Importation et exportation des données

Il y a diverses méthodes de transfert de données de et vers les fonctions, les blocs fonctionnels et les programmes.

Les méthodes les plus restrictives d'accès aux données sont celles des fonctions. Dans les fonctions, seule est permise la lecture de variables en entrée, la fonction doit renvoyer une valeur unique, qui peut être un agrégat (tableau, chaîne ou structure). Les fonctions n'ont pas accès aux variables définies globalement, ni aux variables en entrée ou sortie (voir 3.2.2), ni à des variables représentées directement. Un des autres objectifs de conception de la norme est qu'il ne faut pas de variables statiques, ce qui signifie qu'une fonction ne peut pas, d'un appel à un autre, conserver de valeurs entrées ou calculées. Chaque invocation recevra un ensemble de variables locales nouvellement initialisées, éventuellement à zéro si la fonction n'en a pas décidé autrement lors de sa définition.

Ces restrictions garantissent que la fonction ne causera jamais d'effets de bord. Elle ne peut modifier aucune autre variable et elle ne dépend que du jeu de paramètres en entrée pour l'invocation en cours.

Dans le corps d'un bloc fonctionnel, il est permis de lire des variables en entrée, de lire ou d'écrire des variables en sortie. Les blocs fonctionnels peuvent aussi avoir des variables en entrée et en sortie, qui peuvent être lues ou écrites. Toutes les variables globales définies dans une configuration, une ressource ou un programme à l'aide du mot clé «VAR_GLOBAL» sont accessibles de l'intérieur du bloc fonctionnel, si on a redéclaré ces variables globales à l'aide du mot clé «VAR_EXTERNAL.» Un bloc fonctionnel n'est cependant pas autorisé à accéder à une variable représentée directement, sauf au moyen d'un nom défini globalement et permettant des alias et que l'on peut référencer dans une déclaration «VAR_EXTERNAL». Les valeurs des variables représentées directement peuvent être soumises à un bloc fonctionnel en tant que paramètres d'entrée à partir d'un programme extérieur à ce bloc fonctionnel. Cela est aussi vrai pour les sorties d'un bloc fonctionnel, qui peuvent être copiées dans des variables représentées directement dans un programme. Un bloc fonctionnel peut déclarer des stockages statiques qu'il peut utiliser pour sauver des données à utiliser lors de l'invocation suivante. L'exécution d'un bloc fonctionnel peut donc avoir de nombreux effets sur les données, y compris les données externes, qui peuvent être essentielles pour l'exploitation du bloc fonctionnel.

Les programmes ont le même accès aux données que les blocs fonctionnels, mais ils peuvent en plus utiliser les variables représentées directement et ils peuvent contenir des déclarations de variables globales et des chemins d'accès (voir 2.5.3 de la CEI 61131-3). Ces variables doivent être déclarées dans le programme et peuvent être ensuite utilisées librement.

3.1.7 Use of multi-element variables

There is provision within IEC 61131-3 for multi-element (“aggregate”) variables, including *arrays* and *structures*. Arrays are useful in a wide range of programs. Their use can avoid repetition of code and in many cases can make the program easier to understand. An example of the usage of an array variable is given in figure 7. In this example, “speeds” is an array of permitted line speeds and “lineState” is integer (INT) giving state of the line such as 0 for stopped, 1 to 3 for graded increases in speed.

```
VAR_IN lineState: INT ; END_VAR;  
VBAR_OUT LineSpeed: REAL ; END_VAR;  
VAR speeds: REAL [0..3] := (0.0, 1.0, 3.0, 9.0); END_VAR;  
lineSpeed := speeds [lineState ];
```

IEC 1620/99

Figure 7 – Example of array usage

3.2 Data import and export

There are several methods for the transfer of data to and from functions, function blocks and programs.

The most restricted methods to access data are defined for functions. Inside functions, only reading of input variables is allowed and a function has to return a single value, which may be of an aggregate type (e.g. array, string or structure). Functions do not have any access to globally defined variables, nor can they use input/output variables (see 3.2.2), nor may they access directly represented variables. As another design goal the standard requires functions to have no static variables. This means a function cannot save any of its computed or input values from one invocation to the next. Each invocation will receive a set of fresh initialized local variables, possibly with values of zero if not stated differently by the function definition.

These restrictions ensure that a function will never have any side effects. It cannot modify any other variable and it only depends on the set of input parameters from its current invocation.

Within a function block body, the reading of input variables and reading or writing of output variables are permitted. Function blocks also may have input/output variables, which can be read or written. All global variables that were defined within a configuration, resource or program by use of the “VAR GLOBAL” keyword can be accessed from the inside of the function block, if these global variables are redeclared in the function block definition by use of the “VAR_EXTERNAL” keyword. However, a function block is not allowed to access any directly represented variables, except by means of a globally defined aliasing name that can be referenced in a “VAR_EXTERNAL” declaration. Nevertheless, the values of directly represented variables can be passed to a function block as input parameters from the program outside that function block. The same holds for output values from a function block, which can be copied to directly represented variables in a program. A function block may declare static storage that it can use to save any data from one invocation to the next. Function block execution can therefore have many effects upon data, including external data, which may be essential to the operation of the function block.

Programs have access to data as function blocks have, but additionally may use directly represented variables and can contain declarations of global variables and access paths; see 2.5.3 of IEC 61131-3. These have to be declared within the program and thereafter can be freely read or written.

3.2.1 Variables globales et variables externes

Les variables globales peuvent être définies et initialisées dans une configuration, dans une ressource ou dans un programme à l'aide du mot clé «VAR_GLOBAL». Chaque programme ou chaque bloc fonctionnel qui doit accéder à une ou plusieurs de ces variables doit redéclarer ces variables en utilisant le mot clé «VAR_EXTERNAL».

L'exemple suivant montre comment on définit la variable «TEMP» en dehors d'un bloc fonctionnel et comment on y accède dans un bloc fonctionnel.

```
(* Dans une CONFIGURATION, RESSOURCE ou PROGRAMME *)
VAR_GLOBAL TEMP: INT; END_VAR

(* Accès à partir de l'intérieur d'un BLOC FONCTIONNEL *)
VAR_EXTERNAL TEMP: INT; END_VAR
...
TEMP:=...;
```

Comme les déclarations de variables globales incluent les variables représentées directement, on peut déclarer des alias. Ces alias peuvent être référencés par des déclarations de variables externes dans des blocs fonctionnels et dans des programmes.

Dans l'exemple suivant, on déclare qu'une variable globale «TEMP» est de type INT placée dans un mot en entrée %IW22. Dans un bloc fonctionnel, le nom de variable TEMP est utilisé comme alias pour ce mot en entrée. Un bloc fonctionnel peut ne pas lire directement le mot en entrée %IW22, mais il peut à la place utiliser l'alias.

```
(* Dans une CONFIGURATION, RESSOURCE ou PROGRAMME *)
VAR_GLOBAL TEMP AT %IW22: INT; END_VAR

(* Accès à partir de l'intérieur d'un BLOC FONCTIONNEL *)
VAR_EXTERNAL TEMP: INT; END_VAR
...
...:= TEMP;
```

3.2.2 Variables en entrée/sortie (VAR_IN_OUT)

Les variables en entrée/sortie sont une espèce spéciale de variable que l'on n'utilise qu'avec les blocs fonctionnels et les programmes. Elles ne représentent aucune donnée directement, mais elles référencent d'autres données du type approprié. On les déclare à l'aide du mot clé «VAR_IN_OUT». Les variables en entrée/sortie peuvent être lues ou écrites.

Dans un bloc fonctionnel, les variables en entrée/sortie permettent d'accéder à la valeur originale d'une variable au lieu d'une copie de la valeur contenue par cette variable. Considérons, par exemple, le bloc fonctionnel ACCUM décrit à la figure 8a). A chaque invocation d'une instance du bloc fonctionnel, la valeur en cours de la variable «X» est ajoutée à celle de la variable en entrée/sortie «A». Si une instance de ce bloc fonctionnel est déclarée et invoquée comme le montre la figure 8b), la variable «ACC» sera elle-même augmentée du produit «X1*X2» à chaque invocation d'une instance du bloc fonctionnel «SUM_PROD». Il est inutile de copier la valeur de sortie de AAC1.A dans la variable «ACC». De façon similaire, dans la figure 8c), la variable «ACC» sera augmentée de la somme des produits «X1*X2 + X3*X4» à chaque invocation d'une instance du bloc fonctionnel «SUM_2_PROD».

Comme les sorties d'une fonction n'ont pas de stockage permanent qui leur est associé, la sortie d'une fonction ne peut pas être utilisée comme paramètre dans une variable VAR_IN_OUT, comme le montre la figure 8d). Comme on peut écrire dans une variable en entrée/sortie, il s'ensuit que des littéraux, par exemple «2.0», ou des constantes affectées par des déclarations VAR CONSTANT, ne peuvent être assignées en tant que variables en entrée/sortie, comme le montre la figure 8e).

3.2.1 Global and external variables

Global variables can be defined and initialized within a configuration, resource or program by use of the “VAR_GLOBAL” keyword. Each program or function block that needs access to one or more of these global variables has to redeclare the variables by use of the “VAR_EXTERNAL” keyword.

The following example shows how to define a variable “TEMP” outside a function block and access it within a function block.

```
(* Within a CONFIGURATION, RESOURCE or PROGRAM *)
    VAR_GLOBAL TEMP: INT; END_VAR
(* Access from inside a FUNCTION_BLOCK *)
    VAR_EXTERNAL TEMP: INT; END_VAR
...
    TEMP:=... ;
```

As global variable declarations include directly represented variables, aliases may be declared. These aliases could be referenced by external variable declarations in function blocks and programs.

In the following example, a global variable “TEMP” is declared to be of type INT located at input word %IW22. Within a function block, the variable name “TEMP” is used as an alias for this input word. A function block may not directly read the input word %IW22 but may indirectly use the alias instead.

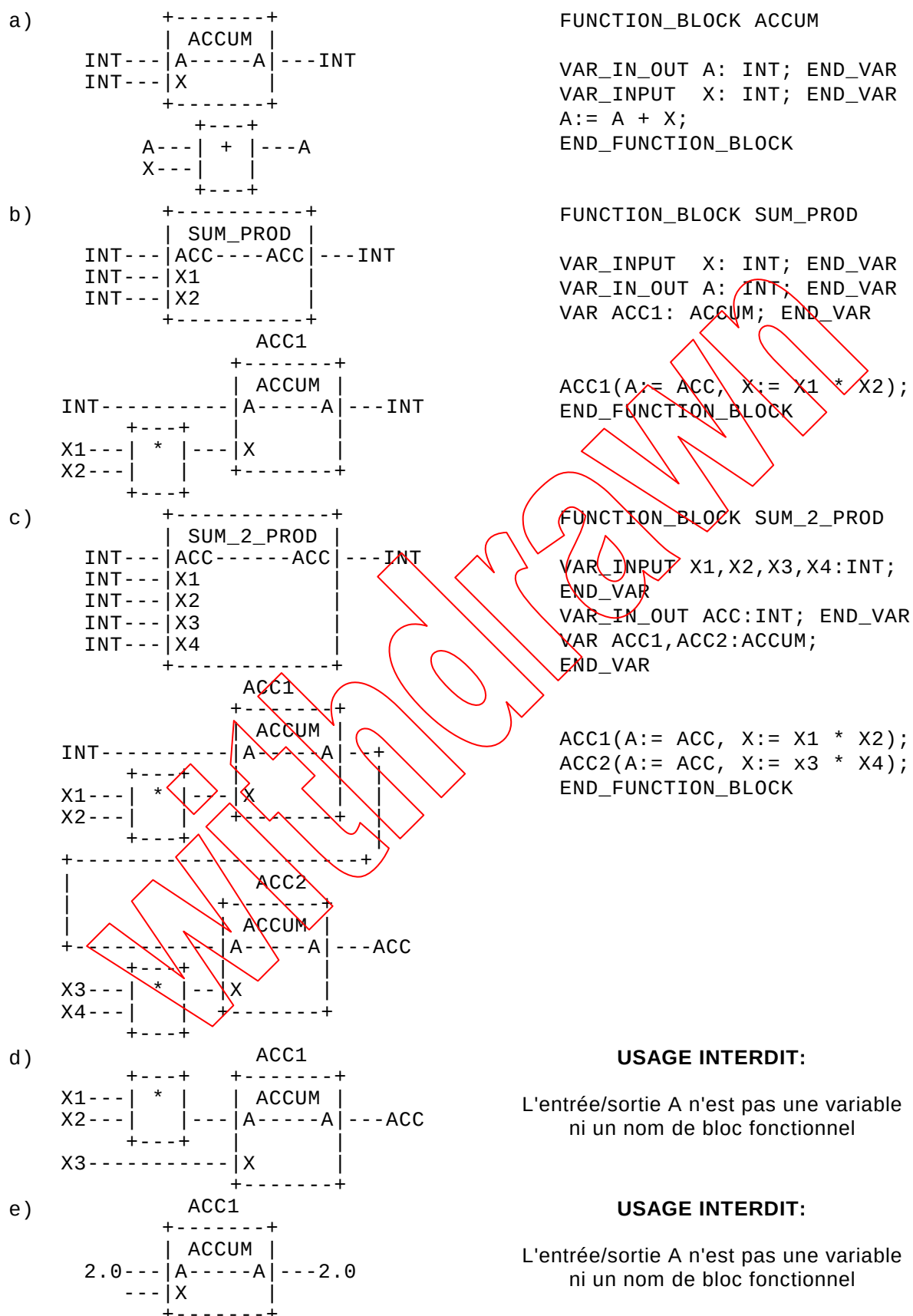
```
(* Within a CONFIGURATION, RESOURCES OR PROGRAM *)
    VAR_GLOBAL TEMP AT %IW22: INT; END_VAR
(* Access from inside a FUNCTION_BLOCK *)
    VAR_EXTERNAL TEMP: INT; END_VAR
...
... := TEMP;
```

3.2.2 Input/output (VAR_IN_OUT) variables

Input/output variables are a special kind of variable used only with function blocks and programs. They do not represent any data directly but reference other data of the appropriate type. They are declared by use of the “VAR_IN_OUT” keyword. Input/output variables may be read or written to.

Inside a function block, input/output variables allow access to the original instance of a variable instead of a copy of the value contained in the variable. Consider, for instance, the function block ACCUM illustrated in figure 8a). Upon each invocation of an instance of the function block, the current value of the input “X” is added to the input/output variable “A”. If an instance of this function block is declared and invoked as shown in figure 8b), the variable “ACC” itself will be augmented by the product “X1*X2” at each invocation of an instance of the function block “SUM_PROD”. There is no need to copy the output value from ACC1.A back into the variable “ACC”. Similarly, in figure 8c), the variable “ACC” will be augmented by the sum of products “X1*X2 + X3*X4” at each invocation of an instance of the function block “SUM_2_PROD”.

Since the output of a function does not have permanent storage associated with it, the output of a function cannot be used as a VAR_IN_OUT parameter, as illustrated in figure 8d). Since an input/output variable may be written to, it follows that integrals, e.g. “2.0” or constants that have been allocated in VAR CONSTANT declarations, cannot be assigned as input/output variables, as illustrated in figure 8e).



- a)
- | | |
|-------------------------|--|
| +-----+ | |
| ACCUM | |
| INT--- A-----A ---INT | |
| INT--- X | |
| +-----+ | |
| +-----+ | |
| A--- + ---A | |
| X--- | |
| +-----+ | |
- FUNCTION_BLOCK ACCUM
- ```

VAR_IN_OUT A : INT ; END_VAR
VAR_INPUT X : INT ; END_VAR
A := A + X ; END_FUNCTION_BLOCK

```
- b)
- |                             |  |
|-----------------------------|--|
| +-----+                     |  |
| SUM_PROD                    |  |
| INT---  ACC-----ACC  ---INT |  |
| INT---  X1                  |  |
| INT---  X2                  |  |
| +-----+                     |  |
- ACC1
- |                         |  |
|-------------------------|--|
| +-----+                 |  |
| ACCUM                   |  |
| INT---  A-----A  ---INT |  |
| +-----+                 |  |
| +-----+                 |  |
| X1---  *  ---X          |  |
| X2---                   |  |
| +-----+                 |  |
- FUNCTION\_BLOCK SUM\_PROD
- ```

VAR_INPUT X : INT ; END_VAR
VAR_IN_OUT A: INT ; END_VAR
VAR ACC1: ACCUM; END_VAR

ACC1(A:= ACC, X:= X1 * X2);
END_FUNCTION_BLOCK

```
- c)
- | | |
|-----------------------------|--|
| +-----+ | |
| SUM_2_PROD | |
| INT--- ACC-----ACC ---INT | |
| INT--- X1 | |
| INT--- X2 | |
| INT--- X3 | |
| INT--- X4 | |
| +-----+ | |
- ACC1
- | | |
|-------------------------|--|
| +-----+ | |
| ACCUM | |
| INT--- A-----A ---INT | |
| +-----+ | |
| +-----+ | |
| X1--- * ---X | |
| X2--- | |
| +-----+ | |
- ACC2
- | | |
|-------------------------|--|
| +-----+ | |
| ACCUM | |
| INT--- A-----A ---ACC | |
| +-----+ | |
| +-----+ | |
| X3--- * ---X | |
| X4--- | |
| +-----+ | |
- FUNCTION_BLOCK SUM_2_PROD
- ```

VAR_INPUT X1,X2,X3,X4:INT; END_VAR
VAR_IN_OUT ACC:INT ; END_VAR
VAR ACC1,ACC2:ACCUM; END_VAR

ACC1(A:= ACC, X:= X1 * X2);
ACC2(A:= ACC, X:= x3 * X4);
END_FUNCTION_BLOCK

```
- d)
- |               |  |
|---------------|--|
| +-----+       |  |
| ACC1          |  |
| X1---  *  --- |  |
| X2---   ---   |  |
| +-----+       |  |
| +-----+       |  |
| X3---   ---X  |  |
| +-----+       |  |
- ACCUM
- |                 |  |
|-----------------|--|
| +-----+         |  |
| A-----A  ---ACC |  |
| +-----+         |  |
- ILLEGAL USAGE:
- Input/output A is not a variable or function block name
- e)
- |                         |  |
|-------------------------|--|
| +-----+                 |  |
| ACC1                    |  |
| ACCUM                   |  |
| 2.0---  A-----A  ---2.0 |  |
| ---  X                  |  |
| +-----+                 |  |
- ILLEGAL USAGE:
- Input/output A is not a variable or function block name

IEC 1621/99

Figure 8 – Examples of VAR\_IN\_OUT usage

### 3.3 Utilisation des blocs fonctionnels

#### 3.3.1 Types et instances de blocs fonctionnels

Un *type de bloc fonctionnel* est une unité d'organisation de programmes (POU = program organisation unit). Cette POU décrit les variables en entrée et en sortie ainsi que les données locales de la zone, pour des *instances* du bloc fonctionnel. Il contient en outre les règles de traitement de ces données lorsqu'une instance du bloc fonctionnel est invoquée. Les variables ne peuvent pas être lues à partir du bloc fonctionnel ou écrites dans le bloc fonctionnel lui-même, il n'est pas non plus possible d'invoquer le type de bloc fonctionnel lui-même; ces opérations sont réservées aux instances du bloc fonctionnel.

On peut utiliser les *instances multiples* de blocs fonctionnels basées sur cette déclaration de type. Les instances individuelles sont indépendantes les unes des autres. Chaque instance de bloc fonctionnel a un identificateur unique (le nom de l'instance) et une zone de données privées utilisée en entrée, en sortie et en variables internes pour cette instance de bloc fonctionnel. On peut accéder à une instance de bloc fonctionnel et l'invoquer par son nom d'instance.

Ces qualités des blocs fonctionnels sont relatives à une programmation orientée objet (OOP = object oriented programming). Le type de bloc fonctionnel est similaire à une *classe*, qui définit la structure de données ainsi que les méthodes de traitement dans le corps du bloc fonctionnel. Les objets individuels sont représentés par les zones de données privées de chaque instance individuelle du bloc fonctionnel. Ces données ne peuvent être modifiées que d'une manière contrôlée, à partir de l'extérieur du corps du bloc fonctionnel. Cela confirme les principes d'ingénierie logicielle d'encapsulation et de masquage de l'information, un élément clé de l'OOP.

Les instances typiques de blocs fonctionnels sont les minuteurs ou les compteurs, qui gardent leurs valeurs d'une invocation à une autre et qui déterminent si on a ou non atteint une valeur finale.

L'accès contrôlé à des dispositifs partagés constitue un autre domaine d'intérêt pour l'utilisation des blocs fonctionnels. Dans ce cas, une instance unique du bloc fonctionnel a le contrôle exclusif de ce dispositif et joue le rôle de sémaphore, le dispositif ne pouvant être atteint que si l'on invoque l'instance correspondante du bloc fonctionnel.

L'avantage qu'il y a à utiliser les instances de blocs fonctionnels est que la fonctionnalité associée à une structure de données définie ne doit être déclarée qu'une seule fois et peut être utilisée indépendamment dans plusieurs instances dans le programme d'un automate programmable. Ce «prototype» est conservé dans le type de bloc fonctionnel, et il peut resservir autant de fois qu'il est nécessaire quand on déclare les instances de ce type. L'utilisateur a donc la garantie qu'il n'y a pas d'erreurs dans aucune des instances de blocs fonctionnels tant qu'il n'y a pas d'erreurs dans le type de bloc fonctionnel associé.

Etant donné que le jeu complet de données d'état de l'instance en cours est facilement accessible pour une surveillance et une correction en ligne, les instances de blocs fonctionnels peuvent être une aide aux essais et au débogage.

Les instances de blocs fonctionnels sont un accessoire spécial des blocs fonctionnels tels qu'ils sont définis dans la CEI 61131-3. Il n'y a pas d'équivalent dans les langages de programmation procéduraux comme Pascal ou C.

L'instanciation des blocs fonctionnels est une extension des possibilités des automates programmables actuels. Dans de nombreuses installations actuelles, il n'y a qu'un nombre fixe d'instances de chaque type disponible de blocs fonctionnels, et il n'est pas possible de créer des instances supplémentaires. Les blocs fonctionnels définis par l'utilisateur pour lesquels on peut créer sans limites des instances sont aussi une extension pour la plupart des systèmes en place.