ISaGRAF

Version 3.4

GUIDE UTILISATEUR

CJ INTERNATIONAL

Imprimé en France par CJ International.
3 Rue Hector Berlioz
F-38600 FONTAINE
Phone: 33 (0)4 76 26 87 30
Fax: 33 (0)4 76 26 87 39

Table des matières

A. MANUEL D'UTILISATION A-11

A.1 Mise en route A-12
A.1.1 Installation d'ISaGRAF A-12
A.1.2 Utiliser l'information en ligne A-15
A.1.3 Application d'exemple A-15

A.2 Gestion des projets A-19
A.2.1 Création et manipulation de projets A-19
A.2.2 Travailler avec plusieurs groupes de projets A-21
A.2.3 Options A-21
A.2.4 Outils A-22

A.3 Gestion des programmes A-23
A.3.1 Les composants d'un projet A-23
A.3.2 Manipuler les programmes A-25
A.3.3 Les outils de génération de code A-28
A.3.4 Les autres outils ISaGRAF A-29
A.3.5 Ajouter des commandes au menu "Outils" A-30
A.3.6 Simuler et tester l'application A-30

A.4 L'éditeur SFC A-33
A.4.1 Bases du langage SFC A-33
A.4.2 Saisie du graphe SFC A-35
A.4.3 Travailler sur un graphe existant A-37
A.4.4 Entrer la programmation du niveau 2 A-38
A.4.5 Utiliser la galerie SFC A-41

A.5 L'éditeur Flow Chart A-43
A.5.1 Bases du langage FC A-43
A.5.2 Saisie du diagramme A-44
A.5.3 Travailler sur un diagramme existant A-46
A.5.4 Entrer la programmation du niveau 2 A-47
A.5.5 Programmation du niveau 2 avec le Quick LD A-48

A.5.6 Options d'affichage A-49

A.6 L'éditeur Quick LD A-50
A.6.1 Bases du langage LD A-50
A.6.2 Saisie du diagramme A-52
A.6.3 Travailler sur un diagramme existant A-55
A.6.4 Options d'affichage A-56

A.7 L'éditeur FBD/LD A-58
A.7.1 Bases des langages FBD et LD A-58
A.7.2 Saisie du diagramme A-60
A.7.3 Travailler sur un diagramme existant A-62
A.7.4 Options d'affichage A-64
A.7.5 Styles et marquage des modifications A-64

A.8 L'éditeur de texte A-66
A.8.1 Commandes d'édition A-66
A.8.2 Options A-67

A.9 Autres commandes des éditeurs de programmes A-68
A.9.1 Lancement des autres outils d'ISaGRAF A-68
A.9.2 Paramètres du programme A-68
A.9.3 Autres commandes du menu "Fichier" A-69
A.9.4 Mise à jour du fichier agenda A-70
A.9.5 Sélectionner une variable déclarée dans le dictionnaire A-70
A.9.6 La fenêtre de messages A-71

A.10 Le dictionnaire A-73
A.10.1 La fenêtre du dictionnaire A-75
A.10.2 Gestion des variables A-75
A.10.3 Description des objects A-77
A.10.4 Déclaration rapide A-78
A.10.5 Carte d'adressage pour les superviseurs Modbus A-79
A.10.6 Exchange avec d'autres applications A-80

A.11 Câblage des E/S A-85
A.11.1 Defininir les équipements d'E/S A-86
A.11.2 Entrer les paramètres d'une carte A-87
A.11.3 Câbler les voies d'E/S A-87
A.11.4 Variables représentées directement A-87

A.11.5 Numérotation A-88
A.11.6 Protections individuelles A-89

A.12 Tables de conversion A-90
A.12.1 Principales commandes A-90
A.12.2 Saisie des points d'une table A-91
A.12.3 Règles et limites A-91

A.13 Génération du code A-93
A.13.1 Principales commandes A-93
A.13.2 Options de compilation A-94
A.13.3 Génération de code source "C" A-96
A.13.4 Affichage A-96
A.13.5 Définition des ressources A-97

A.14 Références croisées A-103

A.15 Mise au point A-105
A.15.1 La fenêtre du debugger A-105
A.15.2 Contrôle de l'application A-106
A.15.3 Options A-108
A.15.4 Commandes de forçage A-108
A.15.5 Modification en ligne A-110
A.15.6 Echanges DDE A-113

A.16 Espionner des listes de variables A-115

A.17 Mise au point des programmes ST et IL A-117

A.18 Mise au point avec SpotLight A-118
A.18.1 Construire l'image graphique A-118
A.18.2 L'affichage en liste A-120
A.18.3 Définir le style d'un élément A-121
A.18.4 Commandes du menu "Fichier" A-121
A.18.5 Note pour les utilisateurs d'ISaGRAF V3.2 A-122

A.19 Télélecture A-123
A.19.1 Récuprérer un projet A-123
A.19.2 Paramétrage de la liaison A-123
A.19.3 Préparation d'un projet pour la télélecture A-124

A.19.4 Comment les sources compressées sont stockées dans la cible A-125
A.19.5 Utilisation de la mémoire sur la cible A-125
A.19.6 Le projet récupéré A-125
A.19.7 Compatibilité A-125

A.20 L'outil de diagnostic A-126

A.21 Le simulateur A-127
A.21.1 Liens avec le debugger A-127
A.21.2 Simulation des E/S A-127
A.21.3 Eléments de la librairie A-128
A.21.4 Options A-129
A.21.5 Enregistrer et restituer l'état des sorties A-129
A.21.6 L'analyseur de cycle A-129
A.21.7 Scripts de simulation A-130

A.22 Gestion des librairies A-138
A.22.1 Manipuler les éléments de librairie A-138
A.22.2 Configuration d'E/S A-140
A.22.3 Equipement complexe d'E/S A-141
A.22.4 Carte d'E/S A-142
A.22.5 Fonctions et blocs fonctionnels écrits en langage IEC A-143
A.22.6 Fonctions et blocs fonctionnel en "C" A-145
A.22.7 Fonctions de conversion A-145

A.23 Archivage A-147
A.23.1 Lancer l'archiveur A-147
A.23.2 Options A-148
A.23.3 Sauvegarder / Restituer A-148
A.23.4 Fichiers d'archive A-148

A.24 Impression du dossier A-150
A.24.1 Personnaliser la table des matières A-150
A.24.2 Options A-151

A.25 Protection par mot de passe A-154

A.26 Techniques avancées A-157
A.26.1 Au sujet des outils d'ISaGRAF A-157
A.26.2 Entrées/sorties verrouillées et virtuelles A-157

A.26.3 Validation de la liaison avec la cible A-160
A.26.4 Les répertoires ISaGRAF A-160
A.26.5 Table des symboles A-162
A.26.6 Limites de l'atelier ISaGRAF "LARGE" (WDL) A-166

B. RÉFÉRENCE DES LANGAGES B-169

B.1 Architecture du projet B-170
B.1.1 Programmes B-170
B.1.2 Opérations cycliques et séquentielles B-170
B.1.3 Fils SFC et sous-programmes FC B-171
B.1.4 Fonctions et sous-programmes B-171
B.1.5 Blocs fonctionnels B-173
B.1.6 Langages B-174
B.1.7 Règles d'exécution B-174

B.2 Objets communs à tous les langages B-176
B.2.1 Types de base B-176
B.2.2 Expressions constantes B-176
B.2.3 Variables B-178
B.2.4 Commentaires B-181
B.2.5 Définition de mots équivalents B-182

B.3 Langage SFC B-184
B.3.1 Format du graphe SFC B-184
B.3.2 Composants de base du graphe SFC B-184
B.3.3 Divergences et convergences B-186
B.3.4 Macro-étapes B-188
B.3.5 Actions dans les étapes B-189
B.3.6 Conditions attachées aux transitions B-195
B.3.7 Règles d'évolution du SFC B-196
B.3.8 Hiérarchie des programmes SFC B-197

B.4 Flow Chart B-199
B.4.1 Composants du langage FC B-199
B.4.2 Structures complexes B-202
B.4.3 Exécution du Flow Chart B-203
B.4.4 Règles syntaxiques B-203

B.5 Langage FBD B-204

B.5.1 Format du diagramme FBD B-204

B.6 Langage LD B-208
B.6.1 Barres d'alimentation B-208
B.6.2 Liaison multiple B-209
B.6.3 Contacts et relais B-210
B.6.4 L'énoncé RETURN B-216
B.6.5 Sauts et étiquettes B-216
B.6.6 Blocs en LD B-217

B.7 Langage ST B-219
B.7.1 Syntaxe du ST B-219
B.7.2 Expressions B-219
B.7.3 Appel de fonction ou de bloc fonctionnel B-220
B.7.4 Opérateurs booléens spécifiques au ST B-221
B.7.5 Enoncés du langage ST B-223
B.7.6 Extension au langage ST B-228

B.8 Langage IL B-234
B.8.1 Syntaxe du langage IL B-234
B.8.2 Opérateur du IL B-235

B.9 Opérateurs, fonctions et blocs fonctionnels B-242
B.9.1 OPERATEURS STANDARDS B-242
B.9.2 Blocs fonctionnels standards B-263
B.9.3 Fonctions standards B-281

C. MANUEL D'UTILISATION DE LA CIBLE C-323

C.1 Introduction C-324

C.2 Installation C-325

C.3 Mise en route de la cible ISaGRAF DOS C-326
C.3.1 Exécution d'ISaGRAF: ISA.EXE C-326
C.3.2 Fonctionalités spécifiques C-327

C.4 Mise en route de la cible ISaGRAF OS9 C-331
C.4.1 Exécution d'ISaGRAF mono-tâche: isa C-331
C.4.2 Exécution d'ISaGRAF multi-tâches: isaker, isatst, isanet C-332

C.4.3 Fonctionalités spécifiques C-336

C.5 Mise en route de la cible VxWorks C-341
C.5.1 Le gestionnaire des ressources systèmes: isassr.o C-341
C.5.2 Fonctionnalités communes de isa.o, isakerse.o et isakeret.o C-341
C.5.3 Exécution d'ISaGRAF mono-tâche: isa.o C-342
C.5.4 Exécution d'ISaGRAF multi-tâches: isakerse.o und isakeret.o C-344
C.5.5 Fonctionnalités spécifiques C-348

C.6 Mise en route de la cible ISaGRAF NT C-353
C.6.1 Exécution d'ISaGRAF C-353
C.6.2 Informations générales sur les options C-353
C.6.3 Fonctionnalités spécifiques C-357
C.6.4 Interface utilisateur C-362

C.7 Programmation "C" C-368
C.7.1 Présentation C-368
C.7.2 Fonctions de conversion C-369
C.7.3 Fonctions "C" C-375
C.7.4 Blocs fonctionnels en "C" C-383
C.7.5 Techniques de compilation et d'intégration C-399

C.8 La liaison Modbus C-406
C.8.1 Réseau et protocole MODBUS C-406
C.8.2 Implémentation d'ISaGRAF C-407

C.9 Gestion des pannes de courant C-413
C.9.1 Bases C-413
C.9.2 Sauvegarde des variables de l'application C-414
C.9.3 Sauvegarde de l'état d'un programme C-417

C.10 Annexe: Liste d'erreurs et description C-419

D. GLOSSAIRE D-429

E. INDEX GÉNÉRAL E-439

A. Manuel d'utilisation

A.1 Mise en route

Ce chapitre explique comment **installer** l'atelier ISaGRAF sur votre machine. Il inclut aussi un exemple complet d'application ISaGRAF, offrant ainsi un point de vue complet sur ses principales fonctionnalités, et permettant la prise en main immédiate de ISaGRAF.

A.1.1 Installation d'ISaGRAF

Ce chapitre présente l'installation de l'atelier ISaGRAF sur une machine, et la configuration de cette machine pour le développement d'applications.

Configuration matérielle et logicielle

L'atelier ISaGRAF peut être installé sur toute machine supportant la version 3.1 de Windows. Toutefois, le matériel suivant est vivement conseillé pour le développement d'applications:

- Un PC ou compatible utilisant un microprocesseur 80486 (ou plus) (Pentium recommandé)
- 8 MOctets de **mémoire** conventionnelle et étendue (16 Moctets recommandé)
- Un lecteur de disquettes 3,5 pouces (1.44 méga-octet)
- Un **disque** dur avec au moins 20 MOctets d'espace libre
- Un adaptateur graphique VGA ou SVGA et un moniteur compatible
- Une souris (nécessaire pour les outils d'édition graphique)
- Un port parallèle LPT1 (requis pour la clé de protection)

Avant d'installer l'atelier ISaGRAF, assurez-vous que l'un des logiciels suivants est présent sur votre système:

- Windows Version 3.1 en mode 386 étendu
- Windows 95
- Windows NT Version 3.51 ou 4.00

Le programme d'installation

L'atelier ISaGRAF est installé par INSTALL, le programme d'installation d'ISaGRAF. Ce programme copie le logiciel ISaGRAF depuis le CD-ROM ou les disquettes sur le disque dur. INSTALL ajoute également le groupe "ISaGRAF" dans le gestionnaire de programmes de Windows (ou menu "Démarrer") , et crée le fichier de paramètres "ISA.ini" dans le sous-répertoire **EXE** installé avec ISaGRAF.

INSTALL est une application Windows qui peut être lancé depuis le Gestionnaire de Programmes de Windows ou le menu "Démarrer" de Windows 95. Pour installer ISaGRAF, les opérations suivantes doivent être effectuées:

- Insérer le CD-ROM ou la première disquette d'ISaGRAF dans le lecteur approprié

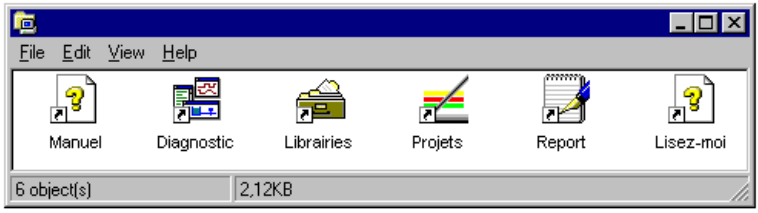
• Depuis le Gestionnaire de programmes ou le menu "Démarrer", lancer "SETUP.EXE" sur la racine du CD-ROM, ou "A:\INSTALL.EXE" dans le cas d'une disquette.
• Suivez les instructions de la procédure d'installation. Il est recommandé d'installer ISaGRAF sur un nouveau répertoire afin d'éviter toute confusion avec une version antérieure.

INSTALL vous demande si les composants suivants doivent être installés:

• Les programmes exécutables d'ISaGRAF
• Les fichiers d'aide et d'information en ligne
• Les librairies standard d'ISaGRAF
• Les applications ISaGRAF d'exemple

Il est fortement recommandé, pour une première installation de l'atelier ISaGRAF, d'installer tous les composants. Toutefois, un composant pourra être ajouté plus tard par une réinstallation de l'atelier ISaGRAF.

Le nom proposé pour le répertoire principal d'ISaGRAF est "\ISAWIN". ISaGRAF peut donc facilement être installé sur le même disque qu'une version antérieure de ISaGRAF pour MS-DOS. Référez-vous à la section "Les répertoires d'ISaGRAF" du chapitre "Techniques avancées" pour de plus amples informations sur l'architecture des répertoires ISaGRAF sur le disque. Quand tous les fichiers ISaGRAF sont copiés, le **groupe** suivant est ajouté à la fenêtre du Gestionnaire de Programmes de Windows:



Voici les icones du groupe ISaGRAF :

Projets:..... Gestion des projets
Librairies:..... Gestion des librairies
Manuel:..... Aide en ligne d'ISaGRAF
Diagnostic:..... Outil de diagnostic pour l'utilisateur final
Lisez Moi:..... Informations décrivant la dernière version
Report: ..... Format standard de rapport d'incident

Dans le cas où vous rencontrez un problème, utilisez le format standard de rapport d'incident. Ouvrez l'icône Report, remplissez les champs requis et utilisez la commande "Fichier / Enregistrer sous..." pour sauver ce rapport avec un nouveau nom. Ensuite envoyez ce fichier à CJ International, par Fax ou e-mail.

Mise à jour des fichiers système

Quand l'installation est terminée vous devez mettre à jour le fichier système CONFIG.SYS et redémarrer votre machine. Le nom du répertoire où est installé ISaGRAF n'a pas besoin d'être inséré dans la variable d'environnement PATH. Le système ISaGRAF n'utilise aucune variable d'environnement de MS-DOS. Cependant, les énoncés suivants peuvent être ajoutés dans CONFIG.SYS:

```
files=20
buffers=20
```

L'atelier ISaGRAF utilise un port de communication série pour le dialogue avec l'automate cible ISaGRAF. Le port série proposé par défaut par ISaGRAF est COM1. Si votre souris utilise une communication série, il est préférable de la connecter sur le port COM2. Le choix par défaut de COM1 sera ainsi valide pour toute nouvelle application développée avec ISaGRAF.

Il est nécessaire, après la mise à jour du fichier CONFIG.SYS, de redémarrer le système afin que les modifications apportées soient prises en compte.

⇒ **Important pour les utilisateurs de Windows NT:**
Quand l'atelier est utilisé sous Windows NT 3.51 ou 4.00, la ligne suivante doit être insérer dans la section [WS001] dans le fichier ISA.INI situé dans le répertoire \ISAWIN\EXE:

```
WS001]
NT=1
Isa=C:\ISAWIN
IsaExe=C:\ISAWIN\EXE
IsaApl=C:\ISAWIN\APL1
IsaTmp=C:\ISAWIN\TMP
```

Ceci est absolument nécessaire pour la communication série.

La clé de protection

Une **clé de protection** matérielle protège le logiciel ISaGRAF contre la copie illégale. Toutefois, la plupart des fonctions de l'atelier ISaGRAF sont encore disponibles quand la clé n'est pas connectée. La clé de protection définit aussi l'option du logiciel ISaGRAF que vous possédez, et détermine la taille maximum des applications développées. Certaines fonctions de l'atelier ISaGRAF ne sont plus accessibles quand la clé est absente ou mal connectée. C'est un comportement NORMAL. Pour vous assurer que la clé est correctement installée, sélectionnez le choix "**A propos**" du menu "**Aide**" dans une des fenêtres ISaGRAF. Le nom de la version installée de ISaGRAF est alors affiché.

La clé de protection peut être connectée sur tout port parallèle. Si la machine dispose de plusieurs ports parallèles, il est préférable de connecter la clé et l'imprimante sur des ports différents. Dans certaines configurations de PC et d'imprimante, la clé peut ne pas être reconnue quand elle est connectée à une imprimante "hors ligne". Dans ce cas, il suffit de déconnecter l'imprimante ou de la positionner dans l'état "en ligne", et de redémarrer l'atelier ISaGRAF.

Notez que la clé n'est pas nécessaire pour l'atelier **ISaGRAF-32**.

⇒ **Important pour les utilisateurs de Windows NT:**

Sur les systèmes Windows **NT**, le driver Sentinel/ Rainbow™ doit être installé pour que la **clé** soit reconnue par ISaGRAF. Ces drivers sont sur le répertoire "SENTINEL" du CD-ROM ISaGRAF.

A.1.2 Utiliser l'information en ligne

L'information en ligne est installée avec l'atelier ISaGRAF, elle concerne:
- le manuel de référence des langages ISaGRAF
- le manuel d'utilisation complet (pour chaque outil ISaGRAF)
- la notice technique des éléments des librairies

Depuis toute fenêtre ISaGRAF, sélectionnez les choix du menu "**Aide**" pour consulter l'information en ligne.

A.1.3 Application d'exemple

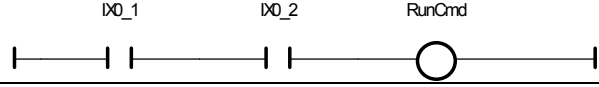
Ce chapitre présente point par point toutes les opérations à effectuer pour concevoir, générer et tester une application simple et complète réalisée à l'aide de plusieurs langages.

Voici les spécifications complètes de l'application, qui utilise les représentations en LD et SFC:

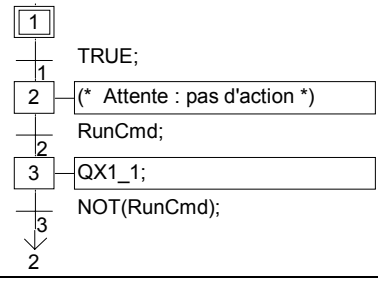
Variables booléennes:

IX0_1, IX0_2:	variables d'entrée pour la commande du process
RunCmd:	commande "Marche/Arrêt" interne
QX1_1:	variable de sortie: état du process

Programme Command: Exécution cyclique (début)- langage LD
Evalue la commande "Marche/Arrêt"



Programme RunStop: section séquentielle - langage SFC
Contrôle du process



Start ***Lancer l'atelier ISaGRAF***

Pour lancer l'atelier ISaGRAF, lancer la commande "Projets" dans le groupe "ISaGRAF", depuis le menu Démarrer de Windows.

Créer le projet

Créez le projet (de nom "RunStop") en utilisant la commande "Nouveau" du menu "Fichier" ou le bouton Nouveau. Dans la boite de dialogue:

Entrez le nom du projet: "**RunStop**"
Sélectionnez la configuration d'E/S: "**Sim_Boo**"
Pressez le bouton "**OK**".
Le projet est créé.

Ouvrir le projet

Pour définir les programmes du projet, il faut ouvrir le Gestionnaire de Programmes d'ISaGRAF. Utilisez la commande "Ouvrir" du Gestionnaire de Projets, ou double-cliquez sur le nom du projet "RunStop" ou utilisez le bouton "Editer".

Créer les programmes

La fenêtre du Gestionnaire de Programmes est ouverte. Elle est vide (aucun programme défini). Utilisez la commande "Nouveau" du menu "Fichier" ou le bouton Nouveau. Dans la boite de dialogue:

Entrez le nom du programme: "**Command**".
Sélectionnez le langage "**Quick LD**".
Sélectionnez la section "**Début de cycle**".
Pressez le bouton "**OK**" pour créer le programme.

La même opération doit être répétée pour le second programme:
Utilisez la commande "Nouveau" du menu "Fichier" ou le bouton Nouveau. Dans la boite de dialogue:

Entrez le nom du programme: "**RunStop**".
Sélectionnez le langage "**SFC**".
Sélectionnez la section "**Séquentiel**".
Pressez le bouton "**OK**" pour créer le programme.

Les deux programmes sont maintenant créés. Ils apparaissent dans la fenêtre du Gestionnaire de Programmes.

Déclarer les variables

Avant de saisir le code des programmes, il faut déclarer les variables internes qui seront utilisées. Pour ceci, utilisez la commande "Dictionnaire" du menu "Fichier" ou le bouton Dictionnaire. Les variables d'E/S sont automatiquement déclarées à la création du projet.

La fenêtre du dictionnaire est ouverte. Utilisez dans le menu "Fichier", le sous-menu "Autre", et le sous-menu "Variables globales" puis la commande "Booléen", pour afficher le dictionnaire des variables globales booléennes. Les boutons Objets globaux et Booléen peuvent être utiliser pour le même résultat.

Utilisez la commande "Nouveau" du menu "Fichier" pour créer de nouvelles variables booléennes. Vous pouvez aussi utiliser le bouton Insérer objet. Dans la boite de dialogue, entrez la description des variables internes:

nom:	**RunCmd**
commentaire:	**Marche/Arrêt: commande interne**
attribut:	Sélectionner l'attribut "**Interne**"

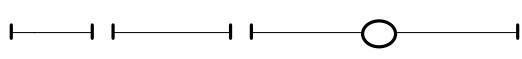
Pressez le bouton "**Enregistre**": la variable est créée.
Pressez le bouton "**Abandonne**" pour fermer la boite de dialogue.
Enfin, sortez du dictionnaire et enregistrez les modifications:
Menu "**Fichier**" - Commande "**Quitter**". Cliquer sur "**OUI**" pour enregistrer les modifications.

Editer le programme Quick LD

Pour lancer l'édition du programme "Command", double-cliquez sur son nom dans la fenêtre du Gestionnaire de Programmes ou utilisez le bouton Editer.

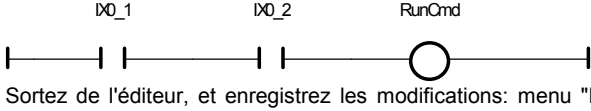
La fenêtre d'édition Quick LD est ouverte. Pour augmenter l'espace de saisie, dimensionnez la fenêtre en plein écran:

F2 F3 Pressez les touches F2 et F3:



Pour associer des variables aux symboles LD, déplacez le curseur en utilisant les flèches du clavier. Placez le curseur sur chaque symbole et pressez la touche Entrer (Retour Chariot). La boite de sélection de variables est ouverte.
Pour le premier contact, saisir le nom: IX0_1 puis Entrer
Pour le second contact, saisir le nom: IX0_2 puis Entrer
Pour le relais, saisir le nom: RunCmd puis Entrer.

Le programme est maintenant complet. Voici le résultat:



Sortez de l'éditeur, et enregistrez les modifications: menu "**Fichier**" - Commande "**Quitter**". Cliquez sur "**OUI**" pour enregistrer les modifications.

Editer le programme SFC

Pour lancer l'édition du programme SFC "RunStop", double-cliquez sur son nom dans la fenêtre du Gestionnaire de Programmes ou utilisez le bouton Editer.

La fenêtre d'édition SFC est ouverte. Pour augmenter l'espace de saisie, dimensionnez la fenêtre en plein écran.

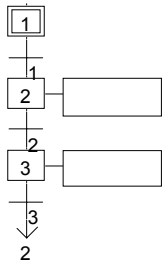
L'étape initiale est déjà présente. Frappez la flèche vers le bas du clavier pour sélectionner la cellule en bas de l'étape initiale (0,1)

F4 F3 Frappez F4 puis F3 pour insérer une étape et une transition.

F4 F3 Frappez F4 puis F3 pour insérer encore une étape et une transition.

F5 Frappez F5 pour insérer un saut vers une étape, puis sélectionnez GS2 comme destination du saut.

Le schéma est maintenant complet. Utilisez le bouton "Zoom" dans la barre d'outils pour augmenter la taille des cellules et permettre l'aperçu des instructions dans le niveau 2. Voici le schéma:



↵ Pour entrer la programmation de la transition "2", sélectionnez-la à l'aide des flèches du clavier et frappez la touche "Entrée". La fenêtre de programmation du niveau 2 est ouverte. Entrez la condition pour la transition 2:

RunCmd;

^TAB Frappez les touches "Contrôle + Tab" pour activer la fenêtre SFC, déplacez la sélection sur l'étape 3, et frappez "Entrée" pour entrer le texte du niveau 2:

QX1_1;

Et suivez la même procédure pour la transition 3:

Not (RunCmd);

^F4 Frappez les touches "Contrôle + F4" pour fermer la fenêtre du niveau 2.
Le schéma SFC est complet. Quittez l'éditeur à l'aide du menu "**Fichier**" et de la commande "**Quitter**", et sauvez les modifications saisies en sélectionnant "**OUI**".

Générer le code de l'application

Utilisez la commande "**Générer l'application**" du menu "**Codage**" ou le bouton correspondant dans la barre d'outils de la fenêtre du Gestionnaire de Programmes pour générer le code de l'application.

Quand la génération est terminée, une boîte de dialogue vous propose de fermer la fenêtre de compilation ou de continuer. Pressez le bouton "**Fermer**".

Simulation

Utilisez le menu "**Test**" et la commande "**Simuler**" ou le bouton dans la barre d'outil de la fenêtre du Gestionnaire de Programmes pour lancer le simulateur.
Quand la fenêtre du simulateur apparaît, l'application peut être testée. Dans cet exemple, les deux entrées (boutons verts) doivent être pressées pour activer le process (sortie: led rouge).

Fermez la fenêtre du **Debugger** pour quitter la simulation: Menu "**Fichier**" - Commande "**Quitter**".

A.2 Gestion des projets

Pour lancer le gestionnaire de projets d'ISaGRAF, cliquez deux fois sur l'icône "Projets" dans le groupe ISaGRAF. La fenêtre du gestionnaire de projets est alors ouverte. Un projet correspond à un cycle automate exécuté sur un automate cible. La fenêtre supérieure montre la liste des projets existants. Le descripteur du projet sélectionné est affiché dans la fenêtre inférieure.

Redimensionner les fenêtres

Cliquez avec la souris sur la ligne de séparation entre la liste des projets et le descripteur pour redimensionner les fenêtres. Le descripteur ne peut pas être complètement caché. Il contient toujours au moins une ligne de texte visible.

Insérer des sépérateurs

Une ligne de séparation peut être insérée avant n'importe quel projet de la liste. Ceci permet de montrer les projets d'une même application dans un même groupe de la liste. Utilisez la commande "**Edition / Séparateur**" pour insérer ou supprimer le séparateur avant la ligne du projet sélectionné.

Déplacer un projet dans la liste

Pour déplacer un projet dans la liste, vous devez d'abord le sélectionner. Puis cliquez sur son nom et faites-le glisser vers une autre position. Lors du déplacement, une flèche dans la marge gauche de la liste indique la nouvelle position. Vous pouvez aussi utiliser les commandes "**Déplacer**" du menu "**Edition**" pour déplacer le projet sélectionné ligne par ligne. S'il existe, le séparateur placé avant le projet est déplacé en même temps que le projet.

A.2.1 Création et manipulation de projets

Les commandes du menu du Gestionnaire de Projets permettent de créer de nouveaux projets, d'éditer (ouvrir) ou de manipuler les projets existants.

Créer un nouveau projet

Pour créer un nouveau projet, il faut tout d'abord entrer son nom. Un projet vide est alors créé. Une configuration d'E/S peut également être sélectionnée lors de la création du projet. Cette configuration doit être définie en librairie. Si une configuration est choisie, ISaGRAF crée automatiquement les variables correspondantes et leur câblage dans le nouveau projet. Quand vous créez ou renommez un projet, vous devez respecter les règles suivantes:

- le nom ne peut pas excéder **8** caractères
- le premier caractère doit être une **lettre**
- les caractères suivants doivent être des **lettres**, **chiffres** ou le caractère de soulignement
- les majuscules et minuscules ne sont pas différenciées

Quand un projet est créé, utilisez la commande "**Edition / Commentaire du projet**" pour entrer le texte qui sera affiché avec le nom du projet dans la liste des projets.

Editer le descripteur d'un projet

Utilisez la commande "**Projet / Editer le descripteur**" pour entrer le descripteur du projet sélectionné. Ce document identifie de façon détaillée le projet. Il peut également accueillir toute remarque ou complément d'information pendant le cycle de vie du projet.

Editer un projet

La commande "**Fichier / Ouvrir**" ouvre le Gestionnaire de Programmes pour le projet sélectionné. De cette fenêtre seront appelés les différents outils d'édition, de compilation et test. Vous pouvez également cliquer deux fois sur le nom du projet dans la liste pour l'éditer.

L'historique des modifications

L'atelier ISaGRAF enregistre toutes les modifications concernant un composant d'un projet dans un fichier historique. Chaque modification est identifiée dans l'historique par un titre, une date et une heure. Le fichier historique regroupe les **500** dernières modifications. Il y a un fichier historique pour chaque projet. La commande "**Projet / Historique des modifications**" permet à l'utilisateur de visualiser ou d'imprimer le contenu du fichier historique pour le projet sélectionné. L'utilisateur peut sélectionner un ou plusieurs items dans la liste, et utiliser les commandes suivantes:

OK	ferme cette fenêtre
Imprimer	imprime le contenu de la liste
[effacer] Sélection	retire (efface) les lignes sélectionnées de la liste
[effacer] Tout	efface toute la liste
Chercher	cherche un texte dans la liste

Le champ de saisie au dessus du bouton "**cherche**" est utilisé pour entrer la chaîne de caractères à rechercher. La recherche ne différencie pas les majuscules des minuscules. Quand la recherche atteint la fin de la liste, elle continue à partir du début de la liste, jusqu'à la position de départ.

Imprimer le dossier

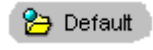
Cette fonction permet de construire et d'imprimer une documentation complète pour le projet sélectionné. Ce document peut grouper tous les composants (programme, variable, paramètres...) du projet. Pour construire un document spécifique (incomplet), il suffit de définir sa table des matières.

Protection par mot de passe

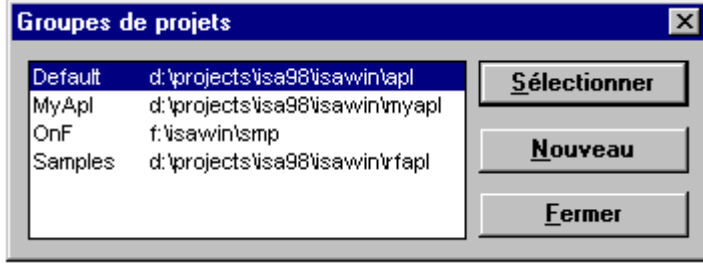
La fonction "**Mot de passe**" du menu "**Fichiers**" permet de protéger l'accès aux fonctions et aux données du projet à l'aide de mots de passe. Consultez la section "**Protection par mots de passe**" à la fin de la première partie de ce document pour plus de détails sur le système de protection et les niveaux d'accès. Les mots de passe sont relatifs au projet sélectionné, et n'ont aucune influence sur les autres projets et les librairies.

A.2.2 Travailler avec plusieurs groupes de projets

Un projet ISaGRAF correspond à un répertoire sur le disque dur, où sont stockés tous les fichiers du projet. Un "groupe de projets" correspond à une liste de projets dont les répertoires sont stockés sous le même répertoire "racine". Un groupe de projets est identifié par un nom. Par défaut ISaGRAF crée les groupes suivants:
"Default" dans "\ISAWIN\APL" votre espace de travail
"Samples" sans "\ISAWIN\SMP" les exemples livrés avec ISaGRAF
Le nom du groupe sélectionné est marqué dans la barre d'outils du gestionnaire de projets, à côté du bouton qui sert à sélectionner un autre groupe:



Vous pouvez également utilisez la commande "**Fichier / Sélectionner un groupe**" pour changer de groupe ou créer un nouveau groupe. La boîte de dialogue suivante est ouverte:



Sélectionnez un groupe dans la liste et pressez le bouton "**Sélectionner**" pour le prendre en compte dans le gestionnaire de projets. Pressez le bouton "**Nouveau**" pour créer un nouveau groupe. Cette commande peut être utilisée soit pour assigner un nom de groupe à un répertoire existant, ou pour créer un nouveau répertoire.
Aucun groupe ne peut être sélectionné ou créé si une autre fenêtre de l'atelier ISaGRAF (gestionnaire de programmes, éditeurs...) est ouverte.

A.2.3 Options

Les commandes du menu "**Options**" permettent d'afficher ou de cacher la barre d'outils, de sélectionner la police de caractères utilisées par les éditeurs de texte d'ISaGRAF, et de positionner le mode de fermeture automatique de la fenêtre du Gestionnaire de Projets.
Quand l'option "**Laisser le gestionnaire de projets ouvert**" n'est pas validée, la fenêtre est automatiquement fermée quand un projet est ouvert.

A.2.4 Outils

Les commandes du menu "**Outils**" permettent de lancer les autres outils d'ISaGRAF. Utilisez la commande "**Outils / Archiver / projet**" pour sauvegarder ou restituer des projets, la commande "**Outils / Archiver / Données communes**" pour sauvegarder ou restituer les données communes à tous les projets.
La commande "**Outils / Librairies**" lance le gestionnaire de librairies d'ISaGRAF.
La commande "**Outils / Importer un programme IL**" permet d'importer un programme complet en IL respectant le format d'échange défini par PLC Open.

A.3 Gestion des programmes

La fenêtre du Gestionnaire de Programmes montre les **programmes** (également appelés modules ou unités de programmation) de l'application et regroupe dans ses menus les commandes disponibles pour mettre en place l'architecture du projet, lancer les éditeurs, les compilateurs et les outils de mise au point. Cette fenêtre est le noyau interactif de l'atelier ISaGRAF pendant le développement d'un projet. La fenêtre du Gestionnaire de Programmes est ouverte par la commande "**Ouvrir**" de la fenêtre du Gestionnaire de Projets.

A.3.1 Les composants d'un projet

Les composants d'une application sont les **programmes**. Un programme est une entité logique qui décrit un sous-ensemble de la loi de commande. Des variables globales (telles que les variables d'E/S) peuvent être utilisées par tous les programmes de l'application. Les variables locales ne peuvent être utilisées que par un seul programme. Les programmes sont organisés dans un "**arbre de hiérarchie**", divisé en "**sections**" logiques. La fenêtre du gestionnaire de programmes montre les programmes et leurs liens. Les programmes principaux apparaissent sur la gauche de l'arbre de hiérarchie.

Programmes principaux

Les programmes principaux apparaissent sur la gauche de l'arbre de hiérarchie. Les programmes principaux des trois premières sections sont toujours actifs, et sont exécutés dans l'ordre suivant pendant le cycle automate:

- (Lecture des entrées)
- Exécution des programmes principaux de la section **DEBUT**
- Exécution des programmes principaux de la section **SEQUENTIEL**
- Exécution des programmes principaux de la section **FIN**
- (Mise à jour des sorties)

Les programmes des sections "**Début**" et "**Fin**" décrivent des opérations cycliques, indépendantes de l'évolution du procédé dans le temps. Les programmes de la section "**Séquentiel**" décrivent des traitements séquentiels, où l'évolution du procédé dans le temps apparaît explicitement pour séparer les groupes de traitements élémentaires. Les programmes principaux de la section "**Début**" sont exécutés systématiquement au début de chaque cycle. Les programmes principaux de la section "**Fin**" sont exécutés systématiquement à la fin de chaque cycle. Les programmes principaux de la section "**Séquentiel**" sont exécutés selon les règles d'évolution des langages **SFC** ou **FC**. Les programmes des sections cycliques ne peuvent pas être décrits en langage SFC ou FC. Tout programme de toute section peut avoir un ou plusieurs **sous-programmes**.

Fonctions et blocs fonctionnels

Les programmes de la section "**Fonctions**" peuvent être appelés par n'importe quel programme de n'importe quelle section dans le projet. Une fonction est un algorithme qui calcule une valeur de sortie en fonction de plusieurs valeurs d'entrée.

Une fonction ne travaille que sur des données volatiles, écrasées d'un appel à l'autre. Ceci implique qu'une fonction ne doit jamais appeler un bloc fonctionnel. Un programme de la section "**Fonctions**" ne peut pas être écrit en SFC ni en FC.
Contrairement aux fonctions, les "**Blocs fonctionnels**" associent un algorithme et des données statiques cachées qui sont copiées (instanciées) à chaque utilisation du bloc. Les programmes de la section "**Blocs fonctionnels**" peuvent être appelés par n'importe quel programme de n'importe quelle section du projet. Ils ne peuvent pas être décrit en SFC ni en FC.

Sous-programmes

Un sous-programme peut être dédié à un programme père (SFC ou FC ou autre). Un sous-programme ne peut être lancé que par son programme père. Chaque programme de chaque section peut avoir un ou plusieurs sous-programmes. Un sous-programme ne peut pas être écrit en SFC ni en FC.

Fils SFC et sous-programmes FC

Un **programme SFC fils** est un traitement parallèle qui peut être lancé, tué, figé ou relancé par son programme père. Le programme père et le programme fils doivent tous deux être décrits avec le langage **SFC**. Quand un programme père lance un programme **SFC** fils, il place un **jeton SFC** dans chacune de ses étapes initiales. Quand un programme père tue un programme **SFC** fils, il enlève tous les jetons existant dans ses étapes.
Tout programme **FC** de la section séquentielle peut contrôler des sous-programmes **FC**. L'exécution d'un programme père **FC** est suspendue (le flux est bloqué) pendant toute l'exécution du sous-programme **FC**, c'est-à-dire jusqu'à ce que le flux dans le sous-programme rencontre le symbole END. Il ne peut en aucun cas y avoir d'exécution simultanée d'un programme **FC** et d'un de ses sous-programmes **FC**.

Liens entre les programmes et les sous-programmes

Les sous-programmes et les programmes fils sont reliés à leur programme père par une ligne dans l'arbre de hiérarchie. Un lien entre deux programmes **SFC** père et fils est terminé par une flèche. Il convient de se rappeler qu'un tel lien représente des opérations **parallèles**.

Langages de programmation

Chaque programme est décrit avec un seul **langage**. Toutefois, le FBD et le LD peuvent être mélés dans le même diagramme. Le langage, choisi lors de la création du programme, ne peut plus être changé par la suite. Les langages graphiques disponibles sont le **SFC (**Sequential Function Chart ou GRAFCET), le **FC (**Flow Chart ou diagramme de flux), le **FBD** (Function Block Diagram ou Logigramme) et le **LD** (Ladder Diagram ou Schéma à contacts). Les langages textuels disponibles sont le **ST** (Structured Text) et le **IL** (Instruction List). Les langages **SFC** et **FC** sont réservés aux programmes principaux et fils de la section séquentielle. Le langage de chaque programme est montré sous la forme d'un icone à côté du nom du programme dans la fenêtre du Gestionnaire de Programmes. Voici la liste des icones utilisés:

 SFC ..... Sequential Function Chart (ou GRAFCET)
 FC ..... Flow Chart (diagramme de flux)

FBD ..... Functional Block Diagram (ou logigramme)
LD..... Ladder Diagram (schéma à contact entré avec l'éditeur Quick LD)
ST..... Structured Text (langage textuel structuré)
IL ..... Instruction List (liste d'instructions)

A.3.2 Manipuler les programmes

Le menu "**Fichier**" groupe les commandes de création et de mise à jour des programmes, ainsi que les commandes de lancement des éditeurs de programmes.

Créer un nouveau programme

La commande "**Nouveau**" du menu "**Fichiers**" vous permet de créer des programmes principaux, des fils ou des sous-programmes, dans chacune des sections. La première information à entrer est le nom du programme à créer, en respectant les règles de nomenclature suivantes:

- la longueur du nom ne peut pas excéder **8** caractères
- le premier caractère doit être une **lettre**
- les suivants doivent être des **lettres**, des **chiffres** ou le souligné ('**_**')
- les majuscules et les minuscules ne sont pas différenciées

Il faut ensuite choisir un langage pour le programme:
SFC..... Sequential Function Chart (GRAFCET)
FC ..... Flow Chart (diagramme de flux)
FBD..... Functional Block Diagram (logigramme)
LD ..... Ladder Diagram (schéma à contact) édité avec Quick LD
ST ..... Structured Text (langage textuel structuré)
IL..... Instruction List (liste d'instructions)

Vous devez finalement sélectionner un style d'exécution pour le programme:
Début:.....programme principal - section "début"
Séquentiel:.....programme principal - section "séquentiel"
Fin: .....programme principal - section "fin"
Fonction:.....programme de la section "fonctions"
Bloc fonctionnel: .....programme de la section "blocs fonctionnels"
Fils de:.....programme SFC fils ou sous-programme FC ou sous-programme

La sélection d'un des cinq premiers choix indique que le programme créé sera placé au niveau supérieur de la section **Début**, **Fin**, **Séquentiel**, **Fonctions** ou **Blocs fonctionnels**. La sélection du dernier choix permet la création d'un programme fils SFC ou d'un sous-programme FC ou d'un sous-programme. Un programme principal de la section **séquentiel** doit être décrit avec le langage **SFC** ou **FC**. Les langages **SFC** et **FC** ne peuvent pas être utilisés pour un programme d'une section cyclique (**Début** ou **Fin**).

Entrer un commentaire pour chaque programme

ISaGRAF permet d'associer à chaque programme un texte de description. Ce commentaire est affiché à côté du nom du programme. Utilisez la commande

"**Fichier / Commentaire du programme**" pour entrer ou modifier le texte du commentaire pour le programme sélectionné.

Editer le contenu d'un programme

Cette commande permet la modification du contenu d'un programme. L'outil ISaGRAF utilisé pour l'édition du programme dépend du langage choisi pour ce programme. L'édition est réalisée dans une fenêtre indépendante. Il est ainsi possible d'éditer plusieurs programmes en parallèle dans des fenêtres différentes. Vous pouvez également frapper directement la touche **ENTREE** pour éditer le programme sélectionné. Vous pouvez encore cliquer deux fois sur le nom du programme pour lancer son édition.

Editer le fichier "agenda"

Un **fichier agenda** est attaché à chaque programme. C'est un fichier de texte qui contient toutes les annotations concernant les modifications apportées au programme pendant son développement. Le fichier agenda peut être édité, librement modifié, ou imprimé à tout moment. Quand vous quittez l'édition du source d'un programme qui a été modifié, une fenêtre est automatiquement ouverte pour la saisie des annotations à stocker dans l'agenda. Ces annotations seront enregistrées avec la date et l'heure de leur saisie.

Le dictionnaire de variables

La commande "**Fichier / Dictionnaire**" lance l'édition du dictionnaire, où sont déclarées les variables du projet. Les variables peuvent être globales (connues par tous les programmes) ou locales à un programme. Le dictionnaire est également utilisé pour la déclaration de **mots équivalents**, noms qui remplacent une expression dans les sources des programmes.

Paramètres d'une fonction, d'un sous-programme ou d'un bloc

La commande "**Fichier / Paramètres**" permet de définir les paramètres d'appel et de retour du sous-programme, fonction ou bloc fonctionnel sélectionné. Cette commande est sans effet si un programme principal des sections "**Début**" ou "**Fin**" ou un programme SFC est sélectionné.
Un sous-programme, une fonction ou un bloc fonctionnel peuvent avoir jusqu'à **32** paramètres (entrées ou sorties). Une fonction ou un sous-programme a toujours un paramètre de retour et un seul, qui doit porter le nom de la fonction, conformément aux règles d'écriture du langage ST.
La liste dans la partie supérieure de la fenêtre montre les paramètres du sous-programme, selon l'ordre défini par le prototype d'appel du sous-programme: d'abord les paramètres d'appel, et enfin le paramètre de retour. La partie inférieure de la fenêtre montre la description détaillée du paramètre sélectionné dans la liste. Chacun des types de données d'ISaGRAF peut être utilisé pour un paramètre. Les paramètres de retour doivent être les derniers de la liste. Les règles suivantes doivent être respectées pour la nomenclature des paramètres:

- la longueur du nom ne doit pas excéder 16 caractères
- le premier caractère doit être une lettre
- les suivants doivent être des lettres, des chiffres, ou le souligné
- les majuscules et les minuscules ne sont pas différenciées

La commande "**Insère**" insère un emplacement libre pour un nouveau paramètre, avant le paramètre sélectionné. La commande "**Supprime**" efface le paramètre sélectionné. La commande "**Arrange**" trie les paramètres, de façon à ce que les paramètres de sortie soient placés en fin de liste.

Déplacer un programme dans la hiérarchie

La commande "**Fichier / Renommer/Déplacer**" permet de changer le nom d'un programme, ou de le déplacer dans l'arbre de hiérarchie. Toutefois le langage choisi pour la description du programme ne peut pas être changé. Quand cette commande est lancée, la même boîte de dialogue que celle utilisée pour créer un programme est ouverte. Tous ses champs sont positionnés avec la valeur courante des attributs du programme sélectionné. Le nom du programme peut être modifié. Une autre section ou un nouveau programme père peut être choisi pour déplacer le programme dans la hiérarchie.
La commande "**Fichier / Ordonner**" permet de donner un ordre explicite entre les programmes de même niveau de hiérarchie ayant le même père. Si le programme sélectionné est un programe principal, cette fonction permet de trier les programmes principaux de la section sélectionnée. Si le programme sélectionné est à un niveau inférieur dans la hiérarchie, cette commande permet de trier les sous-programmes ou programmes fils SFC ayant le même programme père. Quand la boîte de dialogue "**Ordonner**" est ouverte, les boutons "**Haut**" et "**Bas**" permettent de déplacer le programme sélectionné dans la liste affichée.

Copier des programmes

Pour copier un programme dans un autre, sélectionnez le programme source dans la fenêtre du Gestionnaire de Programmes et lancez la commande "**Fichier / Copier**". Quand cette commande est lancée, la même boîte de dialogue que celle utilisée pour créer un programme est ouverte. Tous ses champs sont positionnés avec la valeur courante des attributs du programme source sélectionné. Entrez le nom du programme de destination, et sélectionnez son emplacement dans les sections de l'arbre de hiérarchie. Si le programme de destination n'existe pas encore, il est créé à l'emplacement spécifié dans la hiérarchie. S'il existe déjà, son contenu est effacé avant la copie, et il est éventuellement déplacé dans la hiérarchie. Toutes les variables et définitions de mots équivalents locales au programme sont copiées avec lui. Le langage choisi pour le programme de destination doit obligatoirement être le même que celui du programme source.
La commande "**Copier dans un autre projet**" du menu "**Fichiers**" copie le programme sélectionné dans un autre projet, sous le même nom, et au sommet de la hiérarchie. Les programmes SFC fils et sous-programmes du programme sélectionné peuvent également être copiés avec lui. Les noms des programmes copiés ne doivent pas être déjà utilisés dans le projet de destination. Cette commande ne peut en aucun cas être utilisée pour écraser un programme existant. Toutes les déclarations et définitions de mots équivalents locales aux programmes manipulés sont copiées avec eux.

Supprimer des programmes

Pour supprimer un programme, vous devez le sélectionner dans la fenêtre du Gestionnaire de Programmes, et utiliser la commande "**Fichier / Supprimer**". Un programme ayant des sous-programmes ou des programmes fils ne peut pas être

supprimé. Il faut d'abord détruire ses fils. Toutes les variables et définitions de mots équivalents locales au programme sont supprimées avec lui.

Importer des fonctions ou des blocs de la librairie

La commande "**Outils / Importer de la librairie**" permet de copier une fonction ou un bloc fonctionnel de la librairie (écrit en langages IEC) dans un programme principal de la section "**Fonctions**" ou "**Blocs fonctionnels**" du projet. Les variables et définitions locales à la fonctions sont également importées. Si la fonction a déjà été compilée en librairie, son code objet est également importée dans l'environnement du projet. Quand une fonction a été importée de la librairie, elle peut ensuite être placée à un autre endroit dans l'arbre de hiérarchie du projet en utilisant la fonction "**Fichier / Déplacer/Renommer**". La fonction ou le bloc fonctionnel doit être renommé afin d'éviter les conflits de noms entre le projet et la librairie. N'oubliez pas de renommer également le paramètre de retour dans le cas d'une fonction.

Exporter des fonctions ou des blocs de la librairie

La commande "**Outils / Exporter vers la librairie**" copie un programme de la section "**Fonctions**" ou "**Blocs fonctionnels**" dans la librairie des fonctions ou blocs écrits en langages IEC. Les variables et définitions locales à la fonctions sont également exportées. La fonction exportée devra être re-vérifiée (compilée) depuis le Gestionnaire de Librairie pour s'assurer qu'elle puisse être correctement utilisée dans le contexte d'une librairie. Les fonctions de la librairie ne peuvent pas utiliser de variables globales.

A.3.3 Les outils de génération de code

Les commandes du menu "**Codage**" lancent les outils de compilation et de génération de code, et permettent d'entrer les options et autres paramètres utilisés pendant la génération du code de l'application. Référez-vous à la section "**Utiliser le générateur de code**" pour plus d'information sur ces outils.

Générer le code de l'application

La commande "**Générer l'application**" lance la génération du code. Les options pour les cibles choisies doivent être correctement renseignées avant la génération du code. Avant de générer le code, tous les programmes sont vérifiés et les éventuelles erreurs de syntaxes sont détectées. ISaGRAF inclut un compilateur incrémental qui ne vérifie que les programmes mofifiés depuis leur dernière vérification.

Vérifier le programme sélectionné

La commande "**Vérifier**" lance la vérification de la syntaxe du programme sélectionné dans la fenêtre du Gestionnaire de Programmes. Quand un programme est vérifié, et qu'aucune erreur n'a été détectée, il n'est plus revérifié automatiquement par la génération de code, tant que son contenu, ou les objets dont il dépend (variables, définitions...) ne sont pas modifiés.

Simuler une modification

La commande "**Simuler une modification**" marque tous les programmes comme "modifiés" de façon à ce qu'ils soient tous re-vérifiés durant la prochaine génération de code.

Les options d'exécution

Cette commande ouvre une boîte de dialogue où sont entrés les principaux paramètres d'exécution de l'application. Ceci inclut la programmation du temps de cycle, la gestion des erreurs d'exécution et le support matériel des variables non volatiles. Référez-vous à la section "**Utiliser le générateur de code**" pour plus d'information sur cette commande.

Options de compilation

Cette commande permet de sélectionner les options pour la génération et l'optimisation du code de l'application. Référez-vous à la section "**Utiliser le générateur de code**" pour plus d'information sur cette commande.

Définition des ressources

Une "**ressource**" est un ensemble de données définies par l'utilisateur (par exemple un fichier) qui doit être collé au code ISaGRAF et transféré avec lui. Référez-vous à la section "**Utiliser le générateur de code**" pour plus d'information sur cette commande.

A.3.4 Les autres outils ISaGRAF

Le menu "**Outils**" groupe les commandes de lancement des outils ISaGRAF pour le projet ouvert.

Câbler les entrées / sorties

La commande "**Câbler les E/S**" lance l'éditeur de câblage des variables d'E/S. Cet outil est utilisé pour décrire la correspondance entre les variables de l'application et les capteurs et actionneurs.

Les références croisées

La commande "**Références croisées**" permet de calculer, visualiser ou imprimer les références croisées de l'application. Les références croisées montrent les occurences de toutes les variables du dictionnaire dans tous les programmes du projet. Cette fonction est très utile pour la détection des effets de bord, et offre un point de vue global sur l'ensemble du dictionnaire de l'application.

Editer le descripteur du projet

La commande "**Descripteur du projet**" lance l'édition du fichier de texte contenant le descripteur du projet. Ce document contient l'identification détaillée du projet, ainsi que les remarques sur ses spécifications et son cycle de vie. Le descripteur de projet est affiché dans la fenêtre du Gestionnaire de Projets.

Imprimer le dossier

La commande "**Générer le dossier**" permet de construire et d'imprimer le dossier complet du projet. Ce document peut grouper tous les éléments (programmes, variables, paramètres...) du projet. Pour construire un document personnalisé (incomplet), il suffit d'établir sa table des matières.

L'historique des modifications

Cette commande ouvre une boîte de dialogue où est affiché l'historique des modifications apportées au projet. L'utilisation de cette boîte est détaillée dans la documentation du Gestionnaire de Projets d'ISaGRAF.

A.3.5 Ajouter des commandes au menu "Outils"

ISaGRAF permet d'ajouter d'autres commandes au menu "**Outils**". Les commandes supplémentaires doivent être décrites dans le fichier "**\ISAWIN\COM\ISA.MNU**" (texte). Vous pouvez ajouter jusqu'à 10 commandes. Des commentaires peuvent être insérés sur toute ligne, précédés par le caractère "**;**". Chaque commande est décrite sur deux lignes de texte, selon la syntaxe suivante:

```
M=texte_du_menu
C=ligne_de_commande
```

Le texte du menu est celui qui apparaitra dans le menu "**Outils**". La ligne de commande décrit l'appel d'un programme exécutable MS-DOS ou Windows, et peut être complétée par des arguments. Dans la ligne de commande, vous pouvez utiliser la séquence "**%A**" pour remplacer le nom du projet ouvert, et "**%P**" pour le nom du programme sélectionné. L'exemple suivant lance l'édition du programme sélectionné avec "Notepad" (pour les programmes ST et IL uniquement):

```
M=Edit with Notepad
C=Notepad.exe \isawin\apl\%A\%P.lsf
```

A.3.6 Simuler et tester l'application

Les commandes du menu "**Test**" lancent les outils de mise au point d'ISaGRAF, pour simuler l'application ou la tester en mode connecté.

Simulation

La commande "**Simuler**" lance le debugger pour la simulation. Dans ce mode, une autre fenêtre est ouverte, qui simule le comportement de l'automate cible. Cette commande permet de commencer les tests de l'application, même quand la machine cible n'est pas encore disponible. La fenêtre du Gestionnaire de Programmes est fermée, afin qu'elle puisse être ouverte à nouveau en mode "test" lorsque la liaison entre le debugger et le simulateur est établie. Le simulateur ne peut pas être lancé si le code de l'application n'est pas généré. Le simulateur ne peut pas être lancé tant que des fenêtres filles (éditeurs, générateur de code, câblage des E/S...) sont ouvertes. Fermez chacune de ces fenêtres avant d'utiliser le simulateur.

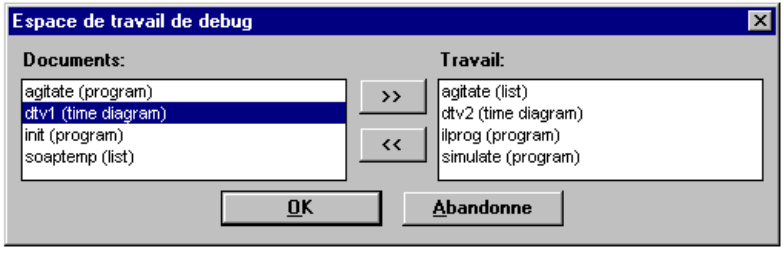
Mise au point en ligne

La commande "**Tester**" ouvre la fenêtre principale du debugger, et ferme la fenêtre du Gestionnaire de Programmes, afin qu'elle puisse être ouverte à nouveau en

mode "test" lorsque la liaison avec l'automate cible sera établie. Le debugger ne peut pas être lancé si le code de l'application n'est pas généré. Le debugger ne peut pas être lancé tant que des fenêtres filles (éditeurs, générateur de code, câblage des E/S...) sont ouvertes. Fermez chacune de ces fenêtres avant d'utiliser le debugger.

Préparer l'espace de travail pour le test

La commande "**Test / Espace de travail**" permet de définir la liste des documents à ouvrir quand la mise au point ou la simulation est lancée. Il peut s'agir de programmes, de graphiques SpotLight, de listes de variables. Les graphiques et listes de chronogrammes des versions antérieures d'ISaGRAF peuvent également apparaître dans la liste des documents à ouvrir.



La boîte de dialogue montre, à gauche, la liste des documents présents dans l'application, et à droite, les documents retenus pour l'espace de travail initial. Utilisez les boutons ">>" et "<<" pour déplacer un document d'une liste vers l'autre. Chaque projet à sa propre liste de documents pour l'espace de travail en mise au point.

Paramétrer la liaison

La commande "**Configurer la liaison**" permet de définir les paramètres de communication entre l'atelier de développement et l'automate ISaGRAF cible.
Le "**Numéro d'esclave**" identifie le système cible ISaGRAF, ou l'application cible dans le cas d'une implémentation multi-application sur la même machine. Le numéro d'esclave est compris entre **1** et **255**. Référez-vous au manuel fourni par le constructeur du système cible pour plus d'informations sur l'implémentation d'ISaGRAF.
Le "**Port de communication**" identifie le média de communication entre l'atelier ISaGRAF et la cible. Il peut être soit le nom d'un port série, soit "**Ethernet**", pour une communicationTCP-IP basée, pour l'atelier, sur l'interface "Winsock" Version V1.1.
Le "**Time out**" est le temps laissé au système cible pour ses opérations de communication, entre la fin d'une question émise par le debugger et le début de la réponse par la cible. Si aucune réponse n'est rendue après ce temps, un défaut de communication est diagnostiqué. Ce temps est exprimé en **secondes**. Le "nombre d'essais" est le nombre d'essais automatiques effectués par le debugger pour un même échange avant de diagnostiquer un défaut de communication.

Liaison série

Quand le port de communication sélectionné est un port série RS232 (COMM1..4), le bouton "**Configure**" permet d'accéder aux autres paramètres de la liaison série.
Le débit de transmission, la parité et le format peuvent être configurés. Quand l'option "**matériel**" est sélectionnée pour le "**Contrôle de flux**", l'atelier ISaGRAF gère les signaux CTS et DSR afin d'assurer un protocole hardware pendant les échanges

Liaison Ethernet

Quand le port de communication sélectionné est "Ethernet", le bouton "**Configure**" permet d'accéder aux autres paramètres de la liaison TCP-IP. Les champs "Adresse Internet" et "Port Ethernet" sont réservés pour la communication TCP-IP avec l'interface Winsock version 1.1. Le fichier WINSOCK.DLL doit être correctement installé sur votre le disque. "**1100**" est le numéro de port utilisé par défaut par la cible ISaGRAF.

A.4 L'éditeur SFC

Le langage SFC est utilisé pour décrire les opérations d'un procédé **séquentiel**. Il met en oeuvre une représentation **graphique** des différents **états** du procédé, et des conditions qui permettent de passer d'un état à un autre. Les programmes SFC sont saisis à l'aide de l'éditeur graphique SFC d'ISaGRAF. Le SFC est un langage de haut niveau. Les autres langages sont généralement utilisés pour décrire les **actions** dans les **étapes**, et les conditions attachées aux **transitions**. L'éditeur graphique SFC d'ISaGRAF permet la saisie de programmes SFC complets. Il combine des fonctions graphiques et textuelles pour la saisie du graphe SFC, et des actions et conditions correspondantes.

A.4.1 Bases du langage SFC

Le langage SFC décrit des comportements séquentiels. Il divise le procédé en un nombre d'étapes connues (situations stables), séparées par des transitions. Référez-vous au manuel de référence des langages d'ISaGRAF pour plus d'informations sur le langage SFC.
Les symboles d'un graphe SFC sont reliés par des **liaisons orientées**. L'orientation par défaut est **du haut vers le bas**. Voici les composants graphiques utilisés pour la construction d'un graphe SFC:

 ..... Etape initiale

 ..... Etape

 ..... Transition

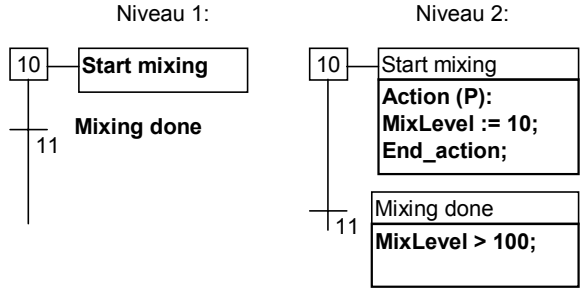
 ..... Renvoi à une étape

 ..... Macro-étape

 ..... Etape de début d'une macro

 ..... Etape de fin d'une macro

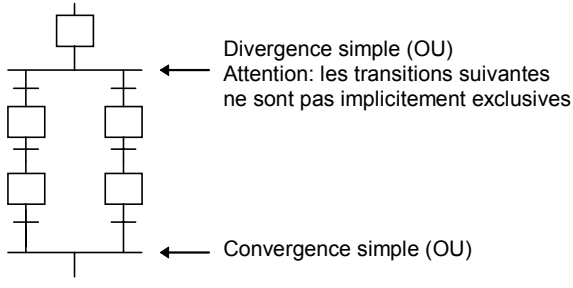
La programmation SFC se décompose en deux niveaux. Le **niveau 1** montre le graphe, les numéros de référence et les commentaires attachés aux étapes et aux transitions. Le **niveau 2** est la programmation en **ST** ou **IL** des actions dans les étapes et des conditions attachées aux transitions. Les actions et conditions peuvent appeler des **sous-programmes** écrits dans d'autres langages (**FBD**, **LD**, **ST** ou **IL**). Voici un exemple de programmation de niveau 1 et de niveau 2:



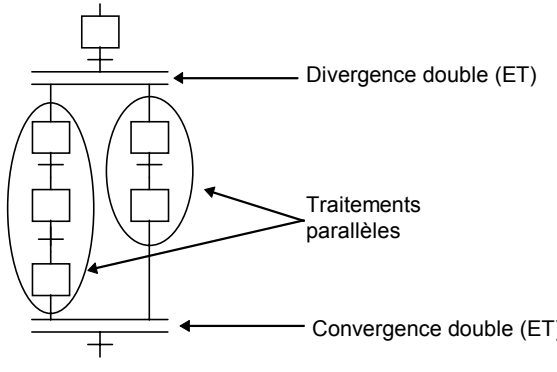
Le niveau 2 d'une étape est entré à l'aide d'un éditeur de textes. Il contient des blocs d'action écrits en ST ou IL. Le niveau 2 d'une transition peut être saisi soit en langage texte IL ou ST, soit avec l'éditeur Qquick LD (schéma à contacts).

Divergences et convergences

Les divergences et les convergences sont utilisées pour représenter des **liaisons multiples** entre les étapes et les transitions. Les divergences et convergences simples représentent plusieurs possibilités (**inclusives**) dans l'exécution du procédé.

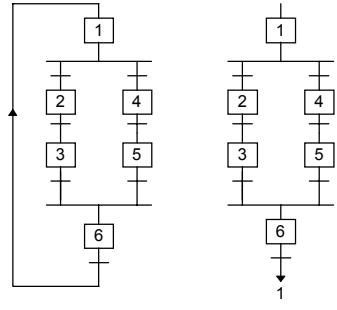


Les divergences doubles représentent des procédés **parallèles**.



Saut à une étape

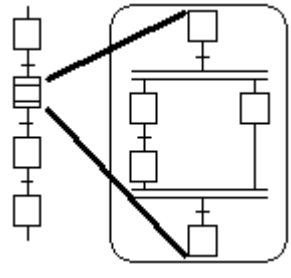
L'éditeur SFC ne visualise que les liaisons tracées du haut vers le bas. Un **renvoi** à une étape peut être utilisé pour représenter un renvoi vers une partie antérieure du graphe. Par exemple:



Un renvoi à une transition est interdit, et doit être explicitement représenté comme une convergence double.

Macro-étape

Une macro-étape est la représentation sous la forme d'un symbole unique d'un groupe **unique** et **connexe** d'étapes et de transitions. Le corps d'une macro-étape commence par une **étape de début** et se termine par une **étape de fin**.



La représentation de la macro-étape doit être décrite dans le même programme SFC. Le symbole de la macro-étape et son étape de début doivent avoir le même **numéro de référence**. Le corps d'une macro-étape peut contenir le symbole d'une autre macro-étape.

A.4.2 Saisie du graphe SFC

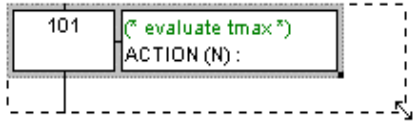
Pour entrer un graphe SFC, l'utilisateur doit poser les composants significatifs du schéma. Toutes les lignes de connexion horizontales et verticales sont tracées automatiquement par l'éditeur. Pour insérer un symbole SFC, il faut déplacer la sélection sur l'emplacement désiré et sélectionner le type de symbole dans la barre

d'outils. Le nouveau symbole est inséré à la position courante. Les touches de fonctions suivantes peuvent aussi être utilisées:

F2: ..... Insérer une étape initiale
F3: ..... Insérer une étape
F4: ..... Insérer une transition
F5: ..... Insérer un saut vers une étape
F6: F7: ..... Insérer une divergence ou convergence OU / Ajouter des branches
↑F6: ↑F7: ..... Insérer une divergence ou convergence ET / Ajouter des branches
F8: ..... Insérer une macro-étape
F9: ↑F9: ..... Insérer une étape de début ou de fin dans le corps d'une macro-étape

(Le symbole "↑" indique une combinaison combination avec la touche SHIFT)

La grille d'édition montre les **cellules** de la **matrice** d'édition graphique. Une option de l'éditeur permet de montrer ou de cacher la grille pendant l'édition du graphe. La grille est très utile pendant l'insertion des composants graphiques et la désignation de parties du schéma.
L'éditeur graphique SFC montre toujours la position courante dans la matrice. La cellule sélectionnée est marquée en gris. Le petit rectangle au coin bas / droite de la sélection permet redimensionner librement les cellules de la matrice. Il permet également de changer la proportion X/Y des cellules:



F6: F7: ↑F6: ↑F7: *Créer une divergence ou une convergence*

Les divergences et les convergences sont toujours tracées **de la gauche vers la droite**. Pour tracer une divergence ou une convergence, il faut d'abord poser ses branches à gauche. Le choix d'un de ces boutons dans la barre d'outils permet de choisir le type de divergence (simple ou double).

F6: F7: *Ajouter des branches à une divergence*

Les positions de **départ** et d'**arrivée** de chaque **branche auxiliaire** doivent être posées en utilisant ces boutons dans la barre d'outils. Les coins à droite ont le même style (simple ou double) que le coin principal à gauche. Vous ne pouvez pas poser les coins auxiliaires tant que le coin à gauche n'est pas défini.

F8: *Insérer une macro-étape*

Utilisez ce bouton dans la barre d'outil pour poser le symbole d'une macro-étape dans le graphe principal. Le corps de la macro-étape doit être décrit dans le même programme SFC.

Créer le corps d'une macro-étape

Les macro-étapes doivent être entrées dans le même programme que le graphe principal. Une macro-étape doit commencer par une **étape de début** et terminer par une **étape de fin**. Le graphe décrivant l'implémentation de la macro doit être **connexe**. L'étape de début de la macro doit avoir le même **numéro de référence** que le symbole de la macro-étape dans le graphe principal.

A.4.3 Travailler sur un graphe existant

Utilisez la souris ou le clavier (flèches combinées avec SHIFT) pour sélectionner un rectangle de la matrice. Ensuite, utilisez les commandes du menu "**Edition**" pour le manipuler.

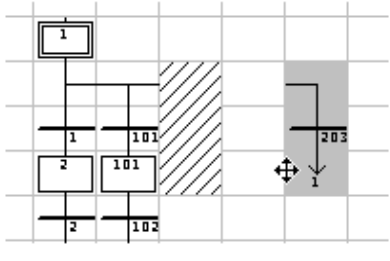
Couper / copier / supprimer / coller

Les commandes suivantes du menu "**Edition**" peuvent être utilisées quand le bouton "**sélection**" (flèche) est sélectionné dans la barre d'outils:
Couper ..... Envoie le bloc sélectionné dans le presse papier SFC
Copier ..... Copie le bloc sélectionné dans le presse papier SFC
Effacer..... Efface le bloc sélectionné

La commande "**Edition / Coller**" copie le contenu du presse papier SFC dans le schéma. Les commandes Copier / Coller manipulent le niveau 1 et le niveau 2 du SFC. Il est également possible de copier des portions d'un graphe SFC depuis un programme vers un autre.

Déplacer des éléments

Quand des éléments SFC sont sélectionnés dans le graphe, vous pouvez les déplacer en glissant la sélection avec la souris. Cette commande ne peut pas être réalisée avec le clavier. Pendant le déplacement, la position d'origine est hachurée dans le schéma.



La zone de destination choisie doit être vide. Le déplacement d'objets ne permet pas l'insertion des symboles sélectionnés dans une branche du graphe.

Renuméroter les étapes et les transitions

Chaque étape ou transition est identifiée par un numéro logique dans le graphe SFC. La commande "**Edition / Renuméroter**" permet d'affecter automatiquement tous les numéros de référence pour le graphe édité. Quand un numéro d'étape est changé, tous les renvois ayant cette étape pour destination sont également mis à jour avec le nouveau numéro de l'étape. (ceci est également appliqué aux macro-étapes et étapes de début)

Atteindre une étape ou une transition

La commande "**Edition / Atteindre**" permet d'accéder directement à une étape ou une transition du schéma. La position de déroulement de la fenêtre d'édition est automatiquement mise à jour pour que l'étape ou la transition soit visible.

Chercher et remplacer du texte

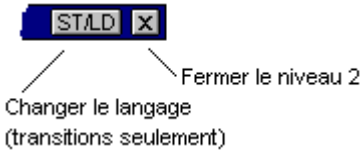
La commande "**Edition / Chercher remplacer**" permet de chercher et de remplacer des textes dans la programmation de niveau 2 de l'ensemble du programmes (toutes les étapes et toutes les transitions). La boîte de dialogue Chercher/Remplacer est utilisée pour saisir le texte cherché, et ouvrir le texte ou schéma de niveau 2 où le texte est trouvé.

A.4.4 Entrer la programmation du niveau 2

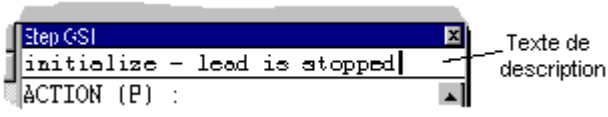
Pour entrer le programme de niveau 2, il suffit de cliquer deux fois sur le symbole d'une étape ou d'une transition. Le programme de niveau 2 apparaît sur la droite de la fenêtre SFC. La ligne de séparation entre les zônes d'édition SFC et niveau 2 peut être déplacée.
Il est possible d'éditer deux programmes de niveau 2 en même temps. Les commandes suivantes sont disponible avec la souris, le clavier, ou depuis le menu "Edition:

	Clavier	*Souris*	*Menu "Edition"*
Dans la même fenêtre	Entrée	Db.Clic	Editer le niveau 2
Dans une fenêtre séparée	Ctrl+Entrée	Ctrl + Db.Clic	Niveau 2 dans une autre fenêtre

Quand deux programmes de niveau 2 sont visibles, la séparation entre les deux zônes peut être déplacée. Le bouton sur la droite de la ligne de titre d'une zône de niveau 2 permet de fermer la fanêtre correspondante.
Le langage par défaut pour la programmation de niveau 2 est le **ST** (Texte Structuré). Pour les transitions, le langage de programmation pour le niveau 2 peut également être le **Quick LD**. Utilisez le bouton "**ST/LD**" dans la barre de titre du niveau 2 pour changer le langage de programmation. Cette commande n'est disponible que lorsque le programme de niveau 2 est vide.



Un champ de saisie d'une ligne de texte apparaît au sommet de la fenêtre de niveau 2. Il permet la saisie d'un texte de description court pour l'étape ou la transition. Ce texte est affiché sous la forme d'un commentaire IEC dans la fenêtre principale du SFC. Il est repris par les commandes telles que "Atteindre" ou pour la génération du dossier pour l'auto-documentation des étapes et transitions SFC.



La commande "**Options / Rafraichir**" pour mettre à jour dans la fenêtre SFC principale les programmes de niveau 2 en cours de modification dans les autres fenêtres.

Insérer le nom d'une variable

Lors de la programmation en langage littéral, pressez ce bouton pour sélectionner un symbole de variable à insérer à la position courante du curseur. Lors de la programmation en Quick LD, pressez sur ce bouton pour sélectionner la variable à attacher au symbole (contact, relais ou paramètre de bloc) sélectionné.

Insérer un bloc d'action impulsionnelle

Lors de la programmation du niveau 2 d'une étape, pressez ce bouton pour insérer un bloc d'action "Pulse" à la position courante du curseur. Voici le format d'un bloc d'action "Pulse":

```
Action (P):
   énoncés ST;
   ...
End_Action;
```

Les actions "Pulse" sont exécutées une seule fois quand l'étape devient active. Référez-vous au manuel de référence des langages ISaGRAF pour plus de détails sur la programmation SFC.

Insérer un bloc d'action non mémorisée

Lors de la programmation du niveau 2 d'une étape, pressez ce bouton pour insérer un bloc d'action "Non mémorisé" à la position courante du curseur. Voici le format d'un bloc d'action "Non mémorisé":

```
Action (N):
   énoncés ST;
   ...
End_Action;
```

Les actions "Non mémorisées" sont exécutées à chaque cycle pendant que l'étape devient active. Référez-vous au manuel de référence des langages ISaGRAF pour plus de détails sur la programmation SFC.

Nouveaux qualificatifs P0 et P1

ISaGRAF supporte les nouveaux qualificateurs d'action **P0** et **P1**. Lors de la programmation du niveau 2 d'une étape, pressez ce bouton pour insérer un bloc d'action P0 ou P1 à la position courante du curseur. Voici le format d'un tel bloc d'action:

```
Action (P0):                Action (P1):
   énoncés ST;                     énoncés ST;
   ...                             ...
End_Action;                 End_Action;
```

Les actions P1 sont exécutées une seule fois quand l'étape devient active (même comportement que les actions "Pulse"). Les actions P0 sont exécutées une seule fois quand l'étape devient inactive. Référez-vous au manuel de référence des langages ISaGRAF pour plus de détails sur la programmation SFC.

Actions booléennes

D'autres structures syntaxiques permettent de forcer directement une variable booléenne en fonction de l'activité de l'étape. Ces actions établissement une relation entre le **signal d'activité de l'étape** et une variable booléenne interne ou de sortie. Voici les syntaxes disponibles pour les actions booléennees:

<variable_boo> (N);..... copie le signal d'activité de l'étape dans la variable
<variable_boo>;..... même effet (l'attribut N est optionnel)
/ <variable_boo>;..... copie l'inverse logique du signal d'activité de l'étape dans la variable

D'autres types d'actions consistent à forcer la variable (à faux ou à vrai) quand l'étape devient active. Voici la syntaxe de ces actions:

<variable_boo> (S);..... force la variable à TRUE quand le signal d'activité de l'étape prend l'état TRUE
<variable_boo> (R);..... force la variable à FALSE quand le signal d'activité de l'étape prend l'état TRUE

Action SFC

Une action SFC est une séquence fille SFC, lancée ou tuée selon les évolutions du signal d'activité de l'étape. Une action SFC peut être décrite avec les qualificatifs d'action **N** (Non mémorisée), **S** (Set), ou **R** (Reset). Voici la syntaxe des actions SFC:

<programme_fils> (N);	lance la séquence fille quand l'étape devient active, et la tue lorsque l'étape redevient inactive
<programme_fils>;	même effet (l'attribut N est optionnel)
<programme_fils> (S);	lance la séquence fille quand l'étape devient active - rien n'est fait lorsque l'étape redevient inactive

<programme_fils> (R);	tue la séquence fille quand l'étape devient active - rien n'est fait lorsque l'étape redevient inactive

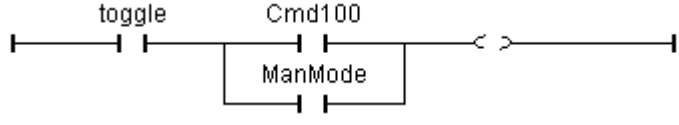
La séquence SFC spécifiée dans l'action doit être un **programme SFC fils** du programme en cours d'édition.

Transitions écrites en ST

Le niveau 2 d'une transition est une expression booléenne. En langage ST, il suffit d'exprimer la condition selon la syntaxe du ST. Optionnellement, la condition peut être terminée par un point-virgule.

Transitions écrites en LD

Il est possible d'utiliser l'éditeur Quick LD pour la programmation du niveau 2 d'une transition. Dans ce cas, le diagramme ne comporte qu'une échelle, dont l'unique bobine de relais représente la transition. Le nom de la transition n'est pas répété dans le diagramme. Voici un exemple de transition programmée avec l'éditeur Quick LD.



Lors de la programmation en Quick LD, utilisez les flèches du clavier pour déplacer le curseur dans la grille du diagramme, et utilisez les touches suivantres pour insérer les symboles LD:

F2:..... insère un contact après le symbole sélectionné / commence une échelle
F3:..... insère un contact avant le symbole sélectionné
F4:..... insère un contact en parallèle avec le symbole sélectionné
F6:..... insère un bloc après le symbole sélectionné
F7:..... insère un bloc avant le symbole sélectionné
F8:..... insère un bloc en parallèle avec le symbole sélectionné

Vous pouvez également cliquer sur les symboles correspondant dans la barre de touches fonctions affichée au bas de la fenêtre d'édition du niveau 2.

Frappez la touche ENTREE pour associer un symbole de variable au contact sélectionné, choisir un type de bloc (opération), ou connecter une variable en entrée ou en sortie d'un bloc. Vous pouvez également cliquer deux fois sur un symbole graphique pour sélectionner la variable associée;

Frappez CONTRÔLE + la BARRE D'ESPACE pour changer le type du contact sélectionné (direct ou inversé). Référez-vous au manuel d'utilisation de l'éditeur Quick LD pour plus de détails sur ses possibilités.

A.4.5 Utiliser la galerie SFC

L'éditeur SFC d'ISaGRAF gère une galerie d'objets SFC: c'est une collection de structures graphiques SFC pouvant être insérées dans tout nouveau graphe. Les éléments de la galerie SFC peuvent optionnellement embarquer les programmes de

niveau 2 associés aux étapes et transitions. Utilisez les commandes suivantes du menu "**Outils**":

Copier dans la galerie	copie les éléments sélectionnés dans un nouvel élément de la galerie
Copier depuis la galerie	copie le contenu d'un élément de la galerie à l'emplacement sélectionné

Lors de la copie vers la galerie SFC (création d'un nouvel élément de la galerie), vous pouvez demander d'embarquer les programmes de niveau 2 des étapes et transitions sélectionnées.

A.5 L'éditeur Flow Chart

L'éditeur graphique Flow Chart d'ISaGRAF permet la saisie des diagrammes FC et la programmation en ST, IL ou Quick LD des actions et des décisions. Le Flow Chart est un diagramme de décision qui peut également être utilisé pour la description des opérations séquentielles puisqu'il permet certaines opérations puissantes tels que les sauts en arrière non bloquants.

A.5.1 Bases du langage FC

Le **Flow Chart** (**FC**) est un langage graphique qui décrit des opérations séquentielles et un flux de décision. Un diagramme Flow Chart est composé d' **Actions** et de **Décisions**. Les actions et décisions sont reliées par des liaisons orientées représentant le flux. Voici les composants graphiques élémentaires du langage Flow Chart:

Début du diagramme: Un symbole "**begin**" doit apparaître au début du diagramme Flow Chart. Il est unique et ne peut pas être omis. Il représente l'état initial du diagramme lors de son activation.

Fin du diagramme: Un symbole "**end**" doit apparaître à la fin du diagramme Flow Chart. Il est unique et ne peut pas être omis. Toutefois il est possible qu'aucune liaison ne mène au symbole "End" (diagramme bouclant indéfiniment), mais le symbole "End" est tout de même présent dans le diagramme. Il représente l'état final du diagramme, quand son exécution est terminées.

Liaison (flux): Une **liaison** est une ligne orientée qui représente le flux de contrôle entre deux points du diagramme. Une liaison est toujours terminée par une flèche qui indique la direction du flux.

Action: Le symbole d'une **action** représente des actions à exécuter. Une action est identifiée par un numéro et un nom. Deux objets du diagramme ne peuvent pas avoir le même numéro. Le langage de programmation d'une action peut être le ST, le LD ou le IL. Une action doit toujours être connectée en amont et en aval aux autres objets du diagramme.

Décision: Une **décision** représente un **test** booléen. Une décision est identifiée par un numéro et un nom. Selon l'expression de la condition, exprimée en ST, LD ou IL, le flux est dirigé soit vers la sortie "YES" soit vers la sortie "NO". Quand la condition est programmée en ST, l'expression peut optionnellement être terminée par un point-virgule. Quand la condition est programmée en LD, la bobine unique représente la valeur de la condition.

Sous-programme FC: ISaGRAF permet la représentation d'une structure verticale de programmes FC. Les programmes FC sont organisés dans un **arbre de hiérarchie**. Chaque programme FC peut appeler un ou plusieurs **sous-**

programmes FC. Les programmes FC sont reliés dans l'arbre de hiérarchie par une relation de type "père / fils". Un symbole de **sous-programme** dans un diagramme Flow Chart représente un appel au sous programme. L'exécution du programme appelant est suspendue pendant toute l'exécution du sous-programme.

Action spécifique: Un symbole d'**action spécifique** représente une série d'actions à exécuter. Comme les autres symboles actions, il est identifié par un numéro et un nom. L'exécution d'une action spécifique n'a pas de différence avec celle d'une action normale. Le but des actions spécifiques est simplement de rendre le diagramme plus lisible en marquant les parties non portables du programme. L'utilisation des symboles d'action spécifique est optionnelle. Les actions spécifiques ont rigoureusement le même comportement que les autres actions.

Connecteur: Les **connecteurs** permettent d'exprimer une liaison entre deux points du diagramme, sans que le trait de la liaison complète soit tracé. Un connecteur est représenté par un cercle, connecté par un trait de liaison à la source du lien à exprimer. Le dessin du connecteur est complété, à sa droite, par le nom de l'élément qui est la destination du lien. Un connecteur se réfère toujours à un élément défini dans le même diagramme. La destination de la liaison est exprimée par le numéro de l'élément de destination.

Commentaire: Les blocs de **commentaires** n'ont pas d'influence sur le comportement du diagramme. Ils peuvent être insérés partout dans le diagramme, et permettent de documenter le programme.

A.5.2 Saisie du diagramme

Pour entrer un diagramme, il faut placer les éléments (actions, décision, connecteurs...) dans l'espace d'édition graphique, et tracer les liens (flux) entre les éléments.

Insérer des éléments

Pour insérer un objet dans le diagramme, sélectionner le type d'objet dans la barre d'outils puis cliquez dans le diagramme, là où vous voulez l'insérer. Vous pouvez soit poser l'élément sur une partie vide de la zône d'édition, soit l'insérer en cliquant sur une liaison. L'insertion sur une liaison n'est autorisée que dans le cas des liaisons verticales dirigées du haut vers le bas.Vous pouvez insérer:

 ..... une action programmée in ST, IL ou Quick LD

 ..... une action spécifique (marque un traitement particulier)

 ..... une décision programmée ST, IL ou Quick LD

 ..... un connecteur

 ..... un appel à un sous-programme FC

 ..... un commentaire

L'éditeur Flow chart d'ISaGRAF vous propose également une liste de structures complexes typiques, qui peuvent être directement insérées sur un lien. Ces structures ne peuvent pas être posées sur un emplacement vide.

 ..... If / Then / Else (décision)
 ..... Repeat until (attend une condition)
 ..... While (boucle tant qu'une condition vaut TRUE)

Sélectionner des éléments

La sélection des objets et des liens est nécessaire pour beaucoup de commandes d'édition. L'éditeur Flow Chart d'ISaGRAF permet les sélections simples et multiples. Pour sélectionner des objets, le bouton "**sélection**" (flèche) doit être sélectionné dans la barre d'outils. Pour sélectionner un seul objet, cliquez simplement sur son symbole.
Pour sélectionner un ensemble d'objets, glissez la souris pour tracer un rectangle de sélection. Tous les objets en intersection avec ce rectangle sont sélectionnés.
Un objet sélectionné est affiché en bleu foncé, avec des marques autour de son symbole. Il est également possible d'ajouter ou d'enlever un objet à la sélection courante en cliquant sur son symbole avec la touche **Maj** ou **Control** enfoncée.
La désignation d'une nouvelle sélection retire la sélection existante. Pour retirer la sélection courante, cliquez simplement dans une partie non occupée de la zone d'édition.
Lors une sélection simple, vous pouvez utiliser les flèches du clavier pour déplacer la sélection le long du diagramme. Les liaisons peuvent également être sélectionnées.

Insérer des commentaires

Des commentaires peuvent être insérés n'importe où dans le diagramme. Ils n'ont aucune influence sur le comportement du diagramme. Ils en augmentent la lisibilité. Pour insérer un bloc de commentaire, sélectionnez le bouton correspondant dans la barre d'outils et cliquez dans une partie inoccupée de la zone d'édition, là où le commentaire doit être inséré. Ensuite, cliquez deux fois sur le symbole du commentaire pour entrer son texte. Aucune syntaxe ni caractère particulier (tel que "**(***" et "***)**") n'est nécessaire. Un bloc de commentaire peut par la suite être déplacé ou redimensionné.

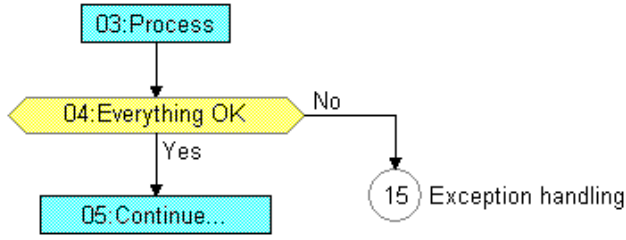
Tracer les flux

Sélectionnez ce bouton dans la barre d'outils pour tracer une liaison entre les éléments du diagramme. Un lien doit toujours être tracé dans la direction du flux. Cliquez d'abord sur un point de sortie non encore connecté, et glissez la souris vers le point de destination. La destination du lien peut être soit le sommet (entrée) d'un symbole non encore connecté, soit un emplacement sur un autre lien. Les points de convergence sont marqués par de petits cercles gris dans le diagramme. Les points de convergence peuvent également être sélectionnés et déplacés pour ré-arranger le diagramme.

Utilisation des connecteurs

L'éditeur Flow Chart d'ISaGRAF permet l'utilisation de connecteurs, en remplacement des arcs de liaisons explicites. Les connecteurs peuvent augmenter la lisibilité et la maintenance des diagrammes très grands. Un connecteur ne peut pas être utilisé pour établir un lien entre un programme et un autre programme.

Un connecteur est posé dans le diagramme comme les autres objets. Il est représenté par un cercle contenant le numéro de l'objet qui est la destination de la liaison. Le nom de l'objet de destination est affiché à droite du connecteur.



Déplacer des éléments

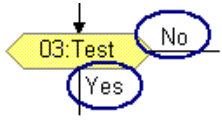
Pour déplacer des objets dans le diagramme, sélectionnez-les et faîtes glisser la souris vers le nouvel emplacement choisi. Vous pouvez déplacer un simple élément ou tous les éléments d'une sélection multiple. Les éléments ne doivent pas être superposés. Déplacer un élément ne peut pas servir à l'insérer sur un lien.
Quand un élément simple est déplacé (action, décision...), l'éditeur déplace automatiquement avec lui tous les éléments connectés en aval.

Redimensionner des éléments

Tous les objets graphiques à l'exception des symboles "Begin", "End" et des connecteurs peuvent être redimensionnés. Pour redimensionner un élément, il faut le sélectionner. Ensuite, glissez avec la souris les coins marqués sur le bord du symbole pour changer sa taille.
Quand un élément est connecté à une liaison, le redimensionnement en largeur agit à la fois à gauche et à droite, de façon à ce que l'objet reste centré sur le lien.

Echanger les sorties d'un test

Vous pouvez inverser les sorties YES / NO d'une décision. Pour ceci, cliquez deux fois sur une des marques "Yes" ou "No" affichées auprès de la décision.



A.5.3 Travailler sur un diagramme existant

Les commandes du menu "**Edition**" permettent de changer ou de compléter un diagramme existant. La plupart de ces commandes agissent sur les éléments sélectionnés dans le diagramme.

Correction du diagramme

Utilisez la touche SUPPR pour supprimer les éléments sélectionnés. Les liens attachés aux éléments sélectionnés sont aussi supprimés. Utilisez la commande

"**Edition / Annuler**" pour restituer les éléments supprimés par SUPPR. Les commandes "**Couper**", "**Copier**", "**Coller**" du menu "**Edition**" permettent de manipuler ou de dupliquer les éléments sélectionnés.

Chercher et remplacer

La commande "**Edition / Chercher remplacer**" permet de chercher et de remplacer des textes dans la programmation de niveau 2 de l'ensemble du programmes (toutes les actions et toutes les décisions). La boîte de dialogue Chercher/Remplacer est utilisée pour saisir le texte cherché, et ouvrir le texte ou schéma de niveau 2 où le texte est trouvé.

Atteindre un élément

La commande "**Edition / Atteindre**" permet d'accéder directement à n'importe quel objet du diagramme. La position de déroulement de la fenêtre d'édition est automatiquement mise à jour pour que l'objet soit visible.

Renuméroter les éléments

La commande "**Edition / Renuméroter**" renumérote tous les objets du diagramme Flow Chart. Tout élément FC doit être identifiié par un numéro unique. Les numéros sont alloués par l'éditeur lors de l'insertion de nouveaux éléments. La commande "**Renuméroter**" vous permet de repositionner tous les numéros selon l'ordre géographique des éléments dans le diagramme.

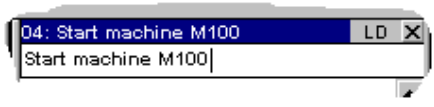
A.5.4 Entrer la programmation du niveau 2

Pour saisir la programmation de niveau 2, cliquez deux fois sur le symbole de l'action ou de la décision. La fenêtre d'édition du niveau 2 est ouverte à droite de la fenêtre Flow Chart. La ligne de séparation entre les niveaux 1 et 2 peut être glissée avec la souris pour redimensionner les fenêtres. Vous pouvez ouvrir jusqu'à deux fenêtres de niveau 2 en même temps. Les commandes suivantes sont disponibles au clavier, ou depuis le menu "Edition":

	Clavier	*Souris*	*Menu "Edition"*
Dans la même fenêtre	Entrée	Db.Clic	Editer le niveau 2
Dans une fenêtre séparée	Ctrl+Entrée	Ctrl + Db.Clic	Niveau 2 dans une autre fenêtre

Quand les deux fenêtres de niveau 2 sont visibles, vous pouvez glisser la séparation entre elles pour les redimensionner. Le bouton en haut et à droite du titre d'une fenêtre de niveau 2 permet de la fermer.
Le langage par défaut pour le niveau 2 est le **ST** (Texte structuré). Vous pouvez également programmer en **IL** ou **Quick LD**. Le nom du langage sélectionné est affiché dans une petite boîte dans la ligne de titre. Utilisez la commande "**Options / Langage niveau 2**" ou cliquez sur cette boîte pour changer le langage de programmation. Le langage ne peut pas être changé si la fenêtre de niveau 2 n'est pas vide.

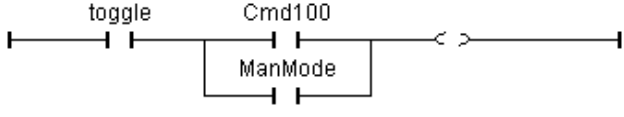
Un champ d'édition est affiché au sommet de la fenêtre de niveau 2. Il est utilisé pour rentrer le nom (texte de description court) de l'action ou de la décision. Ce texte sera affiché dans les symboles Flow Chart. Il est utile pour l'auto-documentation du diagramme, et sera repris pour certaines commandes telles que "Atteindre..." ainsi que lors de l'impression du diagramme pour la description des actions et des décisions.



La commande "**Options / Rafraîchir**" permet à tout moment de rafraîchir la vue principale Flow Chart avec les données en cours d'édition dans les fenêtres de niveau 2.

A.5.5 Programmation du niveau 2 avec le Quick LD

L'éditeur Quick LD est disponible pour la programmation du niveau 2 du Flow Chart. Dans le cas d'une décision, le diagramme LD est composé d'une seule échelle, dont la bobine représente le résultat de la décision. Le nom de la décision n'est pas répété sur la bobine. Voici un exemple de décision programmée en Quick LD.



Lors de la programmation en Quick LD, utilisez les flèches du clavier pour déplacer la sélection dans la grille d'édition, et utilisez les touches de fonction suivantes pour:

F2:..... insérer un contact après le symbole sélectionné / ou pour commencer une échelle
F3:..... insérer un contact avant le symbole sélectionné
F4:..... insérer un contact en parallèle avec le symbole sélectionné
F5:..... ajouter une bobine en parallèle avec celle sélectionnée (pas pour les décisions!)
F6:..... insérer un bloc après le symbole sélectionné
F7:..... insérer un bloc avant le symbole sélectionné
F8:..... insérer un bloc en parallèle avec le symbole sélectionné
F9:..... ajouter un symbole de saut en parallèle avec celle sélectionnée (pas pour les décisions!)

Un saut a pour destination une étiquette d'échelle. Le nom d'une échelle peut être changé en frappant la touche ENTREE quand la sélection est sur le début de l'échelle.
Vous pouvez également utiliser la barre d'outils LD en remplacement des touches de fonction.
Frappez la touche ENTREE quand la sélection est sur un contact ou un bloc pour sélectionner la variable associée, entrer une valeur constante ou choisir un type de bloc.
Frappez les touches Control + ESPACE quand la sélection est sur un contact ou une bobine pour changer son type (direct, inversé). Référez-vous au chapître "Edition Quick LD " pour plus de détails sur les possibilités de l'éditeur Quick LD.

A.5.6 Options d'affichage

La commande "**Options / Présentation**" ouvre une boîte de dialogue où sont groupés les paramètres et les options pour l'espace de travail et le dessin du diagramme FC. Utilisez les choix du groupe "Espace de travail" pour montrer ou cacher les barres d'outils et d'état de l'éditeur. Les options du groupe "Document" permettent de montrer ou de cacher les cellules de la grille, et de forcer l'affichage du diagramme en couleurs ou en noir et blanc.

Le bouton "Zoom" permet de sélectionner le ratio de zoom pour l'affichage du diagramme. Cette commande est également disponible lors de l'édition en Quick LD des actions et des décisions.

Utilisez le bouton "Grille" pour montrer ou cacher les points de la grille d'édition. Cette commande est également disponible lors de l'édition en Quick LD des actions et des décisions.

Utilisez la commande "**Options / Police**" pour sélectionner le nom de la police de caractères à utiliser dans tous les programmes graphiques. Lors de la sélection de la police, le style et la taille n'ont pas de signification et n'ont pas besoin d'être renseignés. Les éditeurs graphiques d'ISaGRAF calculent toujours la taille de la police en fonction du ratio de zoom courant.

A.6 L'éditeur Quick LD

La langage LD permet la représentation graphique d'équations booléennes. Les opérateurs booléens ET, OU, NON sont explicitement représentés par la topologie du diagramme. Des variables booléennes en entrée sont associées aux contacts. Des variables booléennes en sortie sont associées aux bobines de relais. L'éditeur Quick LD d'ISaGRAF permet la saisie rapide et simple d'un diagramme LD à l'aide du clavier ou de la souris. Les éléments sont automatiquement reliés et arrangés par l'éditeur Quick LD. Il n'est pas nécessaire de tracer les liaisons manuellement. L'éditeur Quick LD arrange les éléments du diagramme de façon à optimiser l'espace graphique occupé.

A.6.1 Bases du langage LD

Un programme LD est une suite d'équations appelées **échelles** où les contacts et les bobines de relais sont arrangés. Voici les composants de base d'un diagramme LD:

Début d'échelle (barre d'alimentation à gauche)

Chaque échelle commence par une barre d'alimentation à gauche, qui représente l'état VRAI. L'éditeur Quick LD d'ISaGRAF crée automatiquement la barre d'alimentation à gauche quand le premier contact de l'échelle est placé par l'utilisateur. Chaque échelle peut avoir un nom logique qui peut être utilisé comme une étiquette dans les instructions de saut.

Contacts

Un contact modifie le flux de données booléen, selon la valeur de la variable booléenne qui lui est associée. Le nom de la variable est affiché en dessus du dessin du contact. Les types de contacts suivants sont supportés par l'éditeur Quick LD:

 ..... contact direct

 ..... contact inversé

 ..... contact avec détection de front montant

 ..... contact avec détection de front descendant

Relais

Une bobine de relais représente une action. La variable associée au relais est forcée selon l'état de l'échelle à gauche du relais. Le nom de la variable est affiché en dessus du dessin du relais. Les types de relais suivants sont supportés par l'éditeur Quick LD:

 ..... sortie directe

 ..... sortie inversée

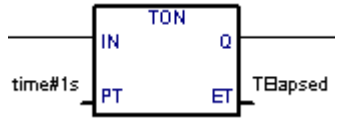
 ..... sortie à action "set"

 ..... sortie à action "reset"

-(P)- ..... sortie avec détection de front montant
-(N)- ..... sortie avec détection de front descendant

Blocs fonctionnels

Un bloc dans un diagramme LD peut représenter un opérateur, une fonction, un sous-programme ou un bloc fonctionnel. Sa première entrée et sa première sortie sont toujours connectés sur l'échelle. Les autres paramètres en entrée et en sortie sont écrits en dehors du rectangle du bloc.



Fin d'échelle (barre d'alimentation à droite)

Une échelle se termine par une barre d'alimentation à droite. L'éditeur Quick LD d'ISaGRAF ajoute automatiquement la barre d'alimentation à droite de chaque bobine de relais posée par l'utilisateur.

Symbole de saut

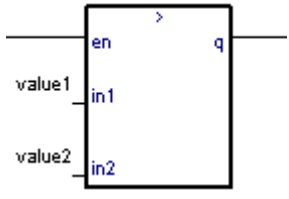
Un symbole de saut se réfère toujours à une étiquette (nom d'échelle) définie ailleurs dans le même diagramme LD. Le symbole de saut est posé à la fin d'une échelle. Quand l'état de la liaison à gauche du symbole est TRUE, l'exécution du diagramme est déroutée vers l'échelle identifiée par l'étiquette associée au saut. Notez que les sauts en arrière sont dangereux car ils peuvent bloquer le cycle automate.

Symbole "Return"

Un symbole "Return" est placé à la fin d'une échelle. Il indique que l'exécution du programme doit être abandonné si l'état de la liaison est TRUE. Un symbole "Return" est équivalent à un saut après la dernière échelle du diagramme.

L'entrée "EN"

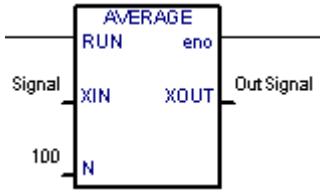
La première entrée de certains opérateurs, bloc fonctionnels ou fonctions, n'est pas du type booléen, et ne peut donc pas être connecté à une ligne LD booléenne. Donc, pour ces blocs, une entrée supplémentaire booléenne est ajoutée comme premier paramètre d'entrée. Cette entrée a le nom "**EN**". Dans ce cas, le bloc est exécuté uniquement si l'entrée **EN** reçoit un signal TRUE. Voici l'exemple du bloc comparateur et son équivalent en ST:




```
IF rung_state THEN
   q:= (value1 > value 2);
ELSE
   q:= FALSE;
END_IF;
(*la liaison à droite
 prend l'état de q *)
```

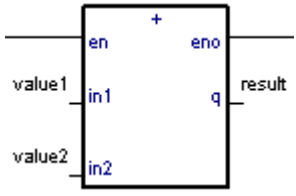
La sortie "ENO"

La première sortie de certains opérateurs, bloc fonctionnel ou fonctions, n'est pas du type booléen, et ne peut donc pas être connectée à une ligne LD booléenne. Donc, pour ces blocs, une sortie supplémentaire booléenne est ajoutée comme premier paramètre de sortie. Cette sortie a le nom "**ENO**". La sortie **ENO** prend toujours l'état de la première entrée du bloc. Voici l'exemple du bloc AVERAGE et son équivalent en ST.




```
AVERAGE(rung_state, Signal, 100);
OutSignal:= AVERAGE.XOUT;
eno:= rung_state;
(*la liaison à droite prend
 l'état de eno *)
```

Dans certains cas, les deux paramètres **EN** et **ENO** sont ajoutés. Voici l'exemple d'un opérateur arithmétique et son équivalent en ST.




```
IF rung_state THEN
   result:= (value1 + value2);
END_IF;
eno:= rung_state;
(*la liaison à droite prend
 l'état de eno *)
```

Limitations de l'éditeur Quick LD

L'éditeur Quick LD d'ISaGRAF ne permet pas de continuer une échelle (insertion d'autres contacts ou relais) à la droite d'une bobine de relais. Si plusieurs sorties doivent être affectées dans la même échelle, elles doivent être posés en parallèle.

A.6.2 Saisie du diagramme

Toutes les commandes proposées par l'éditeur Quick LD peuvent être réalisées soit avec le clavier soit avec la souris.

La grille d'édition

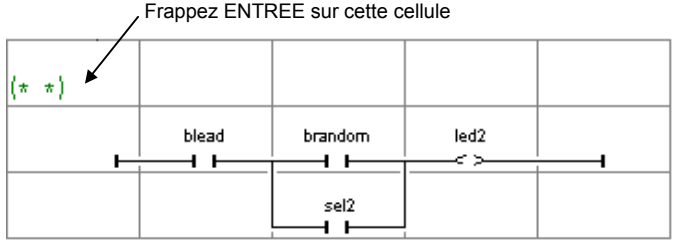
Le diagramme LD est entré dans une matrice logique. Chaque cellule de la matrice peut accueillir un symbole (contact ou relais). Utilisez les flèches du clavier ou cliquer avec la souris dans le diagramme pour changer la sélection courante. La cellule sélectionnée est dessinée en vidéo inverse. Pour certaines opérations (copier / coller), il est possible d'étendre la sélection sur plusieurs cellules. Pour ceci, faites glisser la souris dans le diagramme, ou utilisez les touches du clavier en gardant la touche SHIFT enfoncée.

Commencer une nouvelle échelle

Pour ajouter une nouvelle échelle, déplacer la sélection après la dernière échelle du diagramme et insérez un contact (frappez la touche F2 ou pressez le bouton correspondant dans la barre d'outils). Une nouvelle échelle comprenant un contact et un relais est ajoutée au diagramme.

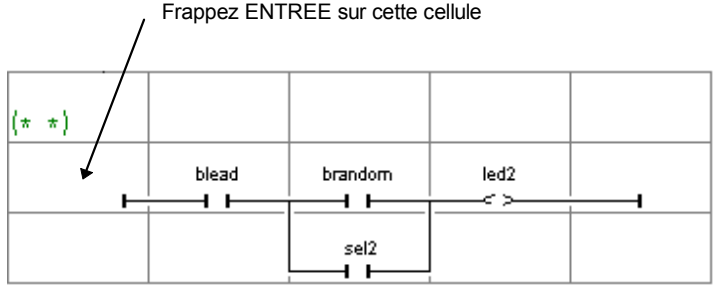
Entrer le commentaire de l'échelle

Chaque échelle peut être documentée avec une ou deux lignes de texte. Pour entrer le commentaire, déplacez la sélection sur la cellule en dessus du début de l'échelle et frappez la touche ENTREE, ou cliquez deux fois sur cette cellule avec la souris:



Entrer le nom de l'échelle

Chaque échelle peut être identifiée par une étiquette. Le nom de l'étiquette peut être utilisé comme destination pour les instructions de saut. Pour entrer ou changer le nom de l'étiquette, déplacez la sélection sur le début de l'échelle et frappez la touche ENTREE, ou cliquez deux fois sur cette cellule avec la souris:



L'éditeur Quick LD mémorise la liste des noms d'étiquettes saisis pour un identificateur d'échelle ou une destination de saut. La boîte de dialogue "Saut / Etiquette"'donne la possibilité d'entrer un nouveau nom ou de sélectionner un nom déjà saisi.
Si un nouveau nom est entré, il est automatiquement ajouté à la liste. Le bouton "**Supprimer**" permet de retirer le nom sélectionné de la liste. Il ne permet pas de retirer l'étiquette dans le diagramme. Pour ceci, pressez simplement le bouton **OK** quand le champ de saisie est vide.

Insérer des symboles sur une échelle

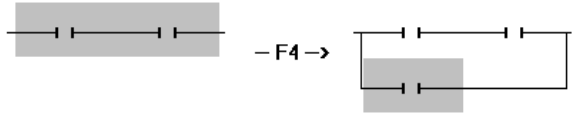
L'insertion de symboles (contacts, relais, blocs...) sur une échelle est toujours effectuée en fonction de la sélection courante. Vous devez sélectionner une cellule valide dans une échelle et frapper une des touches de fonction suivantes pour insérer:

F2..... un contact avant l'élément sélectionné (sur la gauche)
F3..... un contact après l'élément sélectionné (sur la droite)
F4..... un contact en parallèle avec l'élément sélectionné
F6..... un bloc avant l'élément sélectionné (sur la gauche)
F7..... un bloc après l'élément sélectionné (sur la droite)
F8..... un bloc en parallèle avec l'élément sélectionné

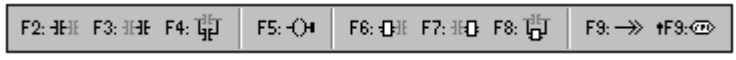
Les commandes suivantes ne sont valides que quand la sélection courante est sur une sortie de l'échelle (relais):

F5..... ajoute un relais en parallèle avec l'éleément sélectionné
F9..... ajoute un symbole de saut en parallèle avec l'élément sélectionné
Shift + F9 ..... ajoute un symbole "Return" en parallèle avec l'élément sélectionné

Pour les insertions en parallèle (F4/F8), si plusieurs contacts sont sélectionnés, le nouveau symbole est inséré en parallèle avec le groupe d'éléments. Voici un exmple:



Pour insérer des éléments dans le diagramme, vous pouvez aussi utiliser les commandes du menu "**Insertion**". Avec la souris, vous pouvez cliquer sur les boutons correspondants de la barre d'outils pour choisir le type d'élément à insérer:



Entrer les noms de variables

Pour associer un symbole de variable à un contact ou une bobine de relais, déplacez la sélection sur l'élément et frappez la touche ENTREE. Avec la souris, cliquez deux fois sur la cellule où est posé l'élément. Une boîte de sélection des variables est ouverte, avec toutes les variables déclarées dans le dictionnaire. Pour associer une fonction, un bloc fonctionnel ou un opérateur à un bloc, frappez ENTREE quand la sélection est à l'intérieur du rectangle du bloc. Pour associer un symbole de variable à une entrée ou une sortie (paramètre) de bloc, placez la sélection sur la ligne correspondante, **en dehors** du rectangle du bloc.
Normalement, des boîtes de dialogue incluant des listes de sélection des variables ou des blocs sont utilisées pour la saisie des textes dans le diagramme. Si le mode "**Saisie manuelle au clavier**" est sélectionné dans le menu "**Outils**", le nom des variables et des blocs est saisi directement dans un champ de saisie. Entrez le nouveau texte et frappez **ENTREE** pour le valider, ou frappez **ECHAP** pour abandonner. Le champ de saisie utilisé pour le mode de saisie manuelle au clavier ne peut pas être refermé avec la souris.

Changer le type des contacts et des relais

La commande "**Edition / Changer le type de symbole**" change le type du contact ou relais sélectionné. Un contact peut être direct, inversé, ou avec détection de front montant ou descendant. Un relais peut être direct, inversé, à action Set ou Reset, ou à détection de front montant ou descendant.

Insérer une échelle dans le diagramme

La commande "**Edition / Insérer une échelle**" insère une nouvelle échelle dans le diagramme, avant l'échelle sélectionnée. La nouvelle échelle est initialisée avec un contact et une bobine de relais.

A.6.3 Travailler sur un diagramme existant

Les commandes du menu "**Edition**" permettent de modifier un diagramme existant. La plupart de ces commandes agissent sur la sélection courante.

Correction du diagramme

Frappez la touche SUPPR pour supprimer les éléments sélectionnés. Il n'est pas possible de supprimer un relais, un saut ou un symbole "Return" quand il s'agit de l'unique sortie d'une échelle. Utilisez la commande "**Edition / Annuler**" pour restituer les éléments supprimés en cas d'erreur. La commande SUPPR peut être utilisée quand la sélection est sur un commentaire. La commande SUPPR, quand la sélection est sur le début d'une échelle, supprime l'échelle entière.

Copier les symboles

Les commandes "**Couper**", "**Copier**", "**Coller**" du menu "**Edition**" permettent de déplacer ou de copier les élements sélectionnés. Ces actions ne peuvent pas être utilisées sur les commentaires. La commande "**Edition / Collage special**" permet de choisir le mode d'insertion des éléments collés:

- avant l'élément sélectionné (sur la gauche)
- après l'élément sélectionné (sur la droite)
- en parallèle avec les éléments sélectionnés

Manipuler des échelles entières

Toutes les commandes d'édition (supprimer, copier, couper...) agissent sur l'échelle entière si la sélection est sur le début d'une échelle. Ceci procure un moyen rapide d'organiser les échelles dans le diagramme tout en gardant la sélection sur la marge gauche (première colonne). Il est également possible d'étendre la sélection sur plusieurs lignes de la première colonne pour manipuler plusieurs échelles en même temps.

Chercher et remplacer

Les commandes "**Edition / Chercher**" et "**Edition / Remplacer**" permettent de trouver et de remplacer des textes dans le diagramme. La recherche porte sur les contacts, les relais, les paramètres de blocs et les étiquettes. Elle ne peut pas trouver un texte à l'intérieur d'un commentaire. La commande Remplacer ne peut pas être utilisée pour changer le type d'un bloc. La recherche peut être effectuée en avant ou en arrière à partir de la sélection courante. Elle reprend au début quand la

fin du diagramme est atteinte. Les commandes suivantes sont également disponibles avec le clavier pour une recherche rapide de nom de variables:

ALT+F2 cherche la prochaine occurrence du nom de variable associé à la cellule sélectionnée. Cette fonction s'applique aussi aux blocs et aux étiquettes.

ALT+F5 cherche le prochain relais associé à la même variable que la cellule sélectionnée. Cette fonction est particulièrement utile pendant la mise au point pour trouver rapidement l'action qui positionne une variable en défaut.

A.6.4 Options d'affichage

Les commandes du menu "**Options**" permettent de personnaliser le dessin du diagramme LD à l'écran, et de cacher ou montrer certains types d'information.

Commentaires d'échelles

Utilisez la commande "**Options / Commentaires**" pour montrer ou cacher les commentaires dans le diagramme. Cacher les commentaires procure une vue plus compacte du diagramme, puisque chaque commentaire occupe une ligne de la matrice d'édition. Cette option de mofifie pas le contenu des commentaires, et peut être basculée à tout moment.

Noms et alias

Chaque variable associée à un contact, un relais ou un paramètre de bloc est identifié par son symbole. L'éditeur Quick LD d'ISaGRAF supporte la notion d'**alias** pour chaque variable. L'alias d'une variable est son commentaire, tronqué à la première occurence du caractère '**:**' et limité à 16 caractères. Voici quelques exemples:

```
commentaire de variable:     alias:
short text                   short text
long text with no separator  long text with n
short text: long description short text
```

Les alias n'ont aucune signification pour l'exécution d'un diagramme LD, et sont considérés comme des commentaires du point de vue syntaxique. L'alias d'une variable est automatiquement extrait de son commentaire quand la variable est sélectionnée dans la liste des objets déclarés. Il ne peut pas être changé manuellement. Utilisez les commandes "**Options / Contacts et bobines**" pour sélectionner le mode d'affichage des variables. Les modes suivants sont disponibles:

- n'afficher que les noms
- n'afficher que les alias
- afficher les noms et les alias ensemble

L'éditeur Quick LD ne met pas à jour automatiquement les alias dans le diagramme LD quand ils sont changés dans le dictionnaire. Utilisez la commande "**Options / Contacts et bobines / Mise à jour des alias**" pour mettre à jour tous les alias dans le diagramme. Vous pouvez aussi sélectionner l'option "**Mise à jour à l'ouverture**" du menu " **Options / Contacts et bobines**" pour demander à ISaGRAF de remettre

à jour automatiquement les alias à chaque fois qu'un diagramme est ouvert. Attention, l'utilisation de cette option peut considérablement augmenter le temps d'ouverture des programmes LD.

Options de dessin

La commande "**Options / Présentation**" ouvre une boîte de dialogue où sont groupés les paramètres et les options pour l'espace de travail et le dessin du diagramme LD.
Utilisez les choix du groupe "Espace de travail" pour montrer ou cacher les barres d'outils, d'état et LD de l'éditeur. Les options du groupe "Document" permettent de montrer ou de cacher les cellules de la grille, et de forcer l'affichage du diagramme en couleurs ou en noir et blanc.

Les options du groupe "Zoom" permettent de sélectionner le ratio de zoom pour l'affichage du diagramme. Vous pouvez également presser le bouton "zoom" de la barre d'outils pour basculer entre les ratios de zoom par défaut.

Vous pouvez personnaliser le ratio X/Y des cellules de la grille. Cette permet de réduire la largeur par défaut des cellules. Vous pouvez également presser le bouton "largeur de cellule" dans la barre d'outils pour changer le ratio X/Y sans ouvrir la boîte de dialogue de présentation.

Utilisez la commande "**Options / Police**" pour sélectionner le nom de la police de caractères à utiliser dans tous les programmes graphiques. Lors de la sélection de la police, le style et la taille n'ont pas de signification et n'ont pas besoin d'être renseignés. Les éditeurs graphiques d'ISaGRAF calculent toujours la taille de la police en fonction du ratio de zoom courant.

A.7 L'éditeur FBD/LD

L'éditeur graphique FBD/LD d'ISaGRAF permet la saisie de diagrammes FBD complets, pouvant comporter des portions en langage LD. Il combine des possibilités graphiques et littérales pour le saisie de diagrammes complets et documentés. Cet éditeur est plus particulièrement dédié au langage FBD. Pour des diagrammes comportant une grande majorité de réseaux LD, il est préférable d'uitiliser l'éditeur Quick LQ d'ISaGRAF.

A.7.1 Bases des langages FBD et LD

Le langage **FBD** est une représentation graphique d'équations mêlant plusieurs types de variables. Les **opérateurs** sont représentés par des boîtes de fonction rectangulaires. Les entrées d'une fonction sont connectées à gauche de la boîte. Les sorties de la fonction sont connectées à droite. Les entrées et les sorties du diagramme (**variables**) sont reliées aux boîtes fonctions par des **arcs de liaison logiques**. Une sortie d'une boîte fonction peut être connectée à une entrée d'une autre boîte.
Le langage **LD** permet la représentation graphique d'expressions booléennes. Les opérateurs booléens **ET**, **OU**, **NON** sont explicitement représentés par la topologie du diagramme. Les variables booléennes d'entrée sont attachées à des **contacts**. Les variables booléennes de sortie sont attachées à des **bobines de relais**. Les contacts et les bobines de relais sont reliés par des **arcs de liaison orientés**. Chaque segment de ligne prend l'état booléen **FAUX** ou **VRAI**. Toutes les liaisons connectées à droite d'une même barre verticale ont le même état booléen. Toutes les lignes horizontales connectées à la **barre d'alimentation à gauche** sont par définition à l'état **VRAI**.
Les diagrammes LD et FBD sont interprétés de la gauche vers la droite, et du haut vers le bas. Référez-vous au manuel de référence des langages d'ISaGRAF pour plus d'information sur les langages LD et FBD.

Barre d'alimentation à gauche

Toutes les lignes horizontales connectées à la **barre d'alimentation à gauche** sont par définition à l'état **VRAI**. L'éditeur FBD d'ISaGRAF permet le connection de tout symbole booléen à la barre d'alimentation à gauche.

Barre d'alimentation à droite

Les bobines de relais peuvent être connectées à droite à la **barre d'alimentation à droite**. Ceci est optionnel avec l'éditeur FBD d'ISaGRAF. Si une bobine n'est pas connecté à droite, son dessin inclut sa propre barre d'alimentation.

Connexion verticale "OU" du LD

Les connections verticales du langage LD acceptent plusieurs connections à gauche et plusieurs connections à droite. Chaque connection à droite à pour l'état le OU logique entre l'état de toutes les connections à gauche.

Contacts

Un contact modifie le flux de données booléen, selon l'état de la variable booléenne qui lui est associée. Le nom de la variable est affiché en dessus du dessin du contact. Les types de contacts suivants sont supportés par l'éditeur FBD/LD d'ISaGRAF:

-| |- ..... contact direct

-|\|- ..... contact inversé

-|P|- ..... contact avec détection de front montant (positif)

-|N|- ..... contact avec détection de front descendant (négatif)

Relais

Une bobine de relais représente une affectation de la variable booléenne qui lui est associée en fonction de l'état de sa connection à gauche. Le nom de la variable est affiché en dessus du dessin du relais. Les types de relais suivants sont supportés par l'éditeur FBD/LD d'ISaGRAF:

-( )- ..... action directe

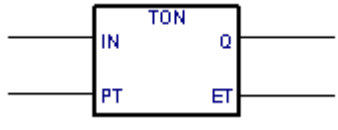
-(\)- ..... action inversée

-(S)- ..... relais à action "set"

-(R)- ..... relais à action "reset"

Blocs fonctionnels

Un bloc dans un diagramme FBD peut représenter un opérateur, une fonction, un bloc fonctionnel ou un sous-programme. Les entrées et les sorties du bloc doivent être connectées à des variables, des contacts ou des bobines de relais, ou à des entrées/sorties d'autres blocs. Les noms de paramètres formels du bloc sont affichés à l'intérieur du rectangle du bloc.



Etiquettes

Des étiquettes peuvent être placées partout dans le diagramme. Les étiquettes sont utilisées comme cible pour les symboles de saut, pour changer l'ordre d'évaluation du réseau. Les étiquettes ne sont pas reliées aux autre éléments du diagramme. Il est vivement conseillé de placer une étiquette sur la marge gauche du diagramme pour augmenter la lisibilité et la compréhension du programme.

Sauts

Un symbole de saut est toujours relatif à une étiquette, placée ailleurs dans le diagramme. Si l'état de la connection à gauche du symbole de saut est VRAI, l'exécution du réseau est déroutée après le symbole de l'étiquette. Notez que les sauts en arrière sont dangereux car ils peuvent bloquer l'exécution du cycle automate.

Symbole "Return"

Un symbole "return" doit être connecté à gauche à un symbole booléen. Il indique que l'évaluation du réseau doit être arrêtée si l'état de la liaison à gauche est VRAI. Il est équivalent à un symbole de saut ayant pour destination la fin du programme.

Variables

Les variables sont représentées dans le diagramme par des rectangles connectés à gauche ou à droite aux autres éléments du diagramme.

Connexions

Les arcs de liaison sont tracés pour relier les éléments du diagramme. Les arcs sont toujours tracé depuis un point de sortie vers un point d'entrée (dans le sens du flux de données).

Connexion avec inversion booléenne

Certains arcs de liaison entre objets booléens sont représentés avec un petit cercle à leur extrémité droite. Ceci représente une inversion booléenne de l'information transportée par la liaison.

Coins définis par l'utilisateur

Des "coins" peuvent être librement placés sur les arcs de liaison. Ils permettent à l'utilisateur de controller manuellement le tracé de l'arc en fonction de l'emplacement du coin. Si aucun coin n'est placé sur un arc, ISaGRAF utilise un algorithme de routage automatique pour tracer l'arc.

A.7.2 Saisie du diagramme

Pour saisir un diagramme, il faut placer les éléments graphiques (blocs, variables, contacts, relais...) dans le schéma, et les connecter ensuite avec des arcs de liaisons.

Insérer des éléments

Pour insérer un objet dans le diagramme, sélectionnez son type dans la barre d'outils de l'éditeur, et cliquez dans le diagramme là où vous voulez l'insérer.

Sélectionner des éléments

La sélection de symboles graphiques est requise pour la plupart des fonctions d'édition. L'éditeur graphique LD / FBD d'ISaGRAF permet la sélection d'un ou plusieurs objets dans la zone d'édition graphique. Pour sélectionner des objets, le choix "**sélection**" (bouton représentant une flêche) doit être validé dans la barre d'outils de l'éditeur. Pour sélectionner un seul objet, il suffit de cliquer sur son symbole. Pour sélectionner un groupe d'objets, faites glisser la souris dans le diagramme en traçant un rectangle. Tous les objets en intersection avec le rectangle sont sélectionnés. Un objet sélectionné est représenté avec de petits carrés noirs autour de son symbole. La désignation d'une nouvelle sélection efface les sélections antérieures. Pour annuler la sélection courante, il suffit de cliquer dans un espace vide du diagramme, en dehors du **rectangle de désignation**, qui englobe la liste des objets sélectionnés.

Insérer des commentaires

Des commentaires peuvent être insérés n'importe où dans le diagramme. Les commentaires n'ont aucune influence sur l'exécution du programme. Ils augmentent considérablement la lisibilité du diagramme. Pour insérer un commentaire, sélectionnez ce bouton dans la barre d'outils, et faites glisser la souris dans le diagramme pour désigner l'emplacement et la taille du bloc de commentaire. Puis entrez son texte. Il n'est pas nécessaire d'encadrer le texte d'un commentaire par les séparateurs "**(***" et "***)**" des langages littéraux. Un bloc de commentaire peut être redimensionné quand il est sélectionné, en faisnt glisser un des coins de son cadre.

Déplacer des éléments

Pour déplacer des objets dans le diagramme, il faut d'abord les sélectionner. Puis cliquez dans le rectangle sélectionné et faites glisser la souris vers un nouvel emplacement. Pour déplacer des objets reliés entre eux, il suffit de déplacer leur symbole. L'éditeur ISaGRAF mettra automatiquement à jour le tracé des arcs de liaison en fonction de leur nouvel emplacement.

Tracer les connexions

Sélectionnez un de ces boutons dans la barre d'outils pour tracer un arc de de liaison entre deux points de connection de symboles déjà posés dans le diagramme. Si vous désignez un espace non occupé pour l'extrémité de l'arc, il est automatiquement terminé par un "coin", et vous pouvez ainsi enchainer le tracé de plusieurs segments d'une connexion logique.

Modifier le tracé des connexions

La commande "**Outils / Déplacer ligne**" est utilisée quand un arc de liaison est sélectionné, pour changer son algorithme de routage automatique. Cette commande est sans effet si l'arc est connecté à un coin manuellement défini par l'utilisateur. Quand la liaison est composée de trois segments, cette commande change la position du segment intermédiaire. Voici quelques exemples:

Changer le type d'une connexion

Vous pouvez facilement changer le type d'un arc de liaison (avec ou sans inversion booléenne) en cliquant deux fois sur sont extrémité à droite.

Saisie des échelles LD

Pour tracer une nouvelle échelle LD, placez tout d'abord sa barre d'alimentation à gauche. Puis placez la bobine de relais. La barre horizontale de l'échelle est tracée. D'autres contacts ainsi que des liaisons verticales "OU" peuvent ensuite être directement insérés sur l'échelle, sans besoin de tracer manuellement les liaisons entre contacts.

Quand un nouveau contact ou relais est posé sur un espace vide de la zône d'édition, une nouvelle ligne LD horizontale est automatiquement tracée autour de son symbole, jusqu'aux barres d'alimentation à gauche et à droite, si elles existent. Ces lignes ne sont pas tracées si le symbole n'est pas posé entre des barres d'alimentation. Le symbole et les lignes ainsi créés peuvent par la suite être librement déplacés dans le diagramme. Les lignes horzontales créées lors de l'insertion de contacts LD peuvent être sélectionnées, déplacées ou supprimées. Il est possible de poser un contact ou un relais LD sur une ligne horizontale existante. Dans ce cas, la ligne est automatiquement coupée pour être re-connectée à gauche et à droite du symbole posé.

Connexions multiples

Une liaison multiple peut être créée à droite de chaque point de connexion en **sortie**. Une liaison multiple indique que l'information doit être **propagée** vers plusieurs autres points du diagramme. Le même état est propagé vers toutes les extrémités à droite. Le nombre d'arcs tirés à droite d'un même point de connexion en sortie n'est pas limité. Deux arcs de liaison ne peuvent pas avoir leur extrémité droite connectée sur le même point d'entrée, sauf s'ils sont connectés à l'un des symboles LD suivants:

 ..... barre d'alimentation à droite

 ..... connection verticale avec liaison multiple à gauche (opérateur OU du LD)

Ces symboles du langage LD peuvent avoir un nombre illimité de connections à gauche.

A.7.3 Travailler sur un diagramme existant

Les commandes du menu "**Edition**" permettent de modifier ou de compléter un diagramme existant. La plupart de ces commandes agissent sur les éléments sélectionnés dans le diagramme.

Correction du diagramme

Utilisez la touche SUPPR pour supprimer les objets sélectionnés. Utilisez la commande "**Edition / Annuler**" pour restituer les éléments effacés en cas d'erreur. Les commandes "**Couper**", "**Copier**", "**Coller**" du menu "**Edition**" permettent de déplacer ou de copier les éléments sélectionnés.

Chercher et remplacer

Les commandes "**Edition / Chercher**" et "**Edition / Remplacer**" permettent de chercher et de remplacer des textes dans le diagrammes. Seuls les noms complets sont trouvés. La rechercher porte sur les contacts, les relais, les noms de bloc, les variables et les étiquettes. Elle ne peut pas trouver un texte dans un bloc de commentaire. La commande Remplacer ne peut pas être utilisée pour changer le type d'un bloc. La rechercher est effectuée du haut vers le bas, à partir de la sélection en cours. Elle reprend en début de diagramme quand la fin est atteinte.

Afficher l'ordre d'exécution

Lorsqu'un diagramme FBD inclut des connections en boucle, l'ordre d'exécution ne peut plus suivre la simple règle haut vers bas / gauche vers droite. Afin d'éviter toute confusion dans l'interprétation du diagramme, utilisez la commande "**Outils / Montrer l'ordre d'exécution**" ou frappez les touches **Control + F1** pour afficher l'ordre dans lequel les sorties du diagramme seront mises à jour à l'exécution. Des repères numérotés à partir de 1 sont ajoutés au diagramme. Ils sont automatiquement effacés si le diagramme est modifié.

Saisie des symboles et des textes

Plusieurs symboles LD ou FBD sont complétés par un nom de variable ou un autre texte. Ce texte est saisi une première fois quand l'objet est inséré dans le diagramme. Pour changer le texte associé à un objet, il suffit de cliquer deux fois sur son symbole et d'entrer le nouveau texte.
Normalement, des boîtes de dialogue incluant des listes de sélection des variables ou des blocs sont utilisées pour la saisie des textes dans le diagramme. Si le mode "**Saisie manuelle au clavier**" est sélectionné dans le menu "**Outils**", le nom des variables et des blocs est saisi directement dans un champ de saisie. Entrez le nouveau texte et frappez **ENTREE** pour le valider, ou frappez **ECHAP** pour abandonner. Le champ de saisie utilisé pour le mode de saisie manuelle au clavier ne peut pas être refermé avec la souris.
Pour les contacts et les relais, le symbole de la variable doit être saisi immédiatement lorsque le symbole est inséré, si l'option "**Saisie immédiate**" est validée. Le symbole doit toujours être saisi immédiatement pour une variable ou une étiquette.

Sélectionner un type de bloc

Cliquez deux fois sur le rectangle d'un bloc pour changer son type. Le type du bloc est sélectionné dans la liste des opérateurs, fonctions et blocs fonctionnels standard et éléments de la librairie. Cette commande permet également de changer le nombre d'entrée dans le cas d'un opérateur commutatif (AND, OR, ADD, MUL...)

Insérer des lignes vides

Quand vous pressez le bouton droit de la souris dans le diagramme FBD, un menu est affiché. Il contient les commandes suivantes qui permettent d'insérer ou de supprimer de l'espace non utilisé à la position indiquée par la souris:

Insérer des lignes:	Cette commande insère un espace libre, composé de 4 lignes complètes, à la position indiquée par la souris quand le menu a été ouvert.
Supprimer des lignes:	Cette commande supprime l'espace (lignes) inutilisé à partir de la position indiquée par la souris quand le menu a été ouvert. Cette commande ne permet pas de supprimer des éléments du diagramme FBD.

Quand le menu est ouvert, une ligne grise dans le diagramme FBD indique la position à laquelle les lignes vides seront insérées ou supprimées.

A.7.4 Options d'affichage

Les commandes du menu "**Options**" permettent de personnaliser le dessin du diagramme à l'écran.

- **Présentation**
  La commande "**Options / Présentation**" ouvre une boîte de dialogue ou les paramètres d'affichage sont saisis. Utilisez les options du groupe "Espace de travail" pour cacher ou montrer les barres d'outils et de status de l'éditeur. Les options du groupe "Document" permettent d'afficher ou de montrer la grille d'édition, et de choisir entre un affichage en couleurs ou en noir et blanc.

Les options du groupe "Zoom" pemettent de saisir le ratio de zoom pour l'affichage. Vous pouvez aussi utiliser le bouton "zoom" dans la barre d'outils pour changer entre les ratios de zoom les plus courants.

Utilisez la commande "**Options / Police**" pour sélectionner le nom de la police de caractères à utiliser dans tous les programmes graphiques. Lors de la sélection de la police, le style et la taille n'ont pas de signification et n'ont pas besoin d'être renseignés. Les éditeurs graphiques d'ISaGRAF calculent toujours la taille de la police en fonction du ratio de zoom courant.

A.7.5 Styles et marquage des modifications

L'éditeur LD/FBD d'ISaGRAF permet d'affecter un style graphique à chaque élément du diagramme. Un style est matérialisé part une couleur particulière. ISaGRAF utilise également les styles pour le marquage des modifications à la saisie, dans une optique de gestion de versions d'un diagramme.
Les styles ne sont pas visibles pendant la simulation ou le test en ligne, puisque les couleurs (rouge, bleu) sont alors utilisées pour le marquage des états TRUE et FALSE des variables espionnées.

- ***Styles prédéfinis***

Les styles suivants sont prédéfinis par ISaGRAF:

Normal..... Couleur par défaut (noir). Pour le marquage des modifications, le style "normal" indique que l'élément fait partie de la version originale du diagramme. Les éléments ayant le style "normal" sont normallement exécutés.

Modifié ..... Les éléments ayant le style "modifié" sont dessinés en rose. Pour le marquage des modifications, le style "modifié" indique que l'élément a été ajouté ou modifié depuis la version originale du diagramme. Les éléments ayant le style "modifié" sont normallement exécutés.

Supprimé ..... Les éléments ayant le style "supprimé" sont dessinés en gris, avec des lignes en pointillés. Ces éléments ne sont pas pris en compte lors de l'erxécution du diagramme. Ce style permet de garder une trace visuelle des éléments supprimés depuis la version originale d'un diagramme.

Personnalisé..... En plus des styles prédéfinis, l'éditeur LD/FBD d'ISaGRAF permet de choisir une couleur quelconque pour l'affichage d'un

élément du diagramme. Ces éléments ont alors un style "personnalisé". Les éléments ayant le style "personnalisé" sont normallement exécutés.

Utilisez les commandes "**Style**" du menu "**Edition**" pour manuellement appliquer un style aux éléments sélectionnés dans le diagramme.

Marquage des modifications

L'utilisation des styles ainsi que le style "Supprimé" permet le marquage automatique à la saisie des modifications apportées au diagramme. Utilisez la commande "**Marquer les modifications**" du menu "**Edition / Style**" pour valider le marquage automatique des modifications.

Quand l'option "Marquer les modifications" est validée, tous les éléments modifiés ou ajoutés au diagramme sont automatiquement forcés en style "Modifié". Quand un élément est supprimé à l'aide des commandes "Effacer" ou "Couper", il continue d'être visible dans le diagramme, mais est forcé en style "Supprimé". Ceci permet de garder une trace de toutes les modifications apportées au diagramme édité.

Utilisez la commande "**Edition / Style / Retirer tous les éléments supprimés**" pour effectivement retirer du diagramme tous les éléments ayant le style "Supprimé". Cette commande est indépendante de la sélection courante, et s'applique toujours à l'intégralité du diagramme.

Pour "restituer" un élément ayant le style "Supprimé", sélectionnez le et appliquez lui le style "Normal" ou "Modifié" ou "Personnalisé". Cette opération peut engendrer des connections invalides entre les éléments du diagramme (plus d'un lien connectés vers le même point) qui seront détectées pendant la prochaine vérification du programme.

A.8 L'éditeur de texte

Ce chapître décrit l'utilisation de l'éditeur de textes d'ISaGRAF, particulièrement quand il est utilisé pour éditer le code source des programmes en ST ou IL.

A.8.1 Commandes d'édition

Les commandes du menu "**Edition**" permettent de manipuler le texte saisi. La plupart de ces commandes agissent sur la portion de texte sélectionné dans le document, ou réalisent une opération à la position courante du curseur.

Couper et coller

La touche DEL supprime le texte sélectionné. Utilisez la commande "**Edition / Annuler**" pour restituer le texte détruit par DEL en cas d'erreur. Les commandes "**Couper**", "**Copier**", "**Coller**" du menu "**Edition**" permettent de déplacer ou de copier un texte dans le document, ou d'insérer des textes copiés depuis d'autres applications.

Chercher et remplacer

Les commandes "**Edition / Chercher**" et "**Edition / Remplacer**" permettent de chercher et de remplacer un texte dans le document. Toute chaîne de caractères peut être trouvée. La recherche peut être effectuée en avant ou en arrière à partir de la position courante du curseur. La recherche ne "boucle" pas quand les limites du document sont atteintes.

Atteindre une ligne

La commande "**Edition / Atteindre**" permet de déplacer directement le curseur sur une ligne du document en indiquant le numéro de la ligne. Ceci est utile lorsqu'une erreur détectée par le générateur de code est référencée par son numéro de ligne dans un programme ST ou IL.

Insérer un nom de variable

Utilisez la commande "**Edition / Insérer variable**" pour insérer à la position courante du curseur le symbole d'une variable déclarée dans le dictionnaire. Le symbole est sélectionné dans la liste des variables déclarées.

Insérer le contenu d'un fichier

La commande "**Edition / Insérer fichier**" insère à la position courante du curseur le contenu intégral d'un fichier de texte. Notez que seuls les fichiers de pur texte ASCII peuvent être manipulés par cette commande.

A.8.2 Options

Les commandes du menu "**Options**" permettent de cacher ou de montrer la barre d'outils de l'éditeur, et de sélectionner la police de caractères. La police sélectionné sera utilisée pour toutes les éditions de texte dans l'ensemble de l'atelier ISaGRAF.
Quand l'éditeur est utilisé pour entrer le code source d'un programme ST ou IL, la commande "**Options / Afficher les mots clés**" permet d'afficher ou de cacher une boîte à boutons dans laquelle sont regroupés les principaux mots réservés du langage. Cliquez sur un des boutons pour insérer le texte du mot réservé à la position courante du curseur.

A.9 Autres commandes des éditeurs de programmes

Ce chapître regroupe des informations utiles sur les fonctionalités communes à tous les éditeurs de programmes d' ISaGRAF. Il s'agit en principal des liens avec les autres outils d'ISaGRAF ainsi que les boîtes de dialogue communes.

A.9.1 Lancement des autres outils d'ISaGRAF

Vérifier (compiler) le programme

La commande "**Fichier / Vérifier**" lance le générateur de code d'ISaGRAF pour vérifier la **syntaxe du programme** en cours d'édition. Dans le cas d'un programme SFC, les informations de niveau 1 et de niveau 2 sont vérifiées. Quand la vérification est terminée, fermez la fenêtre du générateur de code pour reprendre votre travail d'édition du programme ouvert. Si le projet contient un seul programme, le code de l'application est généré s'il n'y a pas d'erreur de compilation.
La commande "**Options / Options de compilation**" permet de sélectionner les options pour la génération et l'optimisation du code de l'application. Référez-vous au manuel d'utilisation du générateur de code d'ISaGRAF pour plus de détails sur les options de compilation et leur signification.

Simuler ou tester l'application

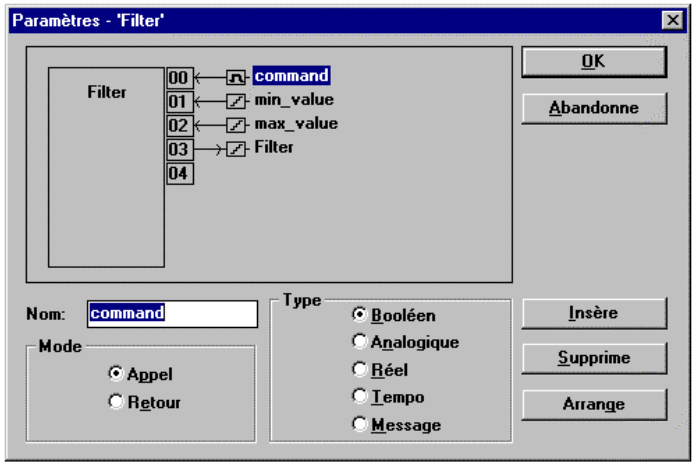
Les commandes "**Fichier / Simuler**" et "**Fichier / Tester**" lancent le debugger graphique d'ISaGRAF en mode simulation ou en mode réel connecté à l'automate, et ré-ouvre le programme édité en mode mise au point. Dans ce mode, il n'est plus possible de modifier le contenu du programme.

Editer le dictionnaire de variables

La commande "**Fichier / Dictionnaire**" lance l'édition du **dictionnaire** de variables de l'application. Le dictionnaire permet également de définir les mots équivalents (alias). Les déclarations de portée **locale** sont toujours relatives au programme édité.

A.9.2 Paramètres du programme

Quand le programme édité est une fonction, un sous-programme ou un bloc fonctionnel, la commande "**Fichier / Paramètres**" permet de définir ses paramètres d'appel et de retour. Cette commande est sans effet si le programme édité est un programme principal des sections "Début" ou "Fin" ou un programme SFC.
Un sous-programme, une fonction ou un bloc fonctionnel peut avoir jusqu'à 32 paramètres (entrées ou sorties). Une fonction ou un sous-programme a toujours un paramètre de retour et un seul, qui doit porter le nom de la fonction, conformément aux règles d'écriture du langage ST.



La liste dans la partie supérieure de la fenêtre montre les paramètres du sous-programme, selon l'ordre défini par le prototype d'appel du sous-programme: d'abord les paramètres d'appel, et enfin le paramètre de retour. La partie inférieure de la fenêtre montre la description détaillée du paramètre sélectionné dans la liste. Chacun des types de données d'ISaGRAF peut être utilisé pour un paramètre. Les paramètres de retour doivent être les derniers de la liste. Les règles suivantes doivent être respectées pour la nomenclature des paramètres:

- la longueur du nom ne doit pas excéder 16 caractères
- le premier caractère doit être une lettre
- les suivants doivent être des lettres, des chiffres, ou le souligné
- les majuscules et les minuscules ne sont pas différenciées

La commande "**Insère**" insère un emplacement libre pour un nouveau paramètre, avant le paramètre sélectionné. La commande "**Supprime**" efface le paramètre sélectionné. La commande "**Arrange**" trie les paramètres, de façon à ce que les paramètres de sortie soient placés en fin de liste.

A.9.3 Autres commandes du menu "Fichier"

Les commandes suivantes sont disponibles dans le menu "**Fichier**" des éditeurs de programmes:

Ouvrir un autre programme

La commande "**Fichier / Ouvrir**" ferme le fichier en cours d'édition et lance l'édition d'un autre programme avec le même langage . Cette fonction ne permet pas d'ouvrir le programme écrit dans un autre langage. Le programme sélectionné remplace le programme en cours d'édition dans la fenêtre de l'éditeur.

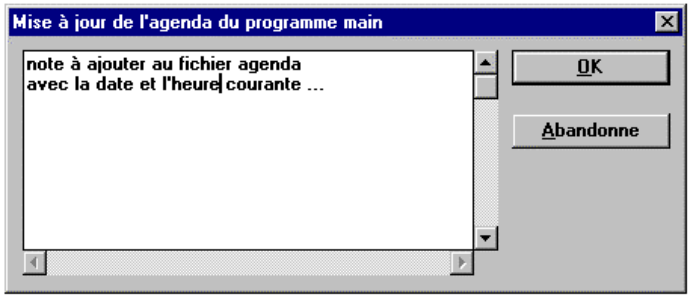
Imprimer le programme édité

La commande "**Fichier / Imprimer**" permet d'imprimer le programme en cours d'édition. Cette commande lance le générateur de dossier d'ISaGRAF pour l'impression du programme et de ses variables locales.
Pour certains langages graphiques (SFC, FBD et Quick LD) vous pouvez également utiliser la commande "**Edition / Copier image**" qui copie dans le presse-papier le dessin du diagramme édité en format "metafile". Le diagramme peut ainsi être collé dans d'autres applications telles qu'un traitement de texte. Pour les graphes SFC, seule la programmation du niveau 1 (graphe, numérotation et commentaires) apparaît dans le metafile.

A.9.4 Mise à jour du fichier agenda

Un fichier est attaché à chaque programme, et peut accueillir les notes et remarques relatives à sa programmation. Utilisez la commande "**Fichier / Agenda**" pour accéder à l'édition de ce fichier. Le fichier agenda est automatiquement mis à jour à chaque vérification syntaxique. Les derniers messages de compilation peuvent y être consultés, ainsi que la date de la compilation.

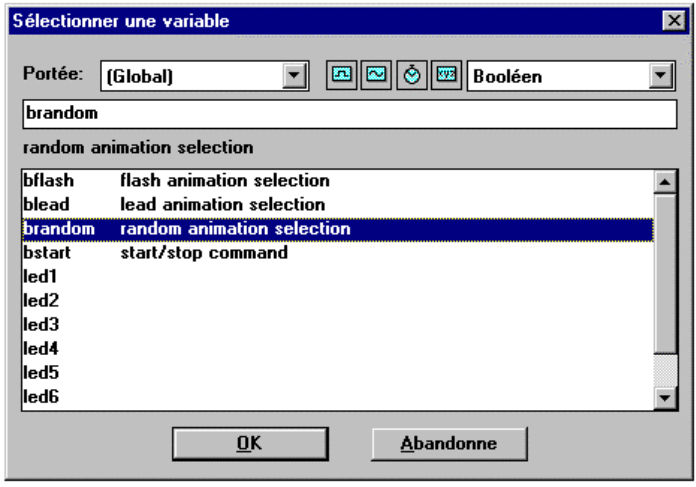
Quand l'option "**Mise à jour de l'agenda**" est choisie, la fenêtre de saisie des annotations pour l'agenda est automatiquement ouverte à chaque sauvegarde des modifications apportées au programme.



Ceci permet de créer facilement une trace des évolutions de chaque programme pendant le développement et la maintenance d'une application. Notez que cette option est commune pour tous les programmes (de tout langage) du projet. Pressez le bouton OK pour enregistrer dans le fichier agenda les notes saisies, ainsi que la date courante.

A.9.5 Sélectionner une variable déclarée dans le dictionnaire

Pendant l'édition des programmes littéraux (ST ou IL) la commande "**Edition / Insérer variable**" permet de sélectionner une variable dont le symbole doit être inséré à la position courante du curseur. Pendant l'édition des programmes graphiques LD ou FBD, la sélection de variable est requise pour l'association aux contacts, relais, paramètres de blocs et champs "variables" des diagrammes FBD. Dans tous les cas, la boîte de dialogue de sélection de variable est utilisée.



Le champ "**Portée**" permet de sélectionner la portée des variables à lister (globales ou locales). La boîte de sélection de droite permet de choisir un type de variable. Les boutons graphiques à côté de cette boîte permettent de choisir les types les plus utilisés:

 ..... Booléen

 ..... Entier / Réel

 ..... Temporisation

 ..... Message

Pour sélectionner une variable, cliquez sur son nom dans la liste. Son nom est alors affiché en tête de liste. Pressez alors le bouton "**OK**" pour confirmer sa sélection. Vous pouvez également cliquer deux fois dans la liste pour sélectionner une variable, ou entrer manuellement son symbole sans utiliser la liste.

A.9.6 La fenêtre de messages

Les commandes suivantes sont disponibles dans le menu "**Outils**" de chaque éditeur de programme. Elle permettent d'afficher des informations dans une liste située en bas de la fenêtre d'édition, et d'utiliser la liste pour naviguer dans le programme édité.

"Afficher les messages de compilation": ouvre la fenêtre de messages et y affiche la liste des messages produits lors de la dernière vérification (compilation) du programme édité.
"Chercher dans...": chercher les occurrences d'un texte dans l'ensemble du programme édité, et affiche les occurences dans la fenêtre de messages. Pour les langages SFC et FC, cette commande cherche dans tous les blocs de niveau 2.
"Fermer la fenêtre": ferme la fenêtre de messages.

Quand des messages d'erreur ou des occurrences sont affichées dans la fenêtre de messages, cliquez deux fois sur une ligne pour déplacer la sélection sur l'emplacement correspondant dans le programme. Pour les langages SFC et FC, cette commande ouvre la fenêtre de niveau 2 de l'élément référencé.

A.10 Le dictionnaire

Le **dictionnaire** d'ISaGRAF est un outil d'édition pour la déclaration des variables internes, d'entrée et de sortie, des instances de blocs fonctionnels, et des définitions de "mots équivalents", pour le projet sélectionné. Le dictionnaire regroupe l'ensemble des **variables** et **instances de blocs fonctionnels** déclarées, et des **définitions** de mots équivalents pour le projet sélectionné.
Les variables, les instances de blocs et les définitions doivent être déclarées dans le dictionnaire avant d'être utilisées dans le code source des programmes. Les variables et les définitions peuvent être utilisées dans tous les programmes écrits en langage SFC, FBD, LD, ST ou IL. Les blocs fonctionnels utilisés dans les diagrammes FBD n'ont pas besoin d'être déclarés. Les éditeurs FBD et Quick LD d'ISaGRAF déclarent automatiquement les instances de chaque bloc inséré dans le diagramme.

Variables

Les variables sont triées selon leur **portée** et leur **type**. Seules les variables de mêmes type et portée peuvent être saisies dans la même grille d'édition. Voici les portées prédéfinies pour les variables:

GLOBAL ..... utilisable par tous les programmes du projet

LOCAL ..... utilisable dans un seul programme

Voici les types de variables prédéfinis:

BOOLEEN ... valeurs binaires true/false

ANALOGIQUE valeurs entières ou réelles

TEMPORISATION valeurs temporelles

MESSAGE ... chaînes de caractères

Une variable est identifiée par un nom, un commentaire, un attribut, une adresse réseau et d'autres champs spécifiques. Voici les attributs prédéfinis pour les variables:
INTERNE ..... variable implantée en mémoire
ENTREE ..... variable reliée à un capteur
SORTIE..... variable reliée à un actionneur
CONSTANTE ..... variable interne en lecture seule (avec valeur initiale)

Note: Les **temporisations** sont toujours des variables **internes**. Les variables d'**entrée** et de **sortie** ont toujours une portée **GLOBALE**.

Mots équivalents

Les définitions de mots équivalents sont triées en fonction de leur portée. Seules les définitions d'une même portée pourront être entrées dans la même grille de saisie. Voici les portées prédéfinies pour les définitions:

COMMUN .... utilisable par les programmes de tous les projets

GLOBAL ..... utilisable par tous les programmes du projet

LOCAL..... utilisable par un seul programme

Une définition est identifiée par un nom, un texte ST équivalent, et un commentaire.

Instances de bloc fonctionnel

Les instances des blocs fonctionnels utilisés dans la programmation en langage ST doivent être déclarées dans le dictionnaire. Parce qu'un bloc fonctionnel contient des données internes "cachées", chaque copie d'un même bloc doit être identifiée. L'exemple suivant montre le bloc fonctionnel "R_TRIG" (détection de front montant) défini en librairie, et utilisé pour la détection de fronts de plusieurs variables. Chaque copie du bloc doit être identifiée par un nom unique. Le type de bloc fonctionnel est déclaré avec le Gestionnaire de Librairies:

Nom du bloc:	**R_TRIG**
***Paramètres*:**	**Input=CLK**
	Output=Q

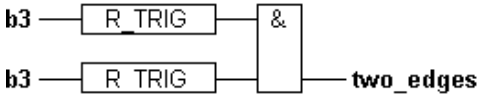
La déclaration des instances (copies) est faite dans le dictionnaire:

***Nom d'instance*:**	**TRIG_B1**	***Bloc*: R_TRIG**
***Nom d'instance*:**	**TRIG_B2**	***Bloc*: R_TRIG**

Les instances déclarées peuvent être utilisées dans le langage ST:

```
TRIG_B1 (b1);
edge_b1:= TRIG_B1.Q;      (* détection des fronts de la variable b1 *)
TRIG_B2 (b2);
edge_b2:= TRIG_B2.Q;      (* détection des fronts de la variable b2 *)
```

Les instances de blocs fonctionnels peuvent être déclarées en **GLOBAL** (connues par tous les programmes du projet), ou en **LOCAL** à un programme. Les blocs fonctionnels utilisés dans les diagrammes FBD n'ont pas à être déclarés, puisque les éditeurs FBD et Quick LD d'ISaGRAF déclarent automatiquement les instances de chaque bloc fonctionnel inséré dans le diagramme.



(* les blocs fonctionnels portent toujours le nom donné en librairie. Les éditeurs FBD et Quick LD d'ISaGRAF déclarent automatiquement une instance à chaque bloc inséré dans le diagramme *)

Les instances déclarées automatiquement par l'éditeur FBD d'ISaGRAF sont toujours **LOCALES** au programme édité.

Adresses réseau

L'adresse réseau d'une variable est une information **optionnelle**. Une variable avec une adresse réseau non nulle pourra être **espionnée** par un système externe (par

exemple un superviseur) à l'exécution. Les adresses réseau peuvent être saisies une à une, lors de la déclaration (création ou modification) de chaque variable.

A.10.1 La fenêtre du dictionnaire

La fenêtre du dictionnaire montre la liste des variables déclarées pour un même type et une même portée. Le type et la portée sélectionés sont toujours indiqués dans la barre de titre.

Les principaux champs de la déclaration d'une variable sont visibles dans la fenêtre principale: nom, attributs, adresse réseau et commentaire. La description détaillée de la variable sélectionnée est visible dans la barre d'état. Cliquez dans la zone de titre des colonnes pour changer leur taille.

Utilisez les boutons suivants pour sélectionner la portée des objets à éditer:

COMMUN utilisable par les programmes de tous les projets

GLOBAL utilisable par tous les programmes du projet

LOCAL utilisable par un seul programme

Utilisez les taquets suivants dans la barre de titre pour sélectionner le type d'objet à éditer:

Utilisez le champ de saisie à gauche de la barre d'outils pour accéder directement à une variable en entrant les premières lettres de son nom. Dans ce cas, la recherche est effectuée sur la liste entière, indépendamment de la position courante. La commande "**Edition / Chercher**" recherche une chaîne de caractères dans les noms et les commentaires de variables éditées, et emmène directement la barre de sélection sur la variable correspondante. La recherche ne différentie pas les majuscules et les minuscules.

A.10.2 Gestion des variables

Les commandes du menu "**Fichiers**" permettent de manipuler l'ensemble des variables, instances ou mots équivalents de la classe (portée / type) en cours d'édition. Utilisez la commande "**Autre...**" pour sélectionner un type et une portée d'objets à éditer.

Imprimer les déclarations

La commande "**Imprimer les variables**" lance l'impression de toutes les déclarations de variables ou définitions de mots équivalents de la liste (type / portée) sélectionnée. L'impression est réalisée par le générateur de dossier et montre la description détaillée de chaque variable, instance ou mot équivalent pour le groupe (type / portée) sélectionné.

Créer de nouvelles variables

La commande "**Nouveau**" du menu "**Edition**" permet de déclarer de nouvelles variables ou définitions de mots équivalents pour le type et la portée sélectionnés. Les nouveaux objets sont insérés juste avant l'objet sélectionné dans la liste. Quand cette commande est lancée, une boîte de dialogue est ouverte pour la saisie de la variable ou de la définition. Quand la description est complète, pressez le bouton "**Enregistre**" pour ajouter le nouvel objet à la liste. La boîte de dialogue est automatiquement ré-ouverte, ce qui autorise la saisie continue d'une liste d'objets par la même commande. Pressez le bouton "**Abandonne**" pour fermer la boîte de dialogue, quand la création des nouveaux objets est terminée.

Modifier les variables existantes

La commande "**Modifier**" du menu "**Edition**" permet de modifier la déclaration de l'objet sélectionné. Quand cette commande est lancée, une boîte de dialogue est ouverte pour la saisie de la variable ou de la définition. Quand la description est complète, pressez le bouton "**Enregistre**" pour valider les modifications. Vous pouvez également utiliser les boutons "**Suivant**" et "**Précédent**" pour étendre la commande de modification aux objets adjacents. Pressez le bouton "**Abandonne**" pour fermer la boîte de dialogue et terminer les modifications.

Couper et coller

L'éditeur de dictionnaire d'ISaGRAF permet la **sélection multiple des éléments**. Plusieurs commandes sont disponibles pour manipuler la liste d'objets (variables ou définitions) sélectionnés.

COPIER: ..... Copie les objets sélectionnés dans le presse papiers
COUPER:..... Copie les objets sélectionnés dans le presse papiers et les enlève de la liste affichée
EFFACER: ..... Supprime les objets sélectionnés dans la liste
COLLER: ..... Insère le contenu du presse papiers dans la liste affichée

Les commandes Copier / Couper / Coller peuvent être utilisées depuis une liste d'objets vers une autre liste ou un autre projet.

Trier les variables

La commande "**Outils / Trier**" trie tous les objets déclarés dans la liste sélectionnée. Voici l'ordre de tri retenu pour les variables:

- d'abord les variables internes
- ensuite les variables d'entrée
- enfin les variables de sortie

Les variables ayant le même attribut sont triées par ordre alphabétique. Les définitions de mots équivalents et les instances de blocs fonctionnels sont toujours triées par ordre alphabétique.

Positionner les adresses réseau

L'adresse réseau d'une variable est une information **optionnelle**. Une variable avec une adresse réseau non nulle pourra être **espionnée** par un système externe (par exemple un superviseur) à l'exécution. Les adresses réseau peuvent être saisies une à une, lors de la déclaration (création ou modification) de chaque variable. La

commande "**Outils / Renuméroter**" vous permet de forcer les adresses réseau de tout un groupe de variables. Quand cette commande est lancée, elle concerne les variables sélectionnées dans la liste. Il suffit d'entrer une **adresse de base** hexadécimale (adresse pour la première variable du groupe sélectionné). Les adresses de toutes les variables du groupe seront automatiquement forcées avec des **valeurs consécutives**.

Importer les mots équivalents à "true/false"

Lors de l'édition des mots équivalents, la commande "**Outils / Importer les équivalents Vrai/Faux**" permet de déclarer automatiquement comme mots équivalents les libellés définis pour les états à Vrai et à Faux des variables booléennes. Ces libellés ne sont normalement définis que pour l'affichage lors de la mise au point. Ils doivent faire l'objet de définition de mots équivalents pour pouvoir être utilisés dans les sources des programmes. Cette commande importe les libellés Vrai/Faux des variables booléennes ayant la même portée que celle des mots équivalents en cours d'édition.

A.10.3 Description des objects

Pour chaque variable, instance de bloc fonctionnel, ou définition de mot équivalent, il faut entrer sa description détaillée. Les champs à renseigner pour la description d'un objet dépendent de son type. Les champs suivants s'appliquent à tous les types de variables:

Nom.....Nom de la variable: le premier caractère doit être une lettre, les caractères suivants des lettres, des chiffres ou le souligné.
Adresse Réseau.....Adresse réseau hexadécimale (optionnelle). Si ce champ est non nul, la variable pourra être espionnée par un système externe à l'exécution.
Commentaire.....Commentaire libre pour la description de la variable.
Non volatile.....Cette option indique que l'état de la variable doit être sauvegardée en mémoire non volatile.

Voici les autres champs à renseigner pour une variable booléenne:

Attribut.....Spécifie une variable interne, constante, d'entrée ou de sortie.
Libellé "Faux".....Libellé utilisé pour remplacer la valeur "False" pendant la mise au point.
Libellé "Vrai".....Libellé utilisé pour remplacer la valeur "True" pendant la mise au point.
Vrai à l'init.....La valeur initiale de la variable est "True" si cette option est validée.

Voici les autres champs à renseigner pour une variable analogique:

Attribut.....Spécifie une variable interne, constante, d'entrée ou de sortie.

Format.....Spécifie une variable entière ou réelle (flottante) et le format d'affichage pendant la mise au point.
Libellé d'unité.....Libellé d'unité affiché pendant la mise au point.
Conversion.....Nom de la table ou fonction de conversion attachée à la variable (pour les entrées et sorties seulement).
Valeur initiale.....Valeur initiale de la variable. Doit avoir le même format (entier / réel) que la variable.

Voici les autres champs à renseigner pour une variable temporisation:

Attribut.....Spécifie une variable interne ou constante.
Valeur initiale.....Valeur initiale de la variable. (valeur de temps).

Voici les autres champs à renseigner pour une variable message:

Attribut.....Spécifie une variable interne, constante, d'entrée ou de sortie.
Longueur Max.....Spécifie le nombre maximum de caractères qui peuvent être stockés dans le message.
Valeur initiale.....Valeur initiale de la variable. Ne peut pas excéder la longueur maximale du message.

Voici les champs à renseigner pour une définition de mot équivalent:

Nom.....Nom utilisé dans les programmes: le premier caractère doit être une lettre, les caractères suivants des lettres, des chiffres ou le souligné.
Equivalence.....Chaîne respectant la syntaxe ST, qui remplacera le nom de la définition pendant la compilation. Exemple: Nom = PI - Equivalence = 3.14159
Commentaire.....Commentaire libre pour la définition.

Voici les champs à renseigner pour une instance de bloc fonctionnel:

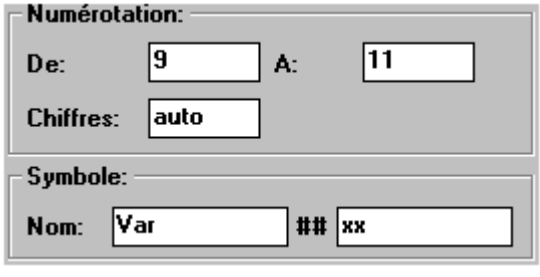
Nom.....Nom utilisé dans les programmes: le premier caractère doit être une lettre, les caractères suivants des lettres, des chiffres ou le souligné.
Type.....Nom en librairie du bloc fonctionnel asocié.
Commentaire.....Commentaire libre pour la définition.

A.10.4 Déclaration rapide

La commande "**Outils / Déclaration rapide**" permet de déclarer plusieurs variable en une seule fois. Les noms des variables ainsi créées sont basés sur une convention de numérotation. Pour ceci, vous devez définir:
- le numéro de la première et de la dernière variable,
- le texte à ajouter avant et après le numéro dans les noms de variables
- le nombre de chiffres utilisés pour exprimer le numéro dans les noms

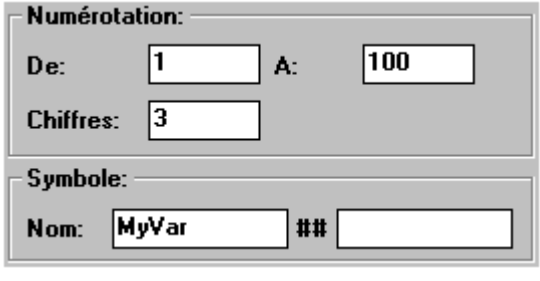
De plus, vous pouvez spécifier quelques attributs de base pour les variables à créer (interne, entrée ou sortie...), plus quelques paramètres dépendant du type de variable (attribut "Non volatile", format entier ou réel, longueur maximum des messages).
Il faut obligatoirement définir un texte à ajouter avant le numéro dans les symboles, puisqu'un nom de variable ne peut pas commencer par un chiffre. Quand l'option "Nombre de chiffres" est positionnée à "Auto", ISaGRAF formatte chaque numéro avec le nombre minimum de chiffres requis. Quand un nombre de chiffres est spécifié, ISaGRAF formatte chaque numéro selon la longueur demandée, en ajoutant des '0' devant la valeur du numéro. Demander un nombre fixe de caractères permet d'assurer un bon mécanisme pour les tris lexicographiques. Voici quelques exemples:

Ex. 1: Ce paramétrage pour la déclaration rapide:



créera les trois variables suivantes:
Var9xx Var10xx Var11xx

Ex 2: Ce paramétrage pour la déclaration rapide:



créera 100 variables avec des noms allant de MyVar001 à MyVar100

A.10.5 Carte d'adressage pour les superviseurs Modbus

Les "adresses réseau" définie par ISaGRAF sont souvent utilisées pour établir un lien entre les variables de la cible ISaGRAF et un superviseur basé sur une communication au protocole Modbus. Dans ce cas, le superviseur est un maître

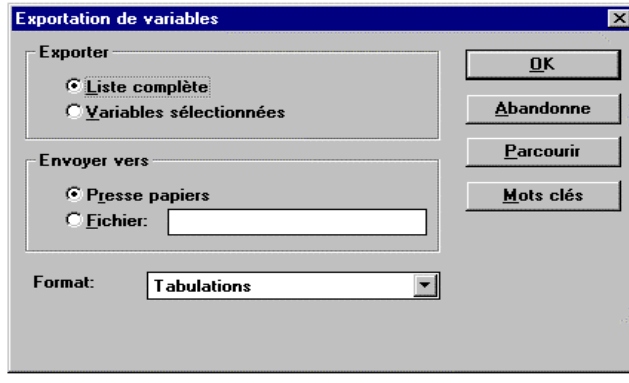
MODBUS et la cible ISaGRAF est un esclave MODBUS. Les adresses réseau sont utilisées pour construire une carte d'adressage MODBUS virtuelle pour toutes les variables devant être espionnées par le superviseur. La commande "**Outils / Carte d'adressage des superviseurs MODBUS** " est un outil dédié qui permet de créer rapidement la carte MODBUS des variables de votre application.
L'outil d'adressage montre deux listes. La liste du haut est un segment (4096 positions) de la carte Modbus, qui montre les variables adressées (celles qui ont une adresse réseau). La liste du bas montre les variables non adressées (sans adresse réseau). L'adresse "0" ne peut pas être utilisée.
Utilisez les commandes "**Connecter**" et "**Retirer**" du menu "**Edition**" pour déplacer une variable d'une liste vers l'autre, et construire la carte. Les mêmes actions peuvent être effectuées en cliquant deux fois sur le nom d'une variable, soit dans la carte, soit dans la liste du bas. A tout moment, vous pouvez utiliser la liste "Segment" pour sélectionner un autre segment de la carte.
Les commandes du menu "**Options**" permettent à tout moment de basculer entre un affichage en décimal et un affichage en hexadécimal.
La commande "**Edition / Chercher**" cherche un nom de variable déclarée, qu'elle soit adressée ou non.

A.10.6 Exchange avec d'autres applications

L'éditeur de dictionnaire d'ISaGRAF offre des fonctions **d'import / export** pour échanger des informations avec d'autres applications, telles que des traitements de textes, tableurs, gestionnaires de base de données... Ces commandes sont groupées dans le menu "**Edition**". La commande "**Exporter texte**" génère un texte ASCII de description des champs de déclaration des objets édités, et le stocke soit dans le presse papiers de Windows, soit dans un fichier. Ces informations sont typiquement destinées à être reprises par une autre application. La commande "**Importer texte**" importe les champs de déclaration d'objets, décrits sous forme de texte ASCII, soit dans le presse papiers de Windows soit dans un fichier, et met à jour ces champs dans les fiches des objets édités. Ces informations proviennent typiquement d'une autre application.

Exportation

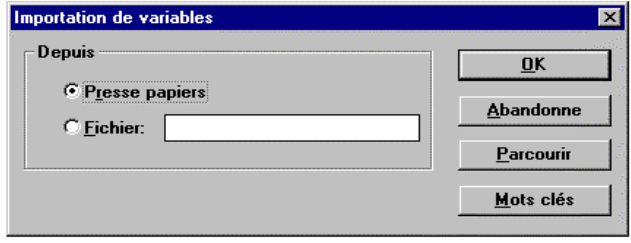
La boîte de dialogue d'exportation est ouverte quand la commande "**Exporter texte**" est lancée. Elle permet le contrôle du processus d'exportation.



Sélectionnez le choix "**Liste complète**" pour indiquer que toute la liste éditée doit être exportée. La sélection courante dans la liste est ignorée dans ce cas. Sélectionnez le choix "**Variables sélectionnées**" pour n'exporter que les objets sélectionnés dans la liste.
Si l'option "**Presse papiers**" est validée, le texte exporté est stocké, sous forme ASCII, dans le presse papiers de Windows. Le texte est alors disponible pour les commandes "**Coller**" d'autres applications. Si l'option "**Fichier**" est validée, le texte exporté est stocké, sous forme ASCII, dans un fichier. Le nom complet (nom + chemin) du fichier doit être entré dans ce cas. La commande "**Parcourir**" permet de retrouver un chemin existant.
Avant de lancer la fonction, il faut sélectionner le format choisi. Les formats disponibles sont décrits dans les sections suivantes. La commande "**Ok**" lance l'exportation. La commande "**Abandonne**" ferme la boîte de dialogue sans lancer la fonction.
Tous les champs concernant les objets édités sont écrits dans le texte exporté, dans leur ordre de déclaration. La première ligne du texte exporté contient le nom des champs. Chaque objet est décrit sur une ligne de texte. Le caractère de fin de ligne est la séquence MS-DOS standard "**0d-0a**". Les noms utilisés pour identifier les champs peuvent être redéfinis, à l'aide de la commande "**Mots clés**". Cette commande est décrite dans les sections suivantes.

Importation

La boîte de dialogue d'importation est ouverte quand la commande "Importer texte" est lancée. Elle permet le contrôle du processus d'importation.



Si l'option "**Presse papiers**" est validée, les informations importées sont lues dans le presse papiers de Windows, en format texte ASCII. Si l'option "**Fichier**" est validée, les informations importées sont lues dans un fichier ASCII. Le nom complet (nom + chemin) du fichier doit être entré dans ce cas. La commande "**Parcourir**" permet de retrouver un chemin existant.
La fonction d'importation reconnaît automatiquement le format (séparateurs) utilisé dans le texte importé. Les formats autorisés sont décrits dans les sections suivantes. La commande "**Ok**" lance l'importation. La commande "**Abandonne**" ferme la boîte de dialogue sans lancer la fonction.
La première ligne du texte doit contenir le nom des champs, dans le même ordre que pour les enregistrements suivants. Chaque objet doit être décrit sur une ligne de texte. Le caractère de fin de ligne est la séquence MS-DOS standard "**0d-0a**". Les champs peuvent apparaître dans n'importe quel ordre. Si certains champs sont absents, ils seront automatiquement complétés avec des valeurs par défaut. Si un objet importé existe déjà dans la liste éditée, l'utilisateur doit donner l'autorisation de l'écraser. La description de l'objet est alors mise à jour avec les champs importés. Si certains champs sont omis, ils ne sont pas mis à jour dans la fiche de l'objet.
Les noms utilisés pour identifier les champs peuvent être redéfinis, à l'aide de la commande "**Mots clés**". Cette commande est décrite dans les sections suivantes.

Formats supportés

Voici la liste des formats disponibles pour la fonction d'exportation. La commande d'importation reconnaît automatiquement ces formats.

- tabulations

Description: *Les champs sont séparés par des tabulations.*

Exemple:

Name	*Attribute*	*Comment*
level	*internal*	*internal calculated water level*
alrm1	*output*	*main alarm output*

- virgules

Description: *Les champs sont séparés par des virgules.*

Exemple:

Name,Attribute,Comment
level,internal,internal calculated water level
alrm1,output,main alarm output

- points-virgules

Description: *Les champs sont séparés par des points-virgules.*

Exemple:

Name;Attribute;Comment
level;internal;internal calculated water level
alrm1;output;main alarm output

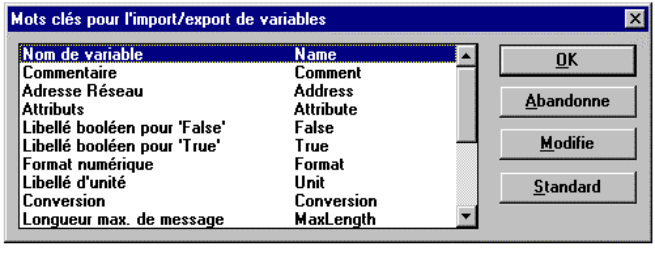
- virgules et guillemets

Description: *Les champs sont séparés par des virgules.*
Chaque champ est écrit entre guillemets.

Exemple: *"Name","Attribute","Comment"*
"level","internal","internal calculated water level"
"alrm1","output","main alarm output"

Mots clés

Les noms utilisés pour identifier les champs dans la première ligne des textes importés ou exportés peuvent être redéfinis, à l'aide de la commande "**Mots clés**" des boîtes de dialogue d'importation et d'exportation.



La fenêtre contient la liste de tous les champs, avec les mots clés associés. Pour modifier un mot clé, il suffit de sélectionner le champ dans la liste et de presser le bouton "**Modifie**". La commande "**Standard**" restitue tous les mots clés originaux. La nomenclature des mots clés doit respecter les règles suivantes:

- la longueur du nom ne doit pas excéder 16 caractères
- le premier caractère doit être une lettre
- les caractères suivants doivent être des lettres, des chiffres ou '_'
- le même nom ne peut pas être utilisé pour plusieurs champs

Voici la liste des mots clés standards proposés par ISaGRAF:

Nom de variable, d'équivalence, d'instance ..... **Name**
Commentaire ..... **Comment**
Adresse réseau ..... **Address**
Attributs (interne, entrée, sortie) ..... **Attribute**
Libellé équivalent à 'False' ..... **False**
Libellé équivalent à 'True' ..... **True**
Format analogique (entier ou réel) ..... **Format**
Libellé d'unité analogique ..... **Unit**
Nom de conversion analogique ..... **Conversion**
Longueur maximum de message ..... **MaxLength**
Type de bloc fonctionnel en librairie ..... **Library**
Equivalence d'un mot défini ..... **Equivalence**
Attribut: interne ..... **Internal**
Attribut: constante ..... **Constant**
Attribut: entrée ..... **Input**
Attribut: sortie ..... **Output**

Format analogique: réel..... **Real**
Format analogique: entier..... **Integer**

A.11 Câblage des E/S

Le but du **câblage des Entrées/Sorties** est d'établir une correspondance logique entre les **variables d'E/S** de l'application et les voies physiques des cartes d'E/S implantées sur l'automate cible. Pour réaliser cette opération, il faut identifier chaque carte de l'automate, et relier les variables d'E/S aux voies des cartes définies.
La liste de gauche montre le rack de l'automate cible, avec les **emplacements des cartes**. Un emplacement peut être libre ou équipée par une carte d'E/S ou un équipement complexe. Chaque emplacement est identifié par un **numéro d'ordre**. Le rack peut contenir jusqu'à **255** cartes. La liste de droite montre les paramètres de la carte sélectionnée, et les variables câblées sur ses voies. Une carte peut grouper jusqu'à **128** voies d'E/S. Le nombre de cartes simples (incluant les cartes simples et les cartes des équipements complexes) ne peut pas excéder **255**.

Icones

Les icones affichés sur la face avant des cartes identifient le type et l'attribut des variables qui peuvent être connectées sur ses voies. Le système ISaGRAF ne permet pas le câblage de variables de différents types sur la même carte. Voici la signification des icones:

..... type booléen
..... type entier ou réel (pas de distinction pour le câblage)
..... type message
..... entrées - pas de voie câblées
..... sorties - pas de voie câblées
..... entrées - au moins une voie câblée
..... sorties - au moins une voie câblée

Voici les icones utilisés pour indiquer le type d'équipement d'E/S installé sur un emplacement:

..... équipement d'E/S complexe
..... carte d'E/S réelle
..... carte d'E/S virtuelle

Voici les icones pour représenter les paramètres et les voies des cartes:

..... paramètre
..... voie libre
..... voie utilisée (connectée à une variable)

Déplacer une carte dans la liste

Utilisez ces boutons de la barre d'outils ou les commandes "**Edition / Déplacer...**" pour déplacer la carte d'E/S vers le haut ou vers le bas dans la liste des cartes. La commande "**Edition / Insérer**" insère un espace libre avant la carte sélectionnée.

A.11.1 Definir les équipements d'E/S

Le menu "**Edition**" regroupe les principales commandes permettant de définir le type des cartes installées et de câbler les variables d'E/S sur les voies des cartes.

Selectionner un type de carte d'E/S

Avant de câbler les variables d'E/S, il faut définir chaque carte ou équipement installé sur le rack. Une librairie de cartes et d'équipements complexes est disponible dans l'atelier ISaGRAF. Cette librairie peut regrouper les descripteurs de cartes provenant de divers constructeurs. La commande "**Edition / Définir carte/équipement**" permet de sélectionner un matériel d'E/S dans la librairie. Cette commande permet de sélectionner une carte unitaire ou un équipement complexe. Vous pouvez également cliquer deux fois sur un emplacement pour définir le matériel d'E/S associé.
Toutes les voies d'une carte unitaire doivent avoir le même type (booléen, analogique ou message) et la même direction (entrée ou sortie). Les analogiques entiers et réels ne sont pas distingués pendant le câblage des E/S. Un équipement d'E/S complexe représente un dispositif d'E/S avec des voies de types et de directions différents. Un équipement complexe est représenté commme un ensemble de cartes unitaires. Il n'occupe qu'un seul emplacement dans le rack.

Supprimer une carte

La commande "**Edition / Libérer l'emplacement**" retire la carte ou l'équipement complexe installé sur l'emplacement sélectionné. Si des variables sont déjà câblées sur les voies correspondantes, elles sont automatiquement déconnectées.

Cartes réelles et virtuelles

La commande "**Edition / Carte réelle/virtuelle**" force la validité de la carte ou de l'équipement installé sur l'emplacement sélectionné. Les icones suivants sont utilisés pour indiquer la validité des équipements:
 ..... carte d'E/S réelle
 ..... carte d'E/S virtuelle
Dans le **Mode Réel**, les variables d'E/S sont directement reliées aux capteurs et actionneurs correspondants. Les opérations de lecture et d'écriture définies dans l'application entraineront des acquisitions et des mises à jour des dispositifs d'E/S correspondants. Dans le **Mode Virtuel**, les variables d'E/S sont traitées exactement comme des variables internes. Elles peuvent alors être lues ou modifiées par le debugger. Vous pouvez ainsi simuler le fonctionnement d'une carte d'E/S non encore disponible.

Fiches techniques

La commande "**Outils / Fiche technique**" permet de consulter la notice d'utilisation de la carte ou de l'équipement sélectionné. La fiche technique de la carte ou de l'équipement est écrite par son constructeur. Elle contient toutes les informations concernant son utilisation, et indique la signification de chacun de ses paramètres.

Retirer les variables câbles

La commande "**Outils / Libérer les voies**" déconnecte toutes les variables actuellement câblées sur les voies de la carte sélectionnée.

Entrer des commentaires pour les voies non utilisées

Le commentaire d'une variable d'E/S déclarée dans le dictionnaire est affiché avec son symbole dans la fenêtre de câblage. ISaGRAF supporte les variables représentées directement (notation %), et autorise donc la saisie de commentaire pour les voies non connectées. Utilisez la commande "**Outils / Commentaire de la voie**" pour saisir le commentaire à attacher à la voie sélectionnée. Cette commande n'est pas valide si une variable déclarée est connectée sur cette voie.

A.11.2 Entrer les paramètres d'une carte

Pour forcer la valeur d'un paramètre, il suffit de cliquer deux fois sur son nom dans la liste de droite. Il est également possible de le sélectionner puis de lancer la commande "**Edition / Définir voie/paramètre**" . Les paramètres sont groupés au début de la liste de droite, avant les voies de la carte. L'icone suivant les indique dans la liste:

..... paramètre d'une carte

Les paramètres d'une carte d'E/S dépendent de son type. Ceux-ci sont définis par le constructeur de la carte. Référez-vous à la fiche technique de la carte d'E/S pour plus d'information sur la signification des paramètres et comment les assigner. Utilisez la commande "**Outils / Fiche techique**" pour consulter la notice de la carte sélectionnée.

A.11.3 Câbler les voies d'E/S

Quand une carte est définie, il faut câbler les variables d'E/S de l'application sur ses voies. Pour câbler une voie, il suffit de cliquer deux fois sur l'emplacement correspondant dans la liste de droite. Il est également possible de le sélectionner puis de lancer la commande "**Edition / Définir voie/paramètre**". Les icones suivants identifient les voies de la carte dans la liste:

..... voie libre

..... voie utilisée (connectée à une variable)

La liste ne contient que les variables cohérentes avec le type et la direction de la carte sélectionnée. Seules les variables non encore câblées apparaissent dans la liste. Le bouton "**Câbler**" câble la variable sélectionnée dans la liste sur la voie choisie. Le bouton "**Libère**" retire (déconnecte) la variable câblée sur la voie sélectionnée. Les boutons "**Suivant**" et "**Précédent**" permettent de sélectionner les autres voies de la carte. L'emplacement de la voie sélectionnée est toujours indiqué dans le titre de la boîte de dialogue.

A.11.4 Variables représentées directement

Les voie libres sont celles qui ne sont pas connectées avec des variables d'E/S déclarées. ISaGRAF permet l'utilisation des **variables représentées directement**

dans le source des programmes pour représenter l'accès à une voie d'E/S libre. L'identificateur d'une variable représentée directement commence toujours par le caractère "**%**".
Voici les conventions d'écriture des identificateur pour les variables représentées directement dans le cas d'une carte simple. Dans les notations suivantes, "***s***" est le numéro de l'emplacement de la carte, et "***c***" est le numéro de la voie.

%IXs.c ..... canal d'une voie booléenne d'une carte d'entrée
%IDs.c ..... canal d'une voie entière d'une carte d'entrée
%ISs.c ..... canal d'une voie message d'une carte d'entrée
%QXs.c ..... canal d'une voie booléenne d'une carte de sortie
%QDs.c ..... canal d'une voie entière d'une carte de sortie
%QSs.c ..... canal d'une voie message d'une carte de sortie

Voici les conventions d'écriture des identificateur pour les variables représentées directement dans le cas d'un équipement complexe. Dans les notations suivantes, "***s***" est le numéro de l'emplacement de l'équipement, "***b***" est l'index de la carte dans l'équipement, et "***c***" est le numéro de la voie.

%IXs.b.c ..... entrée booléenne
%IDs.b.c ..... entrée entière
%ISs.b.c ..... entrée message
%QXs.b.c ..... sortie booléenne
%QDs.b.c ..... sortie entière
%QSs.b.c ..... sortie message

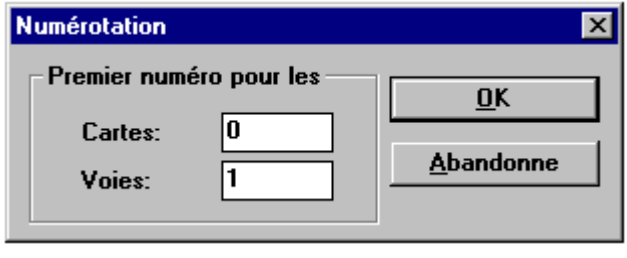
Voici quelques exemples:

%QX1.6 ..... 6ème voie de la carte #1 (sortie booléenne)
%ID2.1.7 ..... 7ème voie de la carte #1 de l'équipement #2 (sortie analogique)

Une variable représentée directement ne peut pas avoir le type "**analogique réel**".

A.11.5 Numérotation

Utilisez la commande "**Options / Numérotation**" pour définir les conventions de numérotation des slots et des voies. Vous pouvez définir l'index du premier slot et celui de la première voie de chaque carte dans la boîte suivante:



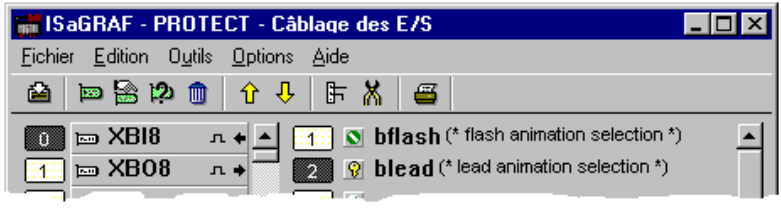
Par défaut, la numérotation des slots commence à **0**, et celles des voies d'une carte à **1**.

Attention: soyez très prudent lorsque vous changez les conventions de numérotation. Ceci a une influence directe sur la symbolique des variables représentées directement (notation "%") et peut engendrer des erreurs de syntaxe dans les programmes existants.

A.11.6 Protections individuelles

L'atelier ISaGRAF offre un système de protection basé sur un ensemble de mots de passe hiérarchisés. Le câblage des E/S peut être globalement verrouillé par un mot de passe. De plus, ISaGRAF permet d'installer une protection individuelle sur chaque voie d'E/S. Ceci présume que:
- des mots de passes sont déjà définis dans le système de protection de l'application (utilisez la commande "**Projet / Mot de passe**" du gestionnaire de projets) et que des niveaux de protections valides sont disponibles pour les protections.
- vous utilisez des niveaux de protection individuelle plus prioritaires que les niveaux utilisés pour la protection globale du câblage.

Quand une voie d'E/S a une protection individuelle, un icone spécial est affiché à côté de son nom dans l'éditeur de câblage:



Utilisez les commandes "**Protéger**" et "**Retirer la protection**" du menu "**Edition**" pour installer ou retirer une protection individuelle sur la voie sélectionnée. Ces deux commandes vous demandent de saisir un mot de passe valide correspondant à un niveau de protection correctement défini. Par la suite, chaque fois que vous voudrez changer le câblage d'une voie protégée, vous devrez entrer un mot de passe suffisant.

Attention: si une voie est protégée avec un niveau, et que le mot de passe correspondant est supprimé dans le système de protection, et s'il n'existe pas de mot de passe plus prioritaire, la voie ne pourra plus être modifiée jusqu'à la définition de nouveaux mots de passe de niveau suffisant.

A.12 Tables de conversion

L'atelier ISaGRAF permet de créer des tables de conversion. Une table de conversion est un ensemble de points qui définissent une conversion analogique. Une table de conversion peut être affectée à chaque variable d'entrée ou de sortie analogique. La même table peut être affectée à plusieurs variables. Une table donne la correspondance entre les valeurs électriques (lues sur un capteur ou envoyées à un actionneur) et les valeurs physiques traitées dans les programmes de l'application.
Les tables de conversions sont éditées dans une boîte de dialogue ouverte par la commande "**Outils / Tables de conversion**" du dictionnaire (éditeur de variables) d'ISaGRAF.
Les tables de conversion définies peuvent être utilisées pour filtrer les valeurs des variables d'entrée ou de sortie analogique du projet. L'affectation d'une table de conversion à une variable est faite depuis l'éditeur du dictionnaire, lors de l'édition des paramètres d'une variable d'E/S analogique. Une variable ne peut pas être filtrée par une table de conversion non encore définie.

A.12.1 Principales commandes

La boîte de dialogue des Tables de conversions montre la liste des tables définies, et contient les boutons permettant d'éditer une table (définir ses points), créer une nouvelle table, renommer ou supprimer une table existante.
Pressez le bouton "**OK**" pour enregistrer la définition des tables et fermer la boîte de dialogue.

Créer une nouvelle table

La commande "**Nouveau**" permet de créer une nouvelle table de conversion. Au maximum **127** tables de conversion peuvent être créées pour le même projet. Seules les tables utilisées (celles affectées à des variables analogiques) seront insérées dans le code de l'application. Les règles suivantes doivent être respectées pour nommer une table:

- la longueur du nom ne peut pas excéder **16** caractères
- le premier caractère doit être une **lettre**
- les caractères suivants doivent être des **lettres**, des **chiffres** ou '_'
- les minuscules et les majuscules ne sont pas différenciées

Modifier une table

Utilisez la commande "**Editer**" pour modifier les points de la table sélectionnée. Vous pouvez également cliquer deux fois sur le nom de la table dans la liste. La commande de définition des points est automatiquement activée quand une nouvelle table est créée. Une table doit toujours contenir au moins deux points pour être définie.

A.12.2 Saisie des points d'une table

La boîte de dialogue "édition" est utilisée pour définir les points d'une table de conversion. La boîte de gauche montre les points déjà définis. La boîte centrale permet de créer de nouveaux points. La boîte en bas et à droite montre l'aspect de la courbe correspondante. Il suffit d'entrer la valeur de chaque point pour définir la table, en respectant les règles de construction imposées par ISaGRAF. La boîte de gauche montre les points définis de la table éditée. La colonne de gauche montre la valeur électrique (externe) des points. La colonne de droite montre la valeur physique (interne) des points. Il faut sélectionner un point de la table pour modifier sa valeur ou l'effacer (le retirer de la table). Le dernier choix de la liste ("**... ...**") doit être sélectionné pour la création de nouveaux points. La boîte en bas et à droite montre l'aspect graphique de la courbe définie par la table de conversion. Aucun axe ou coordonnée n'est indiqué. Le but de cette représentation est de visualiser simplement les éventuelles erreurs de construction (changement de croissance) de la table.

Définir un nouveau point

Pour définir un nouveau point, il faut sélectionner le dernier choix de la liste des points ("**... ...**"). C'est le mode par défaut lors de la définition d'une nouvelle table. Il faut entrer les valeurs électrique (externe) et physique (interne) du point. Les valeurs sont stockées comme des nombres flottants en simple précision. Il faut au moins **deux points** pour définir une table. Quand les valeurs électrique et physique sont saisies, pressez le bouton "**Enregistre**" pour ajouter le point à la table. Une table ne peut pas contenir plus de **32** points.

Modifier un point

Pour modifier la valeur d'un point existant, il faut le sélectionner dans la liste des points. Ensuite, il suffit d'entrer les nouvelles valeurs électrique (externe) et physique (interne) du point. Les valeurs sont stockées comme des nombres flottants en simple précision. Quand les valeurs électrique et physique sont saisies, pressez le bouton "**Enregistre**" pour enregistrer la modification. Une table ne peut pas contenir plus de **32** points.

Supprimer un point

Pour supprimer un point existant, il faut le sélectionner dans la liste des points, et presser le bouton"**Efface**". Il faut au moins **deux points** pour définir une table.

A.12.3 Règles et limites

Les règles suivantes doivent être respectées lors de la construction d'une table, pour que la conversion puisse être utilisée aussi bien en entrée qu'en sortie:

- deux points ne peuvent pas avoir la même valeur électrique
- la courbe doit être strictement croissante ou décroissante
- deux points ne peuvent donc pas avoir la même valeur physique

Les limites suivantes doivent être respectées pour la définition des tables de conversion d'un projet:

- Il y a au maximum **127** tables de conversion pour le projet

- Il y a au maximum **32** points dans une table de conversion

A.13 Génération du code

La fenêtre du générateur de code est automatiquement ouverte par les commandes "**Vérifier**" et "**Générer l'application**" des autres fenêtres de l'atelier ISaGRAF. Elle n'est pas automatiquement refermée à la fin de la fonction requise, ce qui permet à l'utilisateur l'accès aux autres commandes de génération de code et de paramétrage du code fabriqué.

A.13.1 Principales commandes

Le menu **"Fichier"** groupe les commandes de génération du code et de vérification syntaxique des programmes.

- ***Générer le code de l'application***

  La commande "**Générer**" construit le code cible du projet ouvert. Avant la génération, cette commande vérifie la syntaxe des déclarations et des programmes. Toute erreur qui ne peut pas être détectée lors de la vérification isolée d'un programme est détectée à la génération du code. Ceci s'applique aux tables de conversion, le câblage des variables d'E/S et les liens avec les librairies. La génération de code s'arrête après le premier programme contenant une erreur. Ce programme doit être corrigé avant de continuer la génération. Les programmes déjà vérifiés (sans erreur) et qui n'ont pas été modifiés ne sont pas re-vérifiés. La vérification du dictionnaire est toujours effectuée. Pendant la vérification des programmes, la commande "**Générer**" peut être interrompue en frappant la touche "**ECHAP**".

Note: Si la déclaration d'une variable locale à un programme a été modifiée, le programme correspondant est vérifié. Si une variable globale est modifiée, tous les programmes sont vérifiés (de même pour les définitions de mots équivalents)

- ***Vérifier la syntaxe d'un programme***

  La commande "**Vérifier un programme**" lance la vérification d'un seul programme, même s'il n'a pas été modifié. La commande "**Vérifier le dictionnaire**" permet de lancer de façon isolée la vérification du dictionnaire de variables du projet. La commande "**Vérifier tous les programmes**" lance la vérification de tous les programmes du projet, même s'ils n'ont pas été modifiés. Cette commande ne s'arrête pas lorsque des erreurs sont détectées dans un programme. Elle peut donc être utilisée pour produire un listing complet de toutes les erreurs subsistant dans le projet. Cette commande peut être interrompue en frappant la touche "**ECHAP**".

- ***Simuler une modification***

  La commande "**Simuler une modification**" met à jour tous les programmes sans modifier leur contenu, de façon à ce qu'ils soient tous vérifiés lors de la prochaine génération de code. La commande "**Ouvrir**" lance l'édition du dernier programme vérifié. Cette commande permet un accès rapide aux erreurs détectées lors de la vérification d'un programme.

A.13.2 Options de compilation

La commande "**Options de compilation**" permettent de configurer les paramètres utilisés par le générateur de code d'ISaGRAF pour construire et optimiser le code exécutable. Le but de cette commande est de choisir les cibles ISaGRAF pour lesquelles le code doit être généré, et de positionner les paramètres d'optimisation pour établir le meilleur compromis entre le temps de compilation et les performances de l'application à l'exécution.
Le bouton "**Télélecture**" ouvre une seconde boîte pour la saisie des options concernant l'embarquement des sources compactées avec le code téléchargé, de façon à permettre la récupération de l'application par télélecture. Référez-vous au chapître "**Télélecture**" pour plus d'information.

Sélectionner le type d'automate cible

La liste du haut montre les différentes cibles ISaGRAF supportées. Le signe "**>>**" indique le(s) cible(s) sélectionnée(s). Le générateur de code d'ISaGRAF peut construire parallèlement jusqu'à **3** types de code différents. Utilisez les boutons "**Sélectionner**" et "**Libérer**" pour indiquer quel type de code doit être produit, en accord avec le type de logiciel ISaGRAF installé sur votre cible. Voici la liste des cibles ISaGRAF:

SIMULATE:..... Ce code est dédié au Simulateur de l'atelier ISaGRAF. Le simulateur ne pourra pas être lancé si cette cible n'a pas été sélectionnée poour la génération du code.

ISA86M: ..... C'est un code TIC (Target Independent Code) dédié aux noyaux exécutifs ISaGRAF installés sur des processeurs de type Intel. Le type de processeur concerne l'ordre des octets de poids fort et faible dans le code généré.

ISA68M: ..... C'est un code TIC (Target Independent Code) dédié aux noyaux exécutifs ISaGRAF installés sur des processeurs de type Motorola. Le type de processeur concerne l'ordre des octets de poids fort et faible dans le code généré.

SCC: ..... La sélection de cette cible indique qu'ISaGRAF doit générer l'application sous la forme de code source "C" structuré qui sera compilé et lié avec les autres librairies du noyau cible ISaGRAF pour produire un exécutable.

CC86M:..... La sélection de cette cible indique qu'ISaGRAF doit générer l'application sous la forme de code source "C" non structuré qui sera compilé et lié avec les autres librairies du noyau cible ISaGRAF pour produire un exécutable. Cette sélection est fournie pour la compatibilité avec les systèmes antérieurs à la version V3.23 d'ISaGRAF, quand la génération de code "C" structuré n'était pas supportée.

Référez-vous au manuel d'utilisation de votre automate pour connaître le type de code à générer. D'autres types de code (code machine, code source X...) pourront être supportés dans les versions futures de l'atelier ISaGRAF.

Traduction du SFC

Validez l'option "**Utiliser le moteur SFC embarqué**" pour valider l'utilisation du moteur SFc dans la cible ISaGRAF. ***Ce mode est préférable, car il induit en général de meilleures performances***. Toutefois, le moteur SFc peut être absent dans certaines implémentations de la cible ISaGRAF, ou plus particulièrement sur les implémentations d'ISaGRAF basées sur une post-compilation du code. Dans ce cas, vous devez retirer cette option, et ISaGRAF traduit tous les graphes SFC en instructions classiques. Référez-vous au guide d'utilisation de votre automate pour plus d'information sur l'utilisation de cette option.

Options d'optimisation

Cette section présente les paramètres utilisés par le générateur de code d' ISaGRAF pour optimiser le code généré. Ces paramètres peuvent être configurés à l'aide la boîte de dialogue "**Options de compilation**". Le bouton "**Par défaut**" permet de retirer toutes les options d'optimisation, afin de réduire le temps de compilation des programmes.

- Quand l'option "**Deux passes**" est sélectionnée, L'optimiseur de code d'ISaGRAF est lancé deux fois de suite. Les optimisations réalisées lors de la seconde passe sont généralement moins significatives que celles réalisées dans la première passe.
- Quand l'option "**Expressions constantes**" est sélectionnée, les expressions numériques constantes sont évaluéess par le compilateur. Par exemple, l'expression "**2 + 3**" est remplacée par "**5**" dans le code cible. Quand cette option n'est pas sélectionnée, les expressions sont calculées pendant l'exécution de l'application.
- Quand l'option "**Etiquettes inutilisées**" est sélectionnée, l'optimiseur simplifie le système de sauts et d'étiquettes des programmes compilés, afin de supprimer les éventuels sauts "nuls" et étiquettes inutilisées.
- Quand l'option "**Copie des variables**" est sélectionnée, l'utilisation des variables temporaires (utilisées pour stocker les résulltats intermédiaires) est optimisée. Cette option est généralement utilisée avec l'option "**Expressions**". Quand cette option est sélectionnée, l'optimiseur cherche à ré-utiliser les expressions ou sous-expressions qui apparaissent plusieurs fois dans le programme.
- Quand l'option "**Code inutilisé**" est sélectionné, l'optimiseur supprime les portions de code non significatives. Par exemple, si les instructions suivantes sont programmées: "**var:= 1; var:= X;**", le code généré est seulement: "**var:= X;**".
- Quand l'option "**Opérations arithmétiques**" est sélectionnée, l'optimiseur simplifie les opérations numériques en fonction des opérandes particuliers. Par exemple, l'expression "**A + 0**" est remplacée par "**A**". Quand l'option "**Opérations booléennes**" est sélectionnée, l'optimiseur simplifie les opérations logiques en fonction des opérandes particuliers. Par exemple, l'expression "**A & A**" est remplacée par "**A**".
- Quand l'option "**Diagrammes binaires**" est sélectionnée, l'optimiseur remplace les équations booléennes (mêlant les opérateurs **AND**, **OR** , **XOR** et **NOT**), par une liste réduite de sauts conditionnels. La traduction n'est effectuée que si le temps d'exécution attendu pour la liste de sauts est plus faible que celui attendu pour l'expression originale.

La table suivante présente un résumé des temps et des résultats d'optimisation attendus, en fonction des différents paramètres:

	gain (performances)	*temps de compilation*
Deux passes	**xxxx**	(*)
Expressions constantes	**xxxxxxxx**	**xxxx**
Etiquettes inutilisées	**xxxx**	**xxxxxxxx**
Copie de variables	**xxxx**	**xxxxxxxx**
Expressions	**xxxx**	**xxxxxxxx**
Code inutilisé	**xxxx**	**xxxxxxxx**
Opérations arithmétiques	**xxxxxxxx**	**xxxx**
Opérations booléennes	**xxxxxxxx**	**xxxx**
Diagrammes binaires	**xxxxxxxxxxx**	**xxxxxxxxxxx**

(*) le temps de compilation est également doublé.

A.13.3 Génération de code source "C"

L'atelier ISaGRAF supporte la génération de code source en langage "C". Dans ce cas, le contenu total de l'application est traduit en code source C. Ceci inclut la description des graphes SFC, la définition de la base de données et les séquences de code. Il y a deux possibilités, correspondant à deux styles de code "C" généré:

CC86M (C source code - V3.04) produit un code source "C" non structuré. Vous devez choisir ce style si votre cible ISaGRAF est antérieure à la version V3.23.
SCC (structure C source code) produit un code source "C" structuré. Ce style doit être préféré si votre cible ISaGRAF est une version V3.23 ou plus récente.

Les fichiers suivants sont créés dans le répertoire du projet:
APPLI.C ..... code source commun de l'application
APPLI.H ..... définitions communes en langage "C" language

Dans le cas de la génération de code structuré, un fichier source ".C" et un fichier de définitions ".H" est créé pour chaque programme du projet, en plus des fichiers communs "APPLI.C" et "APPLI.H".
Ces fichiers doivent être compilés et liés avec les librairies du noyau cible ISaGRAF pour produire le code exécutable final. Référez-vous au guide d'utilisation de la cible ISaGRAF pour plus d'information sur la procédure d'intégration et les techniques de compilation.

Note: Certaines fonctionalités de mise au point, telles que le téléchargement, la modification en ligne et les points d'arrêt ne sont pas disponibles dans le cas d'une application générée en code C.

A.13.4 Affichage

Le menu "**Edition**" groupe les commandes permettant de visualiser les fichiers de texte créés pendant la vérification des programmes et la génération du code. La fenêtre de génération de code est une zône de texte qui accueille les messages de la génération de code et de la vérification syntaxique. Ces informations sont

également stockées sur le disque et peuvent donc être rappelées à l'aide des commandes du menu "**Edition**".

Commandes d'édition

La commande "**Effacer l'écran**" efface le texte affiché. La fenêtre est automatiquement effacée à chaque nouvelle commande de vérification ou de génération du code. La commande "**Copier**" copie le texte affiché dans le presse-papiers de Windows, de façon à ce qu'il puisse être repris par d'autres applications telles que les éditeurs de texte d'ISaGRAF.

Messages du compilateur

La commande "**Messages d'erreur**" affiche les messages de la dernière opération de vérification syntaxique ou de génération de code, et en particulier la description des erreurs de syntaxe détectées.
D'autres choix du menu "**Edition**" permettent de consulter les fichiers de texte intermédiaires créés pendant la vérification et la génération du code. La consultation de ces fichiers n'est généralement pas nécessaire.

A.13.5 Définition des ressources

La commande "**Ressources**" du menu "**Options**" permet de définir les ressources. Une ressource est un ensemble de données défini par l'utilisateur (configuration de réseau, paramètrage du matériel...) de tout format (fichier, série de valeurs) qui doit être ajouté au code généré, de façon à être téléchargé avec lui dans l'automate cible. Ces données ne sont pas directement traitées par le noyau ISaGRAF, et sont généralement dédiées à d'autres logiciels installés dans l'automate. Référez-vous au manuel d'utilisation de votre automate pour plus d'informations sur le type de ressource supportée ou attendue.

Le fichier de définition des ressources

Les ressources sont définies dans un "**Fichier de définition des ressources**" stocké avec les autres fichiers du projet ISaGRAF. C'est un fichier de texte ASCII, analysé par le Compilateur de Ressources d'ISaGRAF. Le compilateur de ressources est lancé automatiquement lors de la génération du code. Cette section présente la syntaxe de ce fichier. Cette syntaxe suit les règles lexicales du langage ST. Des commentaires, commençant par "(*" et finissant par "*)" peuvent être insérés partout dans le texte. Les chaînes de caractères sont délimitées par des apostrophes. Référez-vous à la deuxième partie de ce manuel pour une présentation détaillée du format des constantes numériques.

Référence du langage

Voici la liste des mots clés et des énoncés utilisés dans le fichier de définition des ressources.

ULONGDATA

Rôle: Spécifie une ressource qui contient une série de valeurs entières. Les valeurs sont stockées dans le code cible comme des entiers non

signés sur 32 bits. Les valeurs sont stockées dans l'ordre indiqué dans le fichier de définition. Les valeurs doivent être séparées par des virgules. Le nom de la ressource ne doit pas excéder 15 caractères.

Syntaxe:
```
ULONGDATA '<nom_ressource>'
BEGIN
  ...sélection de cibles...
  ...liste de valeurs...
  END
```

Exemple:
```
ULongData 'MYDATA'
Begin
  ...
  0, -1, 100_000,        (* decimal *)
  16#A0B1, 2#1011_0101   (* hexadecimal, binary *)
End
```

VARLIST

Rôle: Spécifie une ressource qui est une liste d'adresses de variables. Les variables sont identifiées par leur symbole dans le fichier de définition des ressources. Les adresses sont stockées dans le code de la ressource comme des entiers de 16 bits. Les adresses sont stockées dans l'ordre où elles apparaissent dans le fichier de définition. Les noms de variables doivent être séparés par des virgules. Le nom de la ressource ne doit pas excéder 15 caractères.

Syntaxe:
```
VARLIST '<nom_ressource>'
BEGIN
  ...sélection des cibles...
  ...liste des noms de variables...
  END
```

Exemple:
```
VarList 'LIST'
Begin
  ...
  Var100, MyParameter, Command, Alarm
End
```

BINARYFILE

Rôle: Spécifie une ressource dont le contenu provient d'un fichier binaire MS-DOS. La définition de la ressource sur la cible est complétée par un nom de chemin. Les caractères de fin de ligne ne sont pas convertis par le compilateur de ressources. Le nom de la ressource ne peut pas excéder 15 caractères.

Syntaxe:
```
BINARYFILE '<nom_ressource>'
BEGIN
  ...sélection des cibles...
  FROM '<chemin_source>'
  TO '<chemin_destination>'
  END
```

```
Exemple:   BinaryFile 'MYFILE'
           Begin
             ...
             From 'c:\user\config.bin'
             To '/dd/user/appl/config.dat'
           End
```

TEXTFILE

Rôle: Spécifie une ressource dont le contenu provient d'un fichier ASCII de MS-DOS. La définition de la ressource sur la cible est complétée par un nom de chemin. Les caractères de fin de ligne sont automatiquement convertis par le compilateur de ressources selon les conventions du système cible. Le nom de la ressource ne peut pas excéder 15 caractères.

```
Syntaxe:   TEXTFILE '<nom_ressource>'
           BEGIN
             ...sélection des cibles...
             FROM '<chemin_source>'
             TO '<chemin_destination>'
           END

Exemple:   TextFile 'MYFILE'
           Begin
             ...
             From 'c:\user\config.bin'
             To '/dd/user/appl/config.dat'
           End
```

TARGET

Rôle: Spécifie le nom d'un code cible qui doit contenir la ressource. Référez-vous à la section précédente (options de compilation) pour plus d'information sur les cibles supportées. L'énoncé "**Target**" peut apparaître plusieurs fois dans la description de la ressource, dans le cas où la ressource doit être téléchargée sur plusieurs cibles. Cet énoncé ne peut pas être spécifié si l'énoncé "AnyTarget" est utilisé.

```
Syntaxe:   TARGET '<nom_cible>'

Exemple:   BinaryFile 'MYFILE'
           Begin
             Target 'ISA86M'
             Target 'ISA68M'
             ...
           End
```

ANYTARGET

Rôle: Spécifie que la ressource doit être inclus dans tous les codes générés. Le générateur de code d'ISaGRAF peut produire différents codes lors de la même session de codage. Cet énoncé ne peut pas être spécifié dans une ressource où l'énoncé "**Target**" est utilisé.

Syntaxe: **ANYTARGET**

Exemple:
```
ULongData 'MYDATA'
Begin
  AnyTarget
  ...
End
```

FROM

Rôle: Spécifie le chemin du fichier source (sur le PC où l'atelier ISaGRAF est installé) d'une ressource **BinaryFile** ou **TextFile**. Les caractères utilisés pour séparer les composants du nom de chemin (unité, répertoire, préfixe, suffixe) doivent respecter les conventions du système MS-DOS.

Syntaxe: **FROM '**<chemin_source>**'**

Exemple:
```
BinaryFile 'MYFILE'
Begin
  ...
  From 'c:\user\config.dat'
  To '/dd/user/appl/config.dat'
End
```

TO

Rôle: Spécifie le chemin du fichier de destination (sur le système cible) d'une ressource **BinaryFile** ou **TextFile**. Les caractères utilisés pour séparer les composants du nom de chemin (unité, répertoire, préfixe, suffixe) doivent respecter les conventions du système hôte de la cible.

Syntaxe: **TO '**<chemin_destination>**'**

Exemple:
```
TextFile 'MYFILE'
Begin
  ...
  From 'c:\user\config.dat'
  To '/dd/user/appl/config.dat'
End
```

Exemple

Voici un exemple complet de fichier de définition de ressources:

```
(* resource definition file *)
```

```
ULongData 'DATA1'              (* list of values *)
Begin
   Target 'ISA86M'             (* for this target only *)
   1, 0, 16#1A2B3C4D, +1, -1   (* numerical values *)
End

VarList 'VLIST1'               (* list of variables *)
Begin
   Target 'ISA86M'             (* for this target only *)
   Valve1, StateX, Command, Alrm1 (* variable names *)
End

BinaryFile 'FILE1'             (* binary file resource *)
Begin
   AnyTarget                   (* dedicated to all targets *)
   From 'c:\user\updatef.bin' (* source file on PC *)
   To 'updatef.cfg'            (* target file on PLC *)
End

TextFile 'FILE2'               (* text file resource *)
Begin
   Target 'ISA68M'
   From 'c:\nw\nwbd.txt'       (* source file on PC *)
   To '/nw/dat/nwbd'           (* target file on PLC *)
End
```

Compilation des ressources

Si des ressources ont été définies et si des erreurs ont été détectées dans le fichier de définition des ressources, une boîte de dialogue apparaît à la fin de la génération du code ISaGRAF. Pressez le bouton "Exit" pour fermer la boîte.

Implémentation

Le nombre de ressources, la taille des séries de valeurs et des fichiers ne sont pas limités par ISaGRAF. Le contenu des ressources est stocké à la fin du code généré, précédé par un répertoire de ressources. Voici le format (utilisant les notations du langage C) du répertoire de ressources:

```
__RESOURCE:
{
  long nbres;            /* nombre de ressources définies */
  {
    char name[16];       /* nom de la ressource */
    long type;           /* type de la ressource */
    long size;           /* taille exacte des données */
    uint32 data;
    uint32 path_offset;  /* pointe vers une chaîne de
                            caractères */
  } /* nb d'enregistrements */
}
```

Voici les valeurs possibles pour le champ "type":

- 1 = fichier binaire
- 2 = fichier texte

- 3 = liste de valeurs (path_offset inutilisé dans ce cas)
- 4 = liste de variables (path_offset inutilisé dans ce cas)

Pour les fichiers textes, les caractères de fin de ligne sont convertis par le compilateur de ressources, selon les conventions du système cible. Tous les pointeurs sont des offsets sur 32 bits par rapport à l'adresse de début de la structure correspondante. Tous les noms de ressources et les noms de chemin sont terminés par un caractère nul. Les noms de chemin et le contenu des ressources sont stockés à la suite du répertoire des ressources.

A.14 Références croisées

L'atelier ISaGRAF inclut un éditeur de références croisées qui offre une vue complète sur l'utilisation des variables d'un projet. Le but des références croisées est de lister toutes les **variables** déclarées dans le dictionnaire du projet, et de **localiser**, dans les sources des programmes, les utilisations de chacune des variables. Les références croisées offrent une vue d'ensemble sur le traitement d'une variable. Elles permettent de localiser les **effets de bord**, et de réduire le temps de compréhension pendant la maintenance d'une application. Les références croisées fournissent également une vue globale, en lecture seulement, de tout le dictionnaire de données, de retrouver rapidement les variables déclarées et **non utilisées**, et de mesurer la complexité d'un projet.
La liste de gauche montre les objets déclarés du projet (programmes, variables, mots équivalents...), et les éléments de la librairie (fonctions et blocs fonctionnels) référencés dans le projet. La liste de droite montre les occurences dans les sources des programmes de l'objet sélectionné dans la liste de gauche.
La description d'une occurrence comprend le nom du programme, éventuellement le numéro de l'étape ou de la transition pour le SFC, ou de l'action ou du test pour le FC, plus le numéro de ligne dans le cas des langages littéraux ou les coordonnées (x,y) pour les diagrammes LD ou FBD. Dans le cas du Quick LD, la description est complétée par le numéro de l'échelle. Si la variable apparaît en sortie (sur une bobine), le numéro de l'échelle est suivie du caractère "*****".
Validez le mode "**Montrer les variables inutilisées**" dans le menu "**Options**" pour ajouter à la liste les variables qui ne sont utilisées nulle part dans les programmes.

Sélectionner un type d'objet

Un projet pouvant grouper un grand nombre d'objets déclarés, la boîte de sélection à gauche de la barre d'outils permet de sélectionner le type des objets listés dans la fenêtre. Ceci permet un accès filtré aux objets du projet.
Chaque fois que les références croisées sont re-calculées, la sélection est ramenée à "**Tous les objets**" pour présenter la liste complète.

Re-calculer les références croisées

La commande "**Fichier / Re-calculer**" permet à tout moment de mettre à jour les références croisées pour prendre en compte les modifications entrées dans les autres fenêtres de l'atelier ISaGRAF.

Exporter les références croisées

La commande "**Outils / Exporter**" écrit la liste complète des références croisées dans un fichier de texte ASCII. Ce fichier peut par la suite être repris par d'autres applications telles que le bloc-notes de Windows ou des traitements de texte.

Erreurs de déclaration

La commande "**Edition / Erreurs de déclaration**" liste dans une boîte de dialogue toutes les erreurs de syntaxe ou de cohérence détectées lors du chargement du dictionnaire.

Statistiques

La commande "**Outils / Statistiques**" affiche dans une boîte de dialogue le nombre de variables déclarées dans le projet, les sous-totaux étant triés par type et attribut. Une application particulière de cette commande est de connaître le nombre de variables d'E/S déclarées, afin de vous assurer que le code pourra être généré, si vous utilisez une version limitée de l'atelier ISaGRAF.

Chercher un objet dans la liste

La commande "**Edition / Chercher**" recherche un objet dans la liste affichée. Un objet ne peut pas être trouvé s'il n'est pas affiché. Il est conseillé de sélectionner l'affichage de **tous** les objets avant d'effectuer une recherche.

Ouvrir un programme

La liste de droite montre les occurences dans les programmes et le câblage des E/S de l'objet sélectionné. La commande "**Edition / Ouvrir programme**" permet d'accéder directement aux documents où l'objet sélectionné est référencé. Il est également possible de cliquer deux fois sur le nom d'une occurence dans la liste de droite pour ouvrir le document correspondant.

A.15 Mise au point

L'atelier ISaGRAF contient un debugger graphique et symbolique. La commande "**Tester**" du Gestionnaire de Programmes lance le debugger pour contrôler l'application téléchargée dans l'automate cible. Dans ce mode, le debugger dialogue avec le système cible via une liaison série ou Ethernet. La commande "**Simuler**" du Gestionnaire de Programmes lance simultanément le debugger et une fenêtre de simulation de l'automate. Ceci permet de commencer les tests de l'application quand la machine cible n'est pas encore disponible. La fenêtre du debugger regroupe les commandes permettant de contrôler l'application dans sa totalité.
Quand le debugger a établi le dialogue avec l'application testée, il ouvre automatiquement la fenêtre du **Gestionnaire de Programmes**, en mode "mise au point". Les commandes de cette fenêtre peuvent alors être utilisées pour ouvrir les autres fenêtres d'ISaGRAF (éditeurs textuels ou graphiques, dictionnaire, listes de variables, câblage des E/S...) Toutes les fenêtres ouvertes pendant le test sont ouvertes en mode "**mise au point**". Les commandes d'édition ne sont plus disponibles. Tous les composants affichés (étapes, contacts, variables...) sont montrés avec leur état ou valeur courant. Il suffit de cliquer deux fois sur un objet pour changer son état ou sa valeur.
Quand le debugger est utilisé en mode **simulation**, aucun dialogue n'est plus effectué avec l'automate cible ISaGRAF. Le debugger communique alors avec la fenêtre du simulateur. Puisque le système cible n'existe pas dans ce mode, les commandes de "**téléchargement**", "**arrêt**" ou "**activation**" ne sont plus valides dans les menus du debugger.

A.15.1 La fenêtre du debugger

La fenêtre du debugger regroupe toutes les informations concernant l'état de l'automate et de l'application. Elle dialogue avec les autres fenêtres d'ISaGRAF pour former un système interactif complet de test et de mise au point. Les erreurs détectées à l'exécution sont affichées dans la partie inférieure de la fenêtre du debugger. Utilisez les commandes du menu "**Options**" pour cacher, montrer ou effacer la liste des erreurs d'exécution.
Le tableau de bord du debugger (zone en dessous du menu) indique l'état précis de l'application testée, et des informations concernant le temps de cycle. Voici la liste des messages décrivant l'état de l'application:

Connexion:.....Le debugger est en train d'établir la communication avec l'automate cible.
Déconnecté:.....Le debugger ne peut pas dialoguer avec la cible. Vérifiez le câble de connexion et la valeur des paramètres de communication.
Pas d'application:....La connexion est ok, mais aucune application ISaGRAF n'est installée dans l'automate. Vous devez transférer l'application.
Application active:...La connexion est ok et une application active existe dans l'automate. Le debugger installe maintenant le dialogue avec cette application.

MARCHE: .....L'application est dans le mode "Temps Réel" (marche normale).
ARRET: .....L'application est arrêtée dans le mode "Cycle à Cycle".
Point d'arrêt: .....L'application est en mode "Cycle à Cycle", car un point d'arrêt a été rencontré.
Erreur fatale: .....L'application est interrompue car une erreur grave est survenue.

Voici les informations concernant le temps de cycle:

Autorisé:.....C'est le temps programmé pour le cycle.
Courant: .....C'est la dernière valeur exacte mesurée.
Maximum: .....C'est la durée utilisée par le cycle le plus long depuis le lancement de l'application.
Dépassements: .....C'est le nombre de cycles ayant provoqué un dépassement du temps programmé.

Tous les temps sont donnés en milli-secondes. Les composantes du temps de cycle ne sont pas affichées quand le debugger est utilisé en mode simulation.

A.15.2 Contrôle de l'application

Les menus "**Fichier**" et "**Contrôle**" groupent les commandes pour l'installation de l'application dans la cible ISaGRAF et le contrôle de l'application en temps réel.

Note: Certaines de ces commandes ne sont naturellement pas disponible quand le debugger est utilisé pour la simulation, puisque l'installation testée est automatiquement installée par l'atelier ISaGRAF.

Arrêter l'application

La commande "**Arrête l'application**" du menu "**Fichier**" stoppe l'exécution de l'application active dans l'automate cible ISaGRAF.

Démarrer l'application

La commande "**Active l'application**" lance l'exécution de l'application installée dans l'automate cible. Quand une application est téléchargée, elle est automatiquement lancée, et la commande "**Activer**" n'est pas nécessaire. La commande "**Activer**" est typiquement utilisée après une commande "**Arrêter**".

Note**:** l'application présente dans l'automate cible doit être arrêtée (inactive) avant le téléchargement d'une nouvelle application.

Télécharger le code de l'application

Les commandes "**Transférer**" du menu "**Fichier**" permettent de télécharger le code de l'application dans l'automate cible. Selectionnez le type code à télécharger, en accord avec les options spécifiées pour l'application et le type de processeur implanté dans votre automate.

Afficher le numéro de version

La commande "**Lecture du numéro de version**" du menu "**Fichiers**" est utilisée pour afficher l'identification complète de l'application sur l'atelier et de l'application cible. L'application sur l'atelier est celle actuellement ouverte dans l'atelier ISaGRAF. L'application cible est celle actuellement exécutée dans l'automate cible ISaGRAF. Les informations suivantes sont affichées:

VERSION: .....C'est le numéro de version du code de l'application. Ce numéro a été calculé par le générateur de code.
DATE: .....C'est la date et l'heure de la session de génération de code qui a produit cette application.
CRC: .....C'est un checksum calculé à partir du contenu de la table des symboles de l'application. Ce numéro a été calculé par le générateur de code. Sa valeur dépend du contenu du dictionnaire de variables de l'application.

Note: Cette commande est également disponible pendant la simulation. Pendant la mise au point en réel, cette commande est sans effet si l'automate cible est déconnecté.

Modification en ligne

La commande "**Mise à jour**" du menu "**Fichiers**" permet de réaliser une modification "en ligne" de l'application en cours d'exécution dans l'automate. Cette fonction est décrite en détaille dans les sections suivantes de ce chapître. Cette commande n'est pas disponible quand le debugger est utilisé en mode simulation.

Mode temps réel

La commande "**Contrôle / Temps réel**" n'est disponible que lorsque l'application est active dans l'automate. Elle force l'application en mode normal "Temps Réel": les cycles d'exécution sont cadencés en fonction du temps de cycle programmé.

Mode cycle à Cycle

La commande "**Contrôle / Cycle à cycle**" n'est disponible que lorsque l'application est active dans l'automate. Elle force l'application en mode normal "Cycle à cycle". Dans ce mode, les cycles sont exécutés un par un, à chaque commande "**Exécuter un cycle**" émise depuis le menu du debugger.

Exécuter un cycle

Quand la cible est en mode Cycle à Cycle, la commande "**Contrôle / Exécuter un cycle**" lance l'exécution d'un seul cycle automate.

Le temps de cycle

La commande "**Changer le temps de cycle**" du menu "**Contrôle**" permet de modifier la valeur programmée pour le temps de cycle. C'est la valeur intitulée "**Autorisé**" dans le tableau de bord du debugger. Il est préférable de positionner le mode "**Cycle à cycle**" avant de changer le temps de cycle. Le temps de cycle est saisi comme un nombre entier de milli-secondes.

Retirer tous les points d'arrêt

La commande "**Effacer les points d'arrêt**" du menu "**Contrôle**" détruit tous les points d'arrêt installés (rencontrés ou encore actifs) dans toute l'application. Les

points d'arrêt installés ne sont pas automatiquement détruits quand vous fermez la fenêtre du debugger.

Déverrouiller les variables d'E/S

La commande "**Déverrouiller**" du menu "**Contrôle**" déverrouille toutes les variables d'entrée ou de sortie de l'application qui ont été verrouillées. Quand une entrée / sortie est verrouillée, aucune opération n'est plus effectuée sur le dispositif d'E/S correspondant. Les variables correspondant aux E/S verrouillées peuvent être forcées par le debugger. Les E/S verrouillées ne sont pas automatiquement déverrouillées quand vous fermez la fenêtre de debugger.

A.15.3 Options

Le menu "**Options**" regroupe les différentes options permettant de contrôler les affichages dans la fenêtre du debugger.

Paramètres de communication

Les paramètres pour la communication peuvent être réglés quand le debugger est ouvert. Seuls les paramètres temporels (**time-outs)** peuvent être réglés ici. Les autres paramètres (débit, parité...) doivent être réglés depuis la fenêtre du Gestionnaire de Programmes quand la fenêtre du debugger est fermée.
Le " **time-out de communication**" est le temps laissé au système cible pour commencer à répondre à une requête du debugger. La "**période de rafraîchissement**" est la période des requêtes de "**lecture**" émises par le debugger pour rafraîchir le contenu des fenêtres ouvertes.
Tous les temps sont affichés et saisis en **milli-secondes**. Les paramètres de la communication ne peuvent pas être réglés quand le debugger est utilisé avec le simulateur.

Options d'affichage

L'option "**afficher le temps de cycle**" permet de valider ou d'inhiber l'affichage rafraîchi des composantes du **temps de cycle** dans le tableau de bord de l'automate. Inhiber l'affichage du temps de cycle réduit la charge de communication et d'affichage du debugger.
Quand l'option "**afficher les erreurs**" est choisie, les erreurs détectées à l'exécution sont affichées au fil de l'eau dans la partie inférieure de la fenêtre du debugger. Quand cette option est inhibée, la partie inférieure de la fenêtre est cachée. Inhiber l'affichage des erreurs réduit la charge de communication et d'affichage du debugger.
La commande "**Options / Réduire la fenêtre**" réduit la fenêtre du debugger et la montre comme un petit tableau de bord toujours visible situé au sommet et à droite de l'écran. Seuls l'état de l'application et les boutons graphiques pour les commandes les plus courantes sont visibles dans ce mode.

A.15.4 Commandes de forçage

Le debugger d'ISaGRAF propose de nombreuses commandes pour changer la **valeur** ou l'**état** des composants de l'application. La désignation de l'élément à

modifier se fait toujours en **cliquant deux fois** sur son nom ou son symbole dans les fenêtres de l'atelier ISaGRAF, quand la fenêtre du debugger est ouverte.

Variables

Pour modifier la valeur d'une variable, cliquez deux fois sur son nom dans une des fenêtres suivantes:

- Dictionnaire
- listes de variables
- programmes LD ou FBD
- câblage des E/S

Les commandes suivantes sont proposées dans une boîte de dialogue:

- Forçage de la variable à une nouvelle valeur
- **Verrouillage** de la variable (pour E/S seulement)
- **Déverrouillage** de la variable (pour E/S verrouillées seulement)
- **Lancement** ou **Arrêt** d'une variable temporisation

Les libellés utilisés pour représenter les valeurs **FALSE** et **TRUE** sont ceux définis à la déclaration de la variable dans le dictionnaire. La valeur analogique spécifiée pour une commande de forçage d'une variable analogique doit être entrée en format réel ou entier, en accord avec la déclaration de la variable. La chaine spécifiée lors d'une commande de forçage d'une variable message ne doit pas excéder la longueur maximum spécifiée lors de la déclaration de la variable.

Objets SFCs

Pour réaliser une opération de contrôle d'un **programme SFC** pendant la mise au point d'une application, utilisez les commandes du menu "**Fichiers**" dans la fenêtre du gestionnaire de programmes. Les commandes concernent toujours le programme sélectionné dans la fenêtre. Les commandes suivantes sont disponibles:

Lancer: .....Active le programme en plaçant un jeton dans chacune de ses étapes initiales.
Tuer: .....Tue le programme en détruisant tous les jetons existant dans ses étapes.
Figer: .....Suspend l'exécution du programme.
Relancer: .....Relance un programme suspendu (figé).

Pour les programmes SFC fils, ces commandes correspondent aux énoncés "**GSTART**", "**GKILL**", "**GFREEZE**" et "**GRST**" du langage.
Pour contrôler une **étape SFC** pendant la mise au point d'une application, cliquez deux fois sur son symbole dans la fenêtre d'édition SFC. Les commandes suivantes sont proposées dans une boîte de dialogue:

- Poser un point d'arrêt sur l'**activation** de l'étape
- Poser un point d'arrêt sur la **désactivation** de l'étape
- **Enlever** le point d'arrêt posé sur l'étape

Note: Les points d'arrêt en activation et désactivation ne peuvent pas coexister sur la même étape.

Pour contrôler une **transition SFC** pendant la mise au point d'une application, cliquez deux fois sur son symbole dans la fenêtre d'édition SFC. Les commandes suivantes sont proposées dans une boîte de dialogue:

- Poser un **point d'arrêt** sur le franchissement de la transition
- **Enlever** le point d'arrêt posé sur la transition
- **Franchir** manuellement la transition (déplacement ou création de jetons)

Lors d'un **franchissement conditionnel**, un jeton est créé dans chacune des étapes suivant la transition. Les jetons existant dans les étapes amont sont enlevés. Lors d'un **franchissement inconditionnel**, un jeton est créé dans chacune des étapes suivant la transition. Les jetons existant dans les étapes amont ne sont pas déplacés ni détruits. Ces commandes ne tiennent pas compte de la valeur de la réceptivité de la transition.

A.15.5 Modification en ligne

La fonction de "Modification en ligne" permet à l'utilisateur de modifier une application pendant qu'elle est exécutée dans l'automate cible. Ceci est parfois nécessaire pour certaines applications ne pouvant absolument pas être arrêtées, par exemple dans le cas des procédés chimiques. Bien sûr, cette fonction doit être utilisée avec **beaucoup de précautions** car il est impossible pour ISaGRAF de détecter tout type de conflit ou de non sens dans les modifications apportées à l'application.

Séquences de code

ISaGRAF offrant naturellement de nombreuses possibilités de contrôle des variables, programmes et Entrées/Sorties depuis le debugger, la fonction de "Modification en ligne" décrite ici ne concerne que la modification des séquences de code. Une séquence de code est un ensemble complet d'instructions ST, IL, LD ou FBD exécutées en ligne. Pour un programme de "début de cycle" ou de "fin de cycle", une séquence de code est le contenu complet du programme. Pour un programme SFC, une séquence de code est la programmation du niveau 2 d'une étape ou d'une transition. La "Modification en ligne" consiste à remplacer une ou plusieurs séquences de code, sans arrêter le cadencement des cycles d'exécution. Comme le contrôle des activités d'étapes SFC est très critique, **il n'est pas autorisé de modifier la structure des programmes SFC, d'ajouter, supprimer ou de renuméroter des étapes ou transitions d'un programme SFC**.

Variables

La base de données des variables est un point très sensible de l'application. Cette base de données étant accédée en temps réel par d'autres processus (sur des automates multitâches) et la modification des variables étant possible depuis le debugger, **il n'est pas autorisé d'ajouter, renommer ou supprimer des variables**. Bien sûr, il est possible de modifier l'utilisation de ces variables dans l'application. Il est également possible de réserver des variables internes ou d'E/S "inutilisées" pour assurer la faisabilité des modifications futures.
Il y a différents styles de variables dans la base de données dynamique d'ISaGRAF sur la cible. Les limitations énoncées ci-avant agissent sur tous ces styles de variables:

- *Variables déclarées*
Ce sont les variables déclarées dans le dictionnaire de l'application. Elles ne peuvent pas être changées ni renommées dans le cas d'une modification en ligne. Il est recommandé de déclarer quelques variables de plus et de les initialiser dans les programmes. Ces variables pourront servir dans le futur à implémenter de nouvelles fonctionalités acceptées lors d'une modification en ligne.

- *Instances de blocs fonctionnels*
A chaque instance de bloc fonctionnel en "C" ou en langage IEC correspond un paquet de données rangé dans la base de données dynamique de la cible ISaGRAF. Quand des instances de blocs fonctionnels sont ajoutées ou supprimées, la modification en ligne n'est plus possible. Il est donc préférable, dans des applications nécessitant la modification en ligne, de travailler en ST ou en IL avec des instances de blocs fonctionnels déclarées dans le dictionnaire, plutôt que d'ajouter des blocs dans des diagrammes Quick LD ou FBD (qui correspondent à des instances créées automatiquement). Aussi, tout changement dans la définition des blocs fonctionnels de la librairie ISaGRAF engendrera une modification en ligne refusée.

- *Etapes SFC*
A chaque étape SFC correspond une structure de données (validité, temps d'activation...) stockée dans la base de données deynamique de la cible. L'ajout ou la suppression d'étapes dans les schémas SFC engendrera une modification en ligne refusée.

- *Variables cachées allouées par les compilateurs*
Les compilateurs d'ISaGRAF créent des variables temporaires "cachées" afin de résoudre les expressions complexes énoncées dans les langages. Dans certains cas, le changement d'une expression peut engendrer une modification du nombre de variables temporaires requises, et conduire à une modification en ligne impossible. Pour éviter cette situation, vous pouvez ajouter les entrées suivantes dans le fichier de paramètres ISA.INI, pour forcer l'allocation d'un minimum de variables temporaires pour chaque type de donnée et chaque programme du projet. Voici un exemple de paramétrage:

```
[DEBUG]
MNTVboo=8        ; pour les booléens
MNTVana=4        ; pour les entiers et les réels
MNTVtmr=4        ; pour les temporisations
MNTVmsg=2        ; pour les messages
```

Quand ce paramétrage est présent dans le fichier ISA.INI, le compilateur produit une erreur si le nombre de variables temporaires à allouer excède le nombre minimum explicité dans les paramètres MNTV.

Entrées et sorties

Le système d'E/S d'ISaGRAF étant très ouvert, la plupart des modifications devront être prévues par le concepteur OEM, prenant en compte les fonctionnalités spécifiques du matériel. ISaGRAF **ne permet pas d'ajouter, connecter ou enlever une carte d'E/S, ni ne modifier la description d'une carte**, lors d'une

modification en ligne. Les opérations telles que la modification des paramètres d'une carte, le verrouillage des voies d'E/S, peuvent également être assurées de façon standard par la mise en oeuvre de la fonction "**OPERATE**".

Opérations sur la cible

La modification d'une application en cours d'exécution sur la cible consiste en une série d'opérations élémentaires:

- modification du code source de l'application sur l'atelier
- génération du nouveau code de l'application
- téléchargement du nouveau code généré avec la commande "**mise à jour**"
- passage d'une application à l'autre, en 1 ou 2 cycles d'exécution, avec la commande "**Effectuer la mise à jour**".

Cette procédure garantit que l'application exécutée sur l'automate est toujours complète et cohérente, et permet à l'utilisateur de contrôler chacune des opérations de façon fiable et sécuritive. Elle permet également de réitérer la modification plusieurs fois si nécessaire. A l'exception des conséquences possibles sur le procédé, la "Modification en ligne" est presque aussi sécuritive que l'enchaînement des commandes "**arrêter**, **activer** ou **télécharger**". Les seules différences sont que les états courants des données ne sont pas perdus, et que le temps de passage d'une application à l'autre est très court (généralement une durée de 1 ou 2 cycles). Pendant le passage d'une application à l'autre, aucune variable n'est modifiée, et **toutes les variables internes d'entrée ou de sortie gardent la même valeur** avant et après la modification de l'application. Pendant le passage d'une application à l'autre, aucune action n'est exécutée, et les **activités d'étapes SFC n'évoluent pas**.

Besoins en mémoire

Pour permettre la fonction de "Modification en ligne", l'automate cible doit disposer d'un espace mémoire disponible suffisant pour le stockage de la version modifiée de l'application. Les deux versions du code de l'application sont en effet stockés ensemble dans l'automate pendant l'opération de mise à jour.

Limitations

Comme décrit ci-avant, seules les modifications des séquences de code sont autorisées par la "Modification en ligne". La définition des variables, les paramètres de l'application et le câblage des E/S ne peuvent pas être modifiés. Lors du transfert de la nouvelle version de l'application, ISaGRAF effectue une comparaison entre le code transféré et celui en cours d'exécution, afin de détecter toute modification incohérente. Si le passage d'une application à l'autre est dangereux, une erreur de téléchargement est émise. Actuellement, un des contrôles effectués par ISaGRAF est de comparer les checksums des tables de variables, de façon à détecter toute différence dans la définition des variables, des programmes, ou des éléments graphiques du langage SFC. Si une étape est active lors de la modification, ses actions non mémorisées (type N) sont perdues. Les actions d'activation (type P) décrites dans le nouveau code sont exécutées. Les actions exécutées lors de la désactivation de l'étape sont celles décrites dans le nouveau code de l'application. Si une transition est valide lors de la modification, son équation de réceptivité est mise à jour. Si un changement dans une structure du graphe SFC n'a pu être détecté par ISaGRAF (insertion, suppression ou

déplacement d'éléments), La fonction de "Modification en ligne" peut avoir des effets imprévisibles. Le code de la nouvelle application téléchargée n'est pas sauvegardé dans l'automate. La sauvegarde reste l'image de l'application initialement installée par une commande de téléchargement classique.

Operations

Les opérations suivantes doivent être respectées pendant la mise à jour du code d'une application en cours d'exécution dans l'automate:

- Avant toute modification de l'application, il est vivement recommandé de copier le projet ISaGRAF sous un autre nom. La modification peut indifféremment être effectuée depuis l'une ou l'autre des copies.
- Avant d'éditer les programmes, l'utilisateur doit valider l'option "**mettre à jour l'agenda**" des outils d'édition, pour faciliter la maintenance de l'application.
- Quand les programmes ont été modifiés (sans changement des graphes SFC ni de la hiérarchie des programmes), le code de la nouvelle application doit être généré.
- Depuis le debugger, dans le projet initial, l'utilisateur doit réaliser la connexion avec l'automate et exécuter toutes les opérations nécessaires pour faciliter et sécuriser la mise à jour de l'application.
- Depuis le debugger, dans le nouveau projet, l'utilisateur doit réaliser la connexion avec l'automate. Si le nom de l'application est changé, les données de l'application ne sont pas rafraîchies par le debugger. L'utilisateur doit alors lancer la commande "**Mise à jour**" du menu "**Fichiers**" de la fenêtre du debugger.
- Le transfert de la nouvelle application commence, après la sélection du mode "**différé**". Le transfert peut légèrement ralentir le cycle de l'automate.
- Quand le transfert est terminé, l'utilisateur doit lancer la commande "**Effectuer la mise à jour**" pour effectuer le passage d'une application à l'autre au moment adéquat. Le passage d'une application à l'autre peut durer 1 ou 2 cycles automate.
- Quand le passage a été correctement effectué, les programmes de la nouvelle application sont affichés dans l'écran du debugger. Si ce n'est pas le cas, cela signifie que la passage à la nouvelle application a été refusé et que l'ancienne version est encore active.

A.15.6 Echanges DDE

Le debugger d'ISaGRAF inclut un serveur DDE (Dynamic Data Exchange). Des boucles de notifications (advise loop) peuvent être installées entre le debugger d'ISaGRAF et d'autres applications, pour afficher dynamiquement la valeur de variables ISaGRAF dans ces applications.
Seules les requêtes de type "advise" et "poke" sont supportées par le serveur DDE du debugger ISaGRAF. Vous ne pouvez utiliser les requêtes "request" que sur les variables déjà affectées à une boucle de notification ("advise"). Les autres services DDE tels que "execute" ne sont pas disponibles. Lorsqu'une boucle de notification est établie, la variable est automatiquement rafraîchie dans l'application cliente, à chaque fois que sa valeur évolue. Les variables de tous types peuvent ainsi être espionnées. Voici les éléments d'identification à fournir pour établir une conversation:

Nom de service: ..... "ISaGRAF"
Topic Nom de document: ..... Nom de l'application ISaGRAF
Nom d'item: ..... Nom de la variable

Si la variable est locale à un programme, son nom doit être suivi par le nom de son programme père, écrit entre parenthèses, selon la syntaxe suivante:

nom_variable(nom_programme)

Le serveur DDE du debugger ISaGRAF est dédié aux variables de l'application actuellement contrôlée par le debugger. Au plus **256** variables peuvent être espionnées par le serveur DDE d' ISaGRAF. Le serveur DDE d'ISaGRAF peut être utilisé aussi bien quand le debugger est utilisé en mode "connecté" que pendant la simulation. La période de rafraîchissement des informations est celle établie pour la communication entre le debugger et le système cible ISaGRAF ou le simulateur.

A.16 Espionner des listes de variables

La commande "**Espion / Listes de variables**" de la fenêtre du debugger permet à l'utilisateur de créer des listes de variables qui seront rafraîchies avec leur valeur courante. Les listes sont construites pendant la mise au point, elles peuvent être stockées sur le disque et ré-ouvertes pendant les sessions de mise au point suivantes. Une liste peut contenir jusqu'à **32** variables. Des variables de types différents peuvent être mélangées dans une même liste. Les variables locales et globales peuvent être insérées dans les listes. Une liste de variables est dédiée à un projet. Les listes de variables sont très utiles pour le test fonctionnel d'une application. Elles permettent la visualisation et la surveillance d'une partie limitée du procédé contrôlé, sans regard sur le code source des programmes correspondants. Les listes de variables sont également très utiles pour la mise au point des programmes textuels écrits en ST ou IL. L'utilisateur peut facilement grouper dans une liste la série des variables manipulées dans un programme, pour contrôler son exécution.

Pour chaque variable, ISaGRAF affiche son nom, sa valeur courante et son commentaire. Déplacez les séparation avec la souris dans la ligne de titre pour redimensionner les colonnes.

Enregistrer les listes sur le disque

Les commandes du menu "**Fichiers**" permettent de créer, ouvrir et sauvegarder les listes de variables. Le nombre de listes pour un projet n'est pas limité par ISaGRAF. Lors de la nomenclature des listes de variables, les règles suivantes doivent être respectées:

- la longueur du nom ne doit pas excéder **8** caractères
- le premier caractère doit être une **lettre**
- les suivants doivent être des **lettres**, des **chiffres** ou le souligné
- les majuscules et les minuscules ne sont pas différenciées

L'éditeur de liste ne peut afficher qu'une seule liste dans sa fenêtre. Toutefois, il est possible de lancer simultanément plusieurs fenêtres d'édition contenant des listes différentes.

Insérer des variables dans la liste

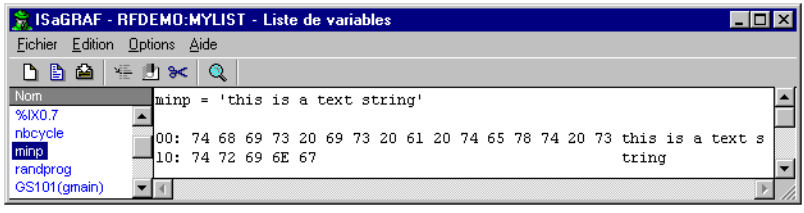
La commande "**Edition / Insérer**" insère une nouvelle variable dans la liste. Le nom de la variable est sélectionné parmi les noms d'objets déclarés dans le dictionnaire du projet. Son identificateur n'a donc pas à être saisi manuellement. La variable est insérée juste avant la variable actuellement sélectionnée dans la liste. La liste ne peut pas contenir plus de **32** variables. La même variable ne peut pas apparaître plusieurs fois dans la même liste.

Changer la variable sélectionnée

la commande "**Edition / Modifier**" remplace la variable sélectionnée par une autre variable. Vous pouvez aussi utiliser la commande "**Couper**" pour retirer la variable sélectionnée de la liste.

Affichage "Dump"

A tout moment, vous pouvez basculez l'affichage entre le mode normal en colonnes et le mode "Dump". Pressez le bouton "zoom" dans la barre d'outils ou utilisez la commande "**Options / Dump**" pour changer le mode d'affichage.
En mode "Dump", une seule variable est rafraichie. Sa valeur en numérique ou en texte est affichée au sommet de la fenêtre. La valeur est également affichée en binaire (mode "dump") avec la valeur de chaque octet en hexadécimal. Ce mode vous permet d'espionner précisément chaque octet de la valeur d'une variable.



Le mode "dump" est particulièrement utile pour la visualisation des chaines de caractères longues ou comprenant des caractères non imprimables.

A.17 Mise au point des programmes ST et IL

Pendant la simulation ou le test en ligne des programmes ST et IL program, aucune modification ne peut être saisie dans le texte.

IL Pour les programmes IL, les instructions sont formattées en liste. La valeur courante de la variable utilisée dans une instruction est affichée sur la même ligne. Cliquez deux fois sur une instruction pour forcer la variable correspondante.

ST Pour les programmes ST, une liste de variables est embarquée dans la fenêtre d'édition du programme. Vous pouvez redimensionner la liste en déplaçant la ligne de séparation avec la souris.

Pour chaque variable de la liste, ISaGRAF affiche son nom, sa valeur courante et son commentaire. Déplacez les séparations avec la souris dans la ligne de titre pour redimensionner les colonnes.

Enregistrer la liste sur le disque

La commande "**Fichier / Enregistrer la liste**" sauve la liste de variables sur le disque, sous le même nom que le programme ST. Cette liste sera automatiquement ouverte à chaque fois que le programme ST sera ouvert en mode test. Cette liste peut également être ouverte et modifiée avec l'éditeur de listes de variables, en utilisant la commande "**Espion / Listes de variables**" dans la fenêtre du debugger.

Insérer des variables dans la liste

La commande "**Edition / Insérer**" insère une nouvelle variable dans la liste. Le nom de la variable est sélectionné parmi les noms d'objets déclarés dans le dictionnaire du projet. Son identificateur n'a donc pas à être saisi manuellement. La variable est insérée juste avant la variable actuellement sélectionnée dans la liste. La liste ne peut pas contenir plus de **32** variables. La même variable ne peut pas apparaître plusieurs fois dans la même liste.
Quand un nom de variable est marqué dans la fenêtre ST, pressez ce bouton dans la barre d'outils ou utilisez la commande "**Edition / Espionner la selection**" pour directement ajouter la variable marquée à la liste.

Changer la variable sélectionnée

la commande "**Edition / Modifier**" remplace la variable sélectionnée par une autre variable. Vous pouvez aussi utiliser la commande "**Couper**" pour retirer la variable sélectionnée de la liste.

A.18 Mise au point avec SpotLight

SpotLight est un outil intégré à ISaGRAF qui permet la construction et l'animation d'images graphiques et de listes de variables pendant la mise au point ou la simulation des applications. Les éléments graphiques correspondent à des variables de l'application ISaGRAF. Le document graphique est défini et animé pendant la mise au point.
Pour forcer la valeur d'une variable, cliquez deux fois sur l'objet graphique correspondant dans l'image graphique ou l'affichage en liste, ou bien frappez la touche ENTRÉE quand l'élément est sélectionné.
Vous pouvez aussi verrouiller le document (interdire toute modification) à l'aide de la commande "**Fichier / Verrouiller**". Quand un document est verrouillé, il est toujours possible de forcer l'état d'une variable en cliquant deux fois sur un symbole.

A.18.1 Construire l'image graphique

Un document SpotLight est fait d'images de fond de plan (bitmaps ou metafiles), et de plusieurs objets graphiques animés pendant la mise au point. Pour consruire le graphique, les opérations suivantes doivent être réalisées: insérer des images de fond de plan, insérer des objets graphiques, associer les objets graphiques aux variables de l'application ISaGRAF.

Images de fond de plan

Les images de fond de plan sont des fichiers "bitmap" (.BMP) ou “metafile” (.WMF). Le nombre d'images insérées dans le graphique n'est pas limité. Les images peuvent être déplacées dans le document ou redimensionnées. Elles n'apparaissent pas dans l'affichage en liste. Les images sont construites à l'aide d'autres outils (par exemple Paint pour les bitmaps). SpotLight n'inclut pas d'outil de dessin. Utilisez la commande "**Options / Couleur de fond**" pour choisir la couleur à utiliser pour les parties du graphiques non couvertes par une image.

Note: Les bitmaps consomment une grande quantité de mémoire. Il est recommandé de correctement ajuster la taille des bitmaps insérés afin d'optimiser l'utilisation de la mémoire.

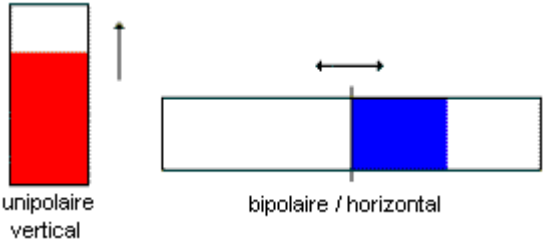
Textes

Un objet de type “affichage littéral” est un texte écrit dans un rectangle. Le texte affiché est la valeur de la variable associée. Un item de ce style peut donc être associé à une variable de type message.
Le rectangle où le texte est affiché peut être transparent ou rempli avec une couleur. La fonte de caractères utilisée est automatiquement ajustée pour que le texte soit entièrement dans la largeur du rectangle.

Histogrammes unipolaires et bipolaires

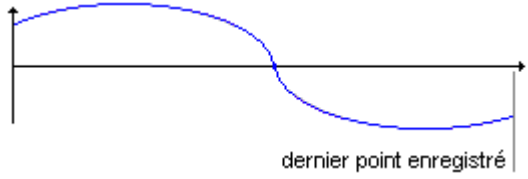
Un histogramme est un rectangle dont la partie colorée représente la valeur de la variable analogique qui lui est associée. Optionnellement, le reste du rectangle peut

être rempli avec une autre couleur. Un histogramme peut être horizontal ou vertical. Un histogramme unipolaire peut varier dans n'importe quelle direction (haut, bas, droite, gauche). Un histogramme bipolaire peut grandir dans deux directions opposée, pour représenter une valeur positive ou négative de la variable. Dans le cas d'un histogramme bipolaire, la valeur maximum affichable est la même pour les valeurs positives et les valeurs négatives.



Courbes

Il est possible d'insérer une courbe dans un document. Une courbe montre l'historique des valeurs de la variable associée. Bien qu'il ne s'agisse pas d'une mesure précise, une courbe peut donner de bonnes informations concernant la synchronisation des évènements.
Une courbe mémorise les 200 dernières valeurs de la variable. Le nombre de points stockés n'est pas changé quand la courbe est redimensionnée.

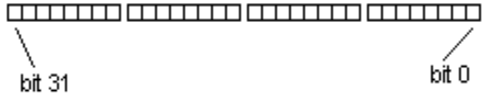


Icones booléen

Un “icône booléen” est utilisé pour afficher une valeur binaire. Un fichier icône (.ICO) est défini pour la valeur FALSE ou 0. Un autre icône est défini pour toutes les autres valeurs (TRUE ou différentes de 0). SpotLight n'inclut pas d'éditeur d'icône, et les fichiers icones doivent être préparés avec un autre outil.

Champs de bits

Un “champ de bits” montre sous forme graphique les 32 bits d'une valeur entière. Le bit le moins significatif est montré à droite. Il est déconseillé d'utiliser les champs de bits pour les autres types de variables, puisque l'information affichée (sous forme binaire) peut engendrer des confusions.



Sélectionner, déplacer et redimensionner des éléments

La sélection des objets est nécessaire pour la plupart des commandes d'édition. SpotLight supporte la sélection multiple dans la page graphique. Pour sélectionner des objets, le bouton "**sélection**" (flèche) doit être enfoncé dans la barre d'outils. Pour sélectionner un seul objet, cliquez simplement sur son symbole. Pour sélectionner plusieurs objets, faites glisser la souris pour tracer un rectangle de sélection. Tous les objets graphiques en intersection avec ce rectangle sont sélectionnés. Un objet sélectionné a des marques noires montrées sur le cadre autour de son dessin.
Une nouvelle sélection annulle la précédente. Pour annuler la sélection, cliquez simplement sur un endroit non utilisé du graphique, en dehors du rectange qui border les éléments sélectionnés.
Pour déplacer des objets, il faut les sélectionner. Puis glissez la sélection avec la souris vers le nouvel emplacement choisi.
Pour redimensionner un objet, il faut le sélectionner. Puis glissez avec la souris une des marques autour du cadre de sélection pour retailler le rectangle de l'objet. Les images de fond peuvent également être redimensionnées. Dans ce cas, l'image est compressée ou étendue pour adopter la nouvelle taille.

Grouper des éléments / dissocier des groupes

Il est possible de grouper des éléments pour les traiter comme un objet unique. Pour créer un groupe, sélectionnez les éléments à grouper et lancez la commande "**Edition / Grouper**". La commande "**Edition / Dissocier**" restitue individuellement tous les éléments d'un groupe.
Un groupe peut contenir des images de fond de plan. Un groupe peut également contenir d'autres groupes.
Quand des éléments sont groupés, leur style ne peut plus être changé. Les éléments du groupe sont toujours rafraichis, mais ne peuvent pas être utilisés (par un double click) pour forcer une variable.
Un groupe apparaît comme une seule ligne dans l'affichage en liste.

A.18.2 L'affichage en liste

A tout moment, vous pouvez basculer entre l'affichage graphique et l'affichage en liste en pressant ce bouton, ou à l'aide de la commande "**Options / Liste/Graphique**".
Dans l'affichage en liste, les éléments sont montrés un par un dans une liste. Chaque élément est montré avec son symbole graphique. Les images de fond de plan n'apparaissent pas dans la liste. La sélection dans la liste est possible pour la définition du style et le forçage des variables. La sélection multiple dans la liste n'est pas disponible.

Utilisez ces boutons ou les commandes “**Edition / Déplacer**” pour déplacer l'item sélectionner vers le haut ou vers le bas et ainsi réorganiser la liste.

A.18.3 Définir le style d'un élément

Le style et les propriétés d'un objet graphique peuvent être changés à l'aide de la commande “**Edition / Changer le style**” quand l'objet est sélectionné. La boîte de “Style” est également ouverte quand un nouvel élément est inséré. Elle groupe les informations suivantes:

Style et propriétés graphiques:
Le style d'affichage (littéral, histogramme, courbe …) d'un élément peut être changé dynamiquement. Quand des couleurs d'arrière et d'avant plan sont utilisées, elles peuvent être personnalisées. Quand le style est “icône booléen”, le nom des fichiers.ICO utilisés doit être spécifié. Utilisez le bouton “**…**” pour parcourir le disque et sélectionner un fichier.

Echelle:
C'est la valeur maximum affichable pour la valeur de la variable, dans le cas des histogrammes et des courbes. Pour les histogrammes bipolaires et les courbes, c'est la valeur absolue utilisée pour les deux axes positif et négatif.

Nom de la variable:
Quand le champ "**Nom**" est actif, pressez le bouton "**…**" pour sélectionner le nom d'une variable déclarée dans le dictionnaire de l'application.

Titre:
Un titre peut être associé à chaque élément. Vous pouvez choisir l'emplacement du titre dans le graphique (sur un des côtés du symbole) et son contenu. Le titre est une combinaison entre le nom de la variable est sa valeur littérale.

Variable de commande:
Si cette option est validée, il est possible de forcer la variable en cliquant deux fois sur le symbole de la variable. Cette fonctionnalité est inhibée si cette option n'est pas validée.

A.18.4 Commandes du menu "Fichier"

Le menu "**Fichier**" groupe les commandes de gestion des documents

La commande "**Nouveau**" démarre l'édition d'un nouveau document. Le nombre de documents définis pour un projet n'est pas limité par ISaGRAF. Avant de commencer un nouveau document, le document édité est fermé. SpotLight ne peut pas montrer plusieurs graphiques en même temps. Toutefois, SpotLight peut être lancé plusieurs fois pour des documents différents.

La commande "**Ouvrir**" charge un document stocké sur disque. Lors de la sélection du document à charger, vous pouvez utiliser le bouton "**Supprimer**" pour détruire le

fichier sélectionné dans la liste. Les fichiers icones et bitmaps associés aux documents ne sont pas détruits.

La commande "**Enregistre**" stocke le document édité sur le disque. Dans le cas d'un nouveau document, vous devez lui choisir un nom, en accord avec les conventions suivantes:

- Le nom ne peut pas excéder **8** caractères
- Le premier caractère doit être une **lettre**
- Les caractères suivants doivent être des **lettres**, **chiffres** ou le caractère **souligné**
- Les majuscules et minuscules ne sont pas différenciées

La commande "**Enregistrer sous**" permet d'enregistrer le document en cours d'édition sous un autre nom.

A.18.5 Note pour les utilisateurs d'ISaGRAF V3.2

Spotlight peut lire des graphiques et les listes de courbes créés avec les autres outils d' ISaGRAF V3.0 ou V3.2. Ces fichiers apparaissent dans la boîte "**Ouvrir**", avec la description de leur origine. Les fichiers peuvent ensuite être librement modifiés ou complétés avec SpotLight.
A l'ouverture d'un graphique d' ISaGRAF V3.2, le document est automatiquement marqué "Verrouiller". Retirer le verrouillage depuis le menu "**Fichier**" si vous voulez modifier le graphique.
Quand un graphique ou une liste de courbes d'ISaGRAF 3.2 est ouvert avec SpotLight, ISaGRAF vous propose toujours de les enregistrer au format SpotLight. La boîte "**Enregistrer sous**" est systématiquement ouverte quand le document est fermé.

A.19 Télélecture

ISaGRAF supporte la télélecture (télé-importation) d'une application chargée dans la cible. La procédure de télélecture communique avec la cible pour récupérer les sources compressées embarquées (EZS) et restituer le projet dans l'environnement de l'atelier.
Le projet implanté dans la cible peut être télé-importé si le système cible est ISaGRAF version 3.23, et si les sources compressées ont été embarquées avec le code de l'application. L'embarquement des sources pour la télélecture est une fonctionnalité optionnelle.

A.19.1 Récupérer un projet

La boîte de dialogue de "**Télélecture**" est lancée depuis le menu "**Fichier**" du Gestionnaire de projets d'ISaGRAF. La télélecture ne se réfère jamais à un projet existant. Le projet sélectionné dans la liste n'a aucune relation avec la procédure de télélecture. Pour réaliser la télélecture, vous devez:
1- vous assurer que la cible est correctement connectée
2- régler les paramètres de communicationen accord avec la liaison physique
3- presser le bouton "**Charger**"

La lecture et la décompression des sources embarquées peut prendre plusieurs secondes. Des messages dans la boîte de dialogue vous informent de la fin de la télélecture ou des éventuels cas d'erreur.
Le nom utilisé pour identifier le projet lu est celui embarqué avec l'application dans la cible. Si ce nom est déjà utilisé pour un projet, vous devez choisir un autre nom. Vous ne pouvez pas abandonner la télélecture une fois le projet restitué. Le projet télé-importé est maintenant prêt à être édité.

Erreurs

Voici les différentes erreurs pouvant survenir pendant la télélecture. Ces erreurs sont affichées dans la boîte de dialogue de télélecture.
- la communication avec la cible ne peut pas être établie
- la cible connectée est antérieure à ISaGRAF version 3.23
- il n'y a pas d'application installée dans la cible
- les sources de l'application n'ont pas été embarquées

A.19.2 Paramétrage de la liaison

Pressez le bouton "**Configurer**" pour définir les paramètres de communication entre l'atelier et la cible. Vous devez vous assurer que la communication est correctement configurée avant de démarrer la télélecture.

A.19.3 Préparation d'un projet pour la télélecture

Vous devez informer le générateur de code d'ISaGRAF que les sources compressées doivent être incluses au code généré si vous désirez bénéficier de la télélecture plus tard. Pour ceci, pressez le bouton "**Télélecture**" dans la boîte des "**Options de compilation**". Une autre boîte de dialogue vous permet de valider ou inhiber l'inclusion des sources compressées. Dans ce cas, seules les sources essentielles sont incluses. D'autres options vous permettent d'embarquer également quelques fichiers optionnels.

Note: Les librairies ne sont pas embarquées avec le code source de l'application. Ceci inclut les fonctions et blocs fonctionnels de la librairie, ainsi que la définition des cartes et équipements complexe d'entrée/sortie.

Fichiers optionnels

En plus des fichiers essentiels à la re-compilation du projet, certains fichiers peuvent être embarqués avec les sources de l'application. Les options suivantes vous permettent de sélectionner les fichiers optionnels à ajouter:

- Descripteur du projet
S'il n'a pas été embarqué, le descripteur du projet télé-importé indiquera seulement la date de télélecture.

- Protection par mots de passe
La fonction de télélecture n'est pas protégée par un mot de passe. Si vous désirez que le projet reste protégé après la télélecture, vous devez embarquer son système de protection avec le code source.

- Commentaires des voies d'E/S non câblées
ISaGRAF offre la possibilité d'affecter des textes de description des voies d'E/S non utilisées. Ne validez pas cette option si vous ne travaillez qu'avec des voies effectivement câblées.

- Historique des modifications
C'est l'historique global des modifications apportées au projet.

- Fichiers agenda
Les fichiers agenda contiennent des notes de maintenance et les résultats de compilation de chaque programme. L'embarquement de ces fichiers peut requérir beaucoup de mémoire supplémentaire sur la cible.

- Listes de variables et chronogrammes
Ces fichiers sont créés pendant la mise au point, et contiennent les listes de variables et de chronogrammes disponibles pour le testeur.

- Synoptiques, icones et bitmaps
Cette option regroupe la définition des synoptiques ISaGRAF, ainsi que tous les fichiers bitmaps (BMP) ou icone (ICO) attachés aux synoptiques, s'ils sont stockés dans le répertoire du projet. Attention: l'embarquement de ces fichiers peut requérir beaucoup de mémoire supplémentaire sur la cible.

A.19.4 Comment les sources compressées sont stockées dans la cible

Les sources compressées embarquées (EZS) sont stockées sous la forme d'une "ressource" dans l'application ISaGRAF. La ressource générée est nommée "**EZS**". Vous ne pouvez donc pas utiliser ce nom pour une autre ressource. L'embarquement des sources pour la télélecture n'implique aucune autre restriction dans l'utilisation des ressources. Le fichier de définition des ressources saisi par l'utilisateur n'est pas affecté par la télélecture.
Référez-vous à la documentation du générateur de code d'ISaGRAF pour plus de détails sur l'utilisation des ressources dans les applications ISaGRAF.

A.19.5 Utilisation de la mémoire sur la cible

Les sources compressées embarquées (EZS) consomment de la mémoire supplémentaire dans la cible. Une estimation générale approximative est que les sources embarquées minimales (aucun fichier optionnel sélectionné) ont de 1,5 à 2 fois la taille du code de l'application. Ceci signifie que le code téléchargé pour une application incluant les sources embarqués sera environ 2,5 fois plus gros.
D'autres types de limitations peuvent apparaître selon le type de système cible. Les sources embarquées doivent être stockées dans le même segment de données que le code de l'application.

A.19.6 Le projet récupéré

Le projet télé-importé contient toutes les informations nécessaires à sa re-compilation (génération du code). Selon les options choisies à la préparation du projet initial, il peut également contenir d'autres fichiers tels que le descripteur du projet est les fichiers agenda.
Vous devez générer le code de l'application avant de pouvoir la tester. Attention: ISaGRAF utilise la date de compilation pour l'identification du code des applications. Au lancement du debugger, un message vous informera que vous ne travaillez pas sur la même version de code que la cible.

Note: Les librairies ne sont pas téléchargées avec les applications ISaGRAF. Vous devez vous assurer que tous les éléments de librairie nécessaire au projet sont correctement installés dans l'atelier avant de télé-importer un projet.

A.19.7 Compatibilité

La télélecture est supportée par les cibles ISaGRAF version 3.23 ou plus. Des extensions au protocole de communication ont été implémentées.
Il n'y a aucune restriction à embarquer les sources (EZS) dans une cible ISaGRAF version 3.20 ou 3.21, puisque les sources sont stockées comme de simples ressources. Mais l'application ne pourra pas être télé-importée dans ce cas, puisque les services de communication requis ne sont pas supportés par ces cibles.

A.20 L'outil de diagnostic

L'outil de " **diagnostic**" est un sous-ensemble du debugger d' ISaGRAF. Il permet à l'utilisateur de travailler sur les variables du projet, et offre les fonctions d'un poste opérateur pour le suivi du procédé. Le debugger d'ISaGRAF est un outil qui offre des possibilités puissantes. L'outil de diagnostic par la limitation de ces possibilités, offre un environnement sécurisé pour la conduite du procédé et les opérations de maintenance. L'outil de diagnostic d'ISaGRAF est lancé directement depuis le menu "Démarrer":

La liste des projets existants est affichée dans une boîte de dialogue. Pour lancer le debugger réduit sur un projet déjà téléchargé dans la cible, sélectionnez le projet dans la liste et pressez le bouton "**OK**". Le bouton "**Abandonne**" permet de fermer la boîte de dialogue. Le bouton "**Configurer**" permet de positionner les paramètres de la liaison avec l'automate. Référez-vous au chapître "**Gestion de programmes**" pour plus d'information sur la configuration de la liaison.

Note: L'outil de diagnostic d'ISaGRAF ne permet pas de télécharger l'application, de l'arrêter ni de la mettre à jour par une modification en ligne. Aucune opération n'est possible si le projet sélectionné n'est pas celui en cours d'exécution sur la cible.

Quand l'outil de diagnostic est lancé et que la connexion avec la cible est établie, les fonctions suivantes sont disponibles::

- Espionner des listes de variables
- Espionner des images graphiques avec SpotLight

A.21 Le simulateur

Le simulateur ISaGRAF est lancé avec le debugger par la commande "**Simuler**" du menu "**Test**" dans la fenêtre du Gestionnaire de Programmes. Le simulateur ISaGRAF est un système cible supportant toutes le fonctionnalités d'un automate ISaGRAF, et les fonctions et blocs fonctionnels "C" de la librairie standard délivrée par CJ international. Les cartes d'E/S sont simulées graphiquement dans une fenêtre. Tout type de carte d'E/S peut être simulé. Les cartes définies comme "**Cartes Virtuelles**" pendant le câblage des E/S apparaissent également dans la fenêtre du simulateur.

A.21.1 Liens avec le debugger

Le simulateur supporte un dialogue complet avec le debugger d'ISaGRAF. Ainsi, toutes les possibilités de mise au point sont encore valides pendant la simulation. Le simulateur travaille toujours sur l'application ISaGRAF courante. Pendant la simulation, les commandes "**Activer**", "**Arrêter**", "**Transférer**" et "**Mise à jour**" du debugger ne sont plus valides. Le simulateur ne peut pas être utilisé si la cible "**SIMULATE**" n'a pas été sélectionnée pour la génération du code de l'application. La fermeture de la fenêtre du simulateur entraine la fermeture du debugger et de toutes les fenêtres ISaGRAF ouvertes pendant la session de mise au point.

A.21.2 Simulation des E/S

Les cartes d'E/S apparaissent dans la fenêtre du simulateur, repérées par leur numéro d'emplacement. Tous les types standard d'entrées/sorties ISaGRAF (booléen, analogique ou message) peuvent être simulés. Les voies d'une carte d'entrée sont représentées par des boutons ou des champs de saisie. Les voies d'une carte de sortie sont représentées par des objets graphiques animés.

Entrée booléenne: Une voie d'entrée booléenne est représentée par un bouton. Le numéro de la voie est affiché à côté du bouton. L'entrée vaut TRUE quand le bouton est enfoncé. Il suffit de cliquer sur le bouton pour inverser l'état de l'entrée correspondante.

Sortie booléenne: Une voie de sortie booléenne est représentée par un petit cercle. Le numéro de la voie est affiché à côté du cercle. La cercle est coloré quand la valeur de la sortie est TRUE.

Entrée analogique: Une voie d'entrée analogique est représentée par un simple champ de saisie, où la valeur de l'entrée peut être saisie. Cliquez sur le champ pour faire apparaître le curseur de saisie, et entrez la nouvelle valeur. Aucune confirmation par la touche **ENTREE** n'est nécessaire. Les entrées analogiques peuvent être saisies en décimal ou en hexadécimal.

Sortie analogique: Une voie de sortie analogique est représentée par un pavé d'affichage numérique rouge. La valeur de la sortie peut être affichée en décimal ou en hexadécimal. Aucune action ne peut être effectuée sur une voie de sortie.

Entrée message: Une voie d'entrée message est représentée par un simple champ de saisie, où la valeur de l'entrée peut être saisie. Cliquez sur le champ pour faire apparaître le curseur de saisie, et entrez la nouvelle valeur. Aucune confirmation par la touche **ENTREE** n'est nécessaire.

Sortie message: Une voie de sortie message est représentée par un pavé d'affichage textuel. Aucune action ne peut être effectuée sur une voie de sortie.

A.21.3 Eléments de la librairie

Le simulateur ISaGRAF supporte l'ensemble des conversions, fonctions et blocs fonctionnels standards fournis par CJ International. Voici la liste des objets supportés:

- ***Fonctions de conversion:***

  bcd, scale

- ***Fonctions:***

  abs, acos, ArCreate, ArRead, ArWrite, ascii, asin, atan, char, cos, delete, expt, find, insert, left, limit, log, max, mid, min, mlen, mod, mux4, mux8, odd, rand, replace, right, rol, ror, sel, shl, shr, sin, sqrt, tan, trunc

- ***Blocs fonctionnels:***

  average, blink, cmp, ctd, ctu, ctud, derivate, f_trig, hyster, integral, lim_alrm, r_trig, rs, sema, sr, stackint, tof, ton, tp

Les conversions, fonctions et blocs fonctionnels "C" écrits par l'utilisateur ne peuvent pas être intégrés au simulateur d'ISaGRAF. Typiquement, ces objets prennent en compte les ressources matérielles et logicielles du système cible, et ces ressources n'ont généralement aucun sens dans l'environnement Windows. Toutefois, le simulateur d'ISaGRAF assure le comportement standard suivant pour toute nouvelle conversion, fonction ou tout nouveau bloc fonctionnel:

- Quand une nouvelle conversion est traitée par le simulateur, elle est remplacée par une conversion "sans effet". La valeur physique d'une entrée ou sortie analogique est toujours égale à sa valeur électrique (saisie ou affichée dans le panneau de simulation).
- Quand une nouvelle fonction (ou bloc fonctionnel) "C" est traitée par le simulateur, aucune opération n'est effectuée. La valeur de retour n'est pas mise à jour.

A.21.4 Options

Les commandes du menu "**Options**" permettent à l'utilisateur de contrôler les affichages des voies d'E/S dans le panneau de simulation. Ces options peuvent être inhibées ou validées à tout moment pendant la mise au point d'une application.

- Quand l'option "**Affichage couleur**" est validée, les voies d'E/S sont affichées sous la forme de dessins en couleurs. Si ces couleurs ne peuvent pas être clairement distinguées sur un écran LCD monochrome, l'utilisateur peut enlever cette option, afin de forcer tous les affichages en noir et blanc.
- Quand l'option "**Noms de variables**" est validée, une étiquette est affichée à côté de chaque voie d'E/S, avec le nom de la variable d'E/S connectée. Enlever cette option permet de réduire la taille du panneau de simulation.
- Quand l'option "**Valeurs hexadécimales**" est validée, toutes les voies analogiques en entrée ou en sortie sont affichées ou saisies dans un format hexadécimal.
- Quand l'option "**Toujours visible**" est validée, la fenêtre du simulateur reste toujours au dessus des autres fenêtres, même quand elle n'est pas active.

A.21.5 Enregistrer et restituer l'état des sorties

Lors de l'utilisation du simulateur d'ISaGRAF, les voies d'entrée sont forcées manuellement pas des actions sur les boutons et contrôles du panneau de simulation. Vous pouvez à tout moment utilisez les commandes suivantes du menu "**Outils**" pour enregistrer ou restituer l'état de toutes les entrées:

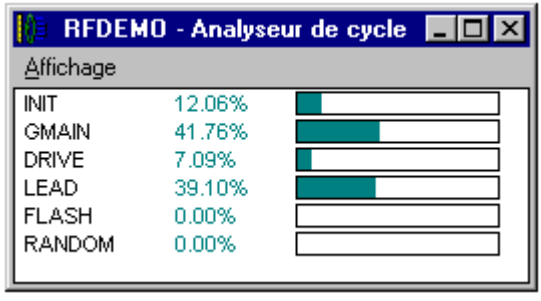
Charger un jeu d'entrées	Force la valeur des voies d'entrée avec les valeurs contenues dans une fichier créé préalablement par la commande "Sauver le jeu d'entrées".
Sauver le jeu d'entrées	Enregistre dans un fichier l'état des entrées, afin de pouvoir le restituer plus tard à l'aide de la commande "Charger un jeu d'entrées". Le fichier de valeurs est enregistré dans le répertoire du projet, et sera pris en compte par le gestionnaire d'archives d'ISaGRAF avec les autres données du projet.

Note: Seules les entrées nommées (celles ayant une variable déclarée connectée) sont sauvées et restituées.

A.21.6 L'analyseur de cycle

L'analyseur de cycle d'ISaGRAF est un outil puissant de diagnostic qui montre comment le temps de cycle est réparti dans les différents programmes, fonctions et blocs fonctionnels de l'application. Cet outil est très utile pour établir un diagnostic rapide sur les performances de l'application, et dirige le programmeur vers les parties du cyles qui nécessitent une optimisation.

L'analyseur de cycle est lancé par la commande "**Outils / Analyseur de cycle**" dans la fenêtre du simulateur ISaGRAF. Il affiche, pour chaque programme, fonction ou bloc fonctionnel, le pourcentage du temps de cycle dépensé pour son exécution:

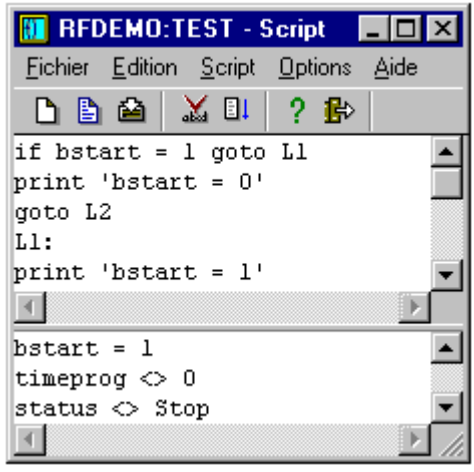


Quand l'option "**Affichage / Moyenne**" est validée, les pourcentages affichés sont moyennés. La moyenne est calculée depuis le lancement de l'application, ou depuis la dernière commande "**Affichage / Ré-initialisation**".
Quand l'option "**Affichage / Moyenne**" n'est pas validée, les pourcentages affichés sont ceux calculés pendant le dernier cycle exécuté. Ce mode peut également être utilisé quand l'application est en mode "**Cycle à Cycle**", et vous permet d'avoir un calcul correspondant au contexte courant de l'application.
Utilisez la commande "**Affichage / Copier**" pour copier les noms de programmes et les pourcentage dans le presse-papiers de Windows, en format texte. Ces données peuvent ensuite être collées dans d'autres applications telles que des éditeurs de textes ou des tableurs.

Notes: Il ne s'agit pas de mesures de temps précises. Les pourcentages affichés sont basés sur la mesure de chaque instruction TIC élémentaire, en tenant compte du ratio entre les différentes instructions. La mesure ne prend pas en compte le temps passé dans les fonctions et les blocs fonctionnels en "C".
La valeur affichée pour une fonction ou un bloc fonctionnel est la somme de tous les appels effectués pendant le cycle.
La mesure des pourcentage est basée sur l'évaluation des codes TIC et ne représente pas une information fiable si l'application est destinée à être traduite en code source "C" et compilée par un compilateur "C".

A.21.7 Scripts de simulation

Le simulateur d'ISaGRAF inclut un outil pour l'édition et l'exécution de scripts de simulation. Un script est décrit à l'aide d'un langage simple de type ST, et permet d'automatiser les tests et la validation des application simulées.
L'éditeur de scripts est lancé par la commande "**Outils / Scripts de simulation**" depuis la fenêtre du simulateur. Voici la fenêtre d'édition des scripts:



La fenêtre du haut est un éditeur de texte où les instructions du script sont entrées. Il est utilisé comme les autres éditeurs de texte d'ISaGRAF, et inclut quelques fonctionalités puissantes telles que l'insertion par sélection du symbole d'une variable. Utilisez les commandes du menu "**Options**" pour changer la largeur des taquets de tabulation ou la police de caractères.
La fenêtre du bas montre tous les messages produits lors de l'exécution du script. La ligne de séparation entre les deux fenêtres peut être glissée avec la souris pour redimensionner les fenêtres. La fenêtre des messages peut être cachée pendant l'édition du script, mais elle est automatiquement ré-ouverte à chaque exécution du script.

Edition des scripts

Utilisez les commandes du menu "**Fichier**" pour gérer les fichiers de script:
Nouveau.....crée un nouveau script sans titre
Ouvrir.....charge un script stocké sur disque
Enregistrer.....enregistre sur disque le contenu du script et de la fenêtre de messages, dans le répertoire du projet
Enregistrer sous.....enregistre le script sous un autre nom

Deux fichiers sont créés pour chaque script dans le répertoire du projet ISaGRAF:
<*nomscript*>.SCC.....contenu du script (instructions)
< *nomscript* >.SCO....contenu de la fenêtre de messages
Où < *nomscript* > est le nom du script. Ces deux fichiers sont des fichiers de texte ASCII qui peuvent être librement ouverts par d'autres applications.

Pendant l'édition d'un script, utilisez la commande "**Edition / Insérer un symbole**" pour sélectionner le nom d'une variable déclarée et l'insérer à la position courante du curseur.

Exécution des scripts

Les scripts doivent être vérifiés avant d'être exécutés. Si nécessaire, la vérification est lancée automatiquement avant l'exécution du script. Utilisez les commandes suivantes du menu "**Script**":

Vérifier: vérifie la syntaxe du script et le compile

Démarrer le script: lance l'exécution du script édité

Dans le cas d'un nouveau script sans titre, il faut l'enregistrer (et lui donner un nom) avant de le vérifier. Dans le cas d'un script ayant déjà un nom, son contenu est automatiquement enregistré avant la vérification.
Quand un script est en cours d'exécution, son contenu ne peut plus être changé. Un message est affiché quand son exécution est terminée. Vous pouvez aussi interrompre l'exécution du script à l'aide de la commande suivante du menu "**Script**":

Arrêter le script: abandonne l'exécution du script

L'exécution du script est faite entre les cycles ISaGRAF. Dans le cas d'une boucle infinie, programmée dans un cycle, le simulateur ISaGRAF assure que la boucle sera interrompue de façon à ce que les cycles ISaGRAF puissent continuer à être exécutés, et à ne pas bloquer les autres applications. L'interpréteur de scripts d'ISaGRAF décide d'interrompre une boucle à chaque fois qu'une étiquette est rencontrée plus d'une fois dans le même cycle d'exécution. L'exécution des scripts peut aussi être explicitement interrompue par l'utilisation des instructions "Cycle" ou "Wait".

Langage de description des scripts

Le langage de description des scripts est un langage littéral très simple, similaire au langage ST, mais où chaque instruction est entrée sur une ligne, et où l'instruction n'a pas besoin d'être terminée par un point-virgule. Utilisez le bouton suivant dans la barre d'outils pour connaître les instructions disponibles et insérer le mot clé d'une instruction à la position du curseur:

insère une instruction (mot clé et aide sous forme de commentaires)

Il y a différents groupes d'instructions. Le premier concerne l'assignation (forçage) des variables:
:= ..... assignation

D'autres instructions permettent d'envoyer des informations dans la fenêtre de messages:
Print..... affiche un texte ou la valeur d'une variable
PrintTime ..... affiche l'heure courante

D'autres instructions permettent de synchroniser le script avec les cycles ISaGRAF:
Cycle..... laisse le simulateur exécuter un cycle ISaGRAF
Wait ..... suspend le script pendant un temps

D'autres instructions permettent de contrôler le flux d'exécution des instructions du script:
Etiquettes..... peuvent être insérées partout dans le cycle
Goto..... saut inconditionnel à une étiquette
If goto..... saut conditionnel à une étiquette
End..... termine l'exécution du script

Le langage de description des scripts ne différentie pas les majuscules et les minuscules. Des commentaires peuvent être entrés à la fin de chaque ligne. Les commentaires peuvent être entrés selon la syntaxe ST (entre "**(***" et "***)**"), ou plus simplement préfixés par le caractère "**;**".

":=" Forçage

Description: Force la valeur d'une variable ISaGRAF. Il peut s'agir d'une variable interne, ou d'une voie d'entrée ou de sortie.

Syntaxe: <varname>:**=** <constant_expression>
<varname> **=** <constant_expression>

Arguments: <varname> est un symbole de variable déclarée dans l'application, ou une E/S représentée directement selon les conventions d'écriture utilisant le caractère "%".
<constant_expression> est une expression constante valide, en accord avec le type de la variable. Pour les booléens, les valeurs "0" et "1" peuvent être utilisées à la place de "FALSE" et "TRUE". Pour les temporisations, le préfixe "T#" ou "TIME#" peut être omis.

Notes: Les variables d'entrées forcées depuis le script n'ont pas besoin d'être verrouillées dans l'application ISaGRAF. Le dessin de la voie d'entrée correspondant est mis à jour dans le panneau de simulation.
Attention: ne forcez pas de variable d'entrée ou de sortie analogique ayant une conversion. L'exécution des scripts ne prend pas en compte les conversions (fonctions ou tables).

Exemple:
```
MyBooVar:= 1          (* équivalent à TRUE *)
MyIntVar:= 1234
MyRealVar:= 1.2345
MyMsgVar:= 'Hello'
MyTmrVar:= t#12s
```

Print

Description: Envoie une chaine de caractères ou la valeur courante d'une variable dans la fenêtre de messages. Le texte est formatté sur une seule ligne, qui est ajoutée à la fin des messages.

Syntaxe: **Print '***<text>***'**
Print <*varname*>

Arguments: <text> est une chaîne de caractères exprimée entre apostrophes
<varname> est un symbole de variable déclarée dans l'application, ou une E/S représentée directement selon les conventions d'écriture utilisant le caractère "%"

Notes: Le formattage des valeurs de variables est conforme aux conventions d'écritures de l'IEC.

Exemple:
```
Print 'Hello'
Print MyBooVar
```

Messages:
```
Hello
MyBoovar = TRUE
```

PrintTime

Description: Envoie l'heure courante dans la fenêtre des messages. Le texte est formatté sur une seule ligne, qui est ajoutée à la fin des messages.

Syntaxe: **PrintTime**

Notes: L'heure est formattée selon le paramétrage du système Windows

Exemple:
```
Print 'Il est maintenant:'
PrintTime
```

Messages:
```
Il est maintenant:
15:45:22
```

Cycle

Description: Suspend l'exécution du script jusqu'à l'exécution du prochain cycle ISaGRAF.

Syntaxe: **Cycle**

Notes: Les instructions du script sont exécutées en début de cycle ISaGRAF. Si le simulateur est en mode "Cycle à Cycle", l'instruction "cycle" est immédiatement suivie par l'exécution d'un cycle ISaGRAF. Les instructions suivantes du script seront exécutées lors de la prochaine commande "Exécuter un cycle" depuis le debugger ISaGRAF.

Exemple:
```
(* le programme ISaGRAF copie A dans B *)
A:= 0
Cycle
Print B
A:= 1
Print B  (* aucun cycle effectué *)
```

```
                (* B n'est pas encore forcé à 1 *)
Cycle
Print B
```

Messages:
```
B = 0
B = 0
B = 1
```

Wait

Description: Suspend l'exécution du script pendant la durée passée en paramètre

Syntaxe: **Wait** <delay>

Arguments: <delay> durée exprimée selon les conventions d'écriture de l'IEC pour les constantes temporelles. Le préfixe "T#" ou "TIME#" peut être omis. La durée doit être comprise entre 10 millisecondes et 1 heure.

Notes: La durée effective des instructions "Wait" n'est pas précise et dépend notament du système Windows. Aussi, la durée devra être assumée avec une précision de plus ou moins un cycle ISaGRAF.
Quand une instruction "Wait" est rencontrée, les cycles ISaGRAF sont exécutés jusqu'à ce que la durée soit écoulée et avant l'exécution des instructions suivantes du script.

Exemple:
```
PrintTime
Wait 2s
PrintTime
```

Messages:
```
15:45:27
15:45:29
```

Etiquettes

Description: Des étiquettes peuvent être placées n'importe où dans le script. Elles sont utilisées comme destination par les instructions de branchement "Goto" et permettent le contrôle du flux d'exécution des instructions du script.

Syntaxe: <labelname>**:**

Arguments: <labelname> nom unique respectant les conventions de nomenclature des variables ISaGRAF: limité à 16 caractères, commençant par une lettre suivie de lettres, de chiffres ou de caractères soulignés. Lors de sa définition, le nom de l'étiquette doit être suivie par le caractère "**:**".

Notes: Aucune instruction ne doit être placée sur la ligne où l'étiquette est définie.
Une étiquette ne peut pas avoir le même nom qu'une variable ISaGRAF

Exemple:
```
(* exemple de script avec une boucle infinie *)
loop:
PrintTime
Wait 1s
Goto loop
```

Goto

Description: Saut inconditionnel à une étiquette

Syntaxe: **Goto** <labelname>

Arguments: <labelname> est le nom d'une étiquette déclarée dans le script

Notes: Les sauts en arrière sont autorisés. Dans le cas d'une boucle infinie, l'exécution du script est automatiquement interrompue pour assurer l'exécution des cycles ISaGRAF.

Exemple:
```
Print 'Avant le saut'
Goto MyLabel
Print 'Pendant' (* instruction jamais exécutée *)
MyLabel:
Print 'Après le saut'
```

Messages:
```
Avant le saut
Après le saut
```

If Goto

Description: Saut conditionnel à une étiquette. La condition peut être la comparaison entre deux variables ISaGRAF, ou la comparaison entre une variable ISaGRAF et une expression constante.

Syntaxe: **If** <var1> ***test*** <var2> **Goto** <labelname>
If <var1> ***test*** <constant_expression> **Goto** <labelname>

Les ***tests*** de comparaison disponibles sont:

= vrai si les deux membres ont la même valeur
<> vrai si les deux membres ont des valeurs différentes
< vrai si le premier membre est inférieur au second
<= vrai si le premier membre est inférieur ou égal au second
> vrai si le premier membre est supérieur au second
>= vrai si le premier membre est supérieur ou égal au second

Arguments: <var1> <var2> sont des symboles de variables déclarées dans l'application, ou des E/S représentées directement selon les conventions d'écriture utilisant le caractère "%"
<constant_expression> est une expression constante valide, en accord avec le type de la variable. Pour les booléens, les valeurs "0" et "1" peuvent être utilisées à la place de "FALSE" et "TRUE". Pour les temporisations, le préfixe "T#" ou "TIME#" peut être omis.
<labelname> est le nom d'une étiquette déclarée dans le script

Notes: Les sauts en arrière sont autorisés. Dans le cas d'une boucle infinie, l'exécution du script est automatiquement interrompue pour assurer l'exécution des cycles ISaGRAF.

Exemple:
```
(* Ce script boucle jusqu'à ce que MyVar = TRUE *)
Loop:
If MyVar = TRUE Goto TheEnd
Print MyVar
Goto Loop
TheEnd:
```

End

Description: Termine le script

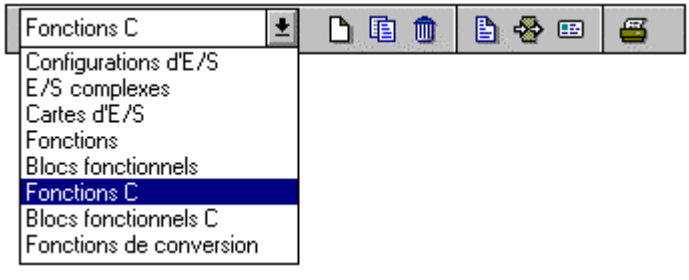
Syntaxe: **End**

Notes: Il n'est pas obligatoire de placer une instruction "End" sur la dernière ligne du script

Exemple:
```
(* Ce script boucle jusqu'à ce que MyVar = TRUE *)
Loop:
If MyVar = FALSE Goto Continue
End
Continue:
Print MyVar
Goto Loop
```

A.22 Gestion des librairies

Les librairies ISaGRAF constituent l'interface entre le développement des applications et les ressources **logicielles** et **matérielles** du système cible ISaGRAF. Il y a une librairie pour chaque type d'interface. Le gestionnaire de librairies d'ISaGRAF est dédié au constructeur de matériel ou à l'ingénieur informaticien. Il est utilisé pour la description des interfaces des ressources qu'il ajoute à l'automate.
Le gestionnaire de librairies d'ISaGRAF montre les éléments d'une des librairies ISaGRAF. Dans la partie gauche de la fenêtre apparaît la **liste des éléments** de la librairie sélectionnée. Dans la partie droite est affichée la **fiche technique** (guide d'utilisation) de l'élément sélectionné dans la liste. Les menus du gestionnaire de librairies regroupent les commandes permettant de créer, définir et modifier les éléments de la librairie sélectionnée. La commande "**Autre librairie**" permet d'ouvrir une autre librairie. La boîte de sélection à gauche de la barre d'outil peut également être utilisée pour sélectionner une librairie:



A.22.1 Manipuler les éléments de librairie

Utilisez les commandes du menu "**Fichier**" pour créer et manipuler les élémernts de la librairie sélectionnée.

Créer un nouvel élément

Utilisez la commande "**Nouveau**" du menu "**Fichier**" pour créer un nouvel élément dans la librairie sélectionnée. Entrez le nom du nouvel élément en respectant les règles de nomenclature suivantes:

- la longueur du nom ne doit pas excéder **8** caractères,
- le premier caractère doit être une **lettre**,
- les caractères suivants doivent être des **lettres**, des **chiffres** ou '_'.
- Les majuscules et les minuscules ne sont pas différenciées.

Un commentaire est associé à chaque élément de librairie. Le texte de ce commentaire est saisi à la création de l'élément. Quand le nouvel élément est créé, vous devez saisir:

- ses paramètres pour une carte d'E/S,
- sa définition pour une configuration d'E/S

- son interface pour une fonction ou un bloc fonctionnel.

Quand une conversion, une fonction "C" ou un bloc fonctionnel "C" est créé, la trame complète de son code source "C" est automatiquement générée.

Manipuler les éléments existants

La commande "**Fichier / Renommer**" vous permet de changer le nom et le commentaire de l'élément sélectionné dans la liste. La commande "**Fichier / Copier**" vous permet de copier l'élément sélectionné de la librairie active dans un autre élément de la même librairie. Si l'élément de destination existe déjà, tout son contenu sera écrasé par la copie. S'il n'existe pas encore, il sera automatiquement créé. La commande "**Fichier / Supprimer**" détruit l'élément sélectionné et le retire de la librairie. Tous les composants de l'élément sont détruits. Les composants suivants sont manipulés par les commandes "**Renommer**", "**Copier**" et "**Supprimer**":

- fiche technique
- définition complète pour une configuration d'E/S
- paramètres d'une carte ou d'un équipement complexe d'E/S
- interface pour une fonction ou d'un bloc fonctionnel
- code source d'une fonction écrite en langage IEC
- code source C des fontions, blocs fonctionnels et fonctions de conversion

⚠ si l'élément est une conversion, une fonction ou un bloc fonctionnel, son nom n'est pas automatiquement remis à jour dans son code source "C" par les commandes "**Copier**" et "**Renommer**".

⚠ si l'élément est une fonction en langage IEC, le nom du paramètre de retour n'est pas mis à jour par les commandes "**Renommer**" ou "**Copier**".

Protection par mot de passe

La commande "**Fichier / Mot de passe**"permet de protéger l'élément de librairie sélectionné à l'aide de mots de passe. Consultez la section "**Protection par mots de passe**" à la fin de la première partie de ce document pour plus de détails sur le système de protection et les niveaux d'accès. La protection est dédié à l'élément sélectionné et n'influe pas sur les autres éléments ni les autres librairies ISaGRAF.

Compiler les fonctions et les blocs fonctionnels

Quand la librairie des fonctions ou blocs fonctionnels écrits en langage IEC est sélectionnée, la commande "**Vérifier/compiler**" du menu "**Fichiers**" permet de vérifier la syntaxe de la fonction ou du bloc et de générer son code objet. Les fonctions et les blocs doivent être compilés avant d'être utilisés dans les projets ISaGRAF. Cette commande est sans effet si une autre librairie est sélectionnée.

Fiches techniques

La commande "**Edition / Fiche technique**" lance l'édition de la fiche technique de l'élément sélectionné dans la librairie. L'édition se fait à l'aide de l'éditeur de texte d'ISaGRAF. La fiche technique d'un élément est son **guide d'utilisation**. Elle sera consultée par le programmeur automaticien lors de l'intégration de l'élément dans

une application ISaGRAF. La fiche technique doit en particulier présenter la description générale de la fonction de l'élément, la description détaillée de son interface de programmation ou de ses paramètres, son contexte et ses limites d'utilisation., et des exemples d'utilisation ou cas typiques.
La commande "**Outils / Format de fiche technique**" vous permet de définir un format standard pour tous les éléments de la librairie sélectionnée. Cette trame sera reprise pour initialiser la fiche technique de chaque nouvel élément créé dans la librairie. Cette fonction permet de réduire le travail de saisie des fiches, et d'assurer un aspect cohérent pour tous les éléments d'une même librairie.

Paramètres

Les paramètres d'un élément décrivent l'**interface** entre son implémentation logicielle et son utilisation dans une application ISaGRAF.
Les paramètres ont une signification différente pour chaque type de librairie. Les paramètres d'une configuration d'E/S définissent un ensemble de cartes d'E/S et des noms par défaut pour chacune des voies. Les paramètres d'une carte ou d'un équipement complexe d'E/S définissent la configuration matérielle et logicielle de la carte. Les paramètres d'une fonction ou d'un bloc fonctionnel décrivent son interface, selon les règles sémantiques d'appel de fonction du langage ST. Il n'y a pas de paramètre pour une fonction de conversion, car elle utilise une interface prédéfinie, et non modifiable.

Code source C

L'atelier ISaGRAF permet à l'informaticien de gérer le code source des conversions, fonctions et blocs fonctionnels de la librairie. Le code source d'une fonction ou d'un bloc en langage IEC est un texte ou un diagramme correspondant au langage choisi pour la fonction. Le code source d'un élément "C" est divisé en deux fichiers: un fichier de **définitions** qui contient la description exacte de son **interface**, conformément à la définition des paramètres de l'élément, et un fichier de **code source** qui contient l'implémentation de la fonction "C" réalisée pour l'élément.
Le gestionnaire de librairie d'ISaGRAF génère automatiquement la trame du code source quand un nouvel élément est créé. Il crée également, et tient à jour, le fichier de définitions, à chaque fois que les paramètres de l'élément sont modifiés. L'informaticien peut utiliser l'éditeur de texte d'ISaGRAF pour compléter le fichier de code source.

Archiver les éléments de librairie

Utilisez la commande "**Outils / Archiver**" pour lancer le gestionnaire d'archivage d'ISaGRAF, qui vous permet de sauvegarder ou de restituer des éléments de la librairie sélectionnée. L'archiveur ne permet pas de gérer les élements des autres librairies. Pour celà, il vous faut tout d'abord sélectionner le type de librairie dans le Gestionnaire de Librairie.

A.22.2 Configuration d'E/S

La librairie de configurations d'E/S d'ISaGRAF offre un moyen facile et rapide pour initialiser les nouveaux projets ISaGRAF avec une configuration d'E/S pré-établie. Une configuration définit:

- un ensemble de cartes et d'équipements complexes
- des valeurs par défaut pour les paramètres des cartes
- des noms par défaut pour les voies des cartes

Quand un nouveau projet ISaGRAF est créé avec une configuration d'E/S, le câblage des E/S correspondant est automatiquement réalisé, et les variables d'E/S associées aux voies des cartes sont directement déclarées dans le dictionnaire du projet créé.

La définition d'une configuration d'E/S est réalisée à l'aide de l'éditeur de câblage d'E/S d'ISaGRAF (le même outil utilisé pour les projets). Référez-vous à la section "**Câblage des E/S**" dans ce document pour une description complète de cet outil. Quand une carte est insérée dans une configuration, toutes ses voies sont nommées avec des libellés standard. Le nom standard d'une voie d'E/S respecte le format suivant:
<direction><type><emplacement>_<voie>

Le premier caractère indique la direction du canal d'E/S:
"**I**" ..... en entrée
"**Q**" ..... en sortie

Le second caractère indique le type de donnée associée:
"**X**"..... booléen
"**D**"..... analogique
"**M**" ..... message

Voici des exemples de noms standards pour des voies d'E/S:
IX0_7..... entrée booléenne - carte #0 - voie #7
QD2_4..... sortie analogique - carte #2 - voie #4

La commande de câblage des voies de l'éditeur de câblage peut être utilisée pour modifier le nom par défaut proposé par ISaGRAF. Les noms associés aux voies peuvent être librement modifiés, ou supprimés.

A.22.3 Equipement complexe d'E/S

ISaGRAF impose que toutes les voies d'une carte d'E/S simple aient le même type (booléen, analogique ou message) et la même direction (entrée ou sortie). Un équipement d'E/S complexe représente un dispositif d'E/S mêlant des canaux de type et de direction différents. Un équipement complexe est représenté comme une liste de cartes élémentaires, et n'utilise qu'un emplacement dans le rack de câblage de l'application.

Pour définir un équipement complexe, il faut définir chacun de ses composants comme une carte élémentaire. Chaque carte élémentaire doit être nommée. Il faut ensuite entrer les paramètres de chacune des cartes composant l'équipement. Les composants d'un équipement complexe sont définis dans une boîte de dialogue.
Le bouton "**Ajoute**" permet d'ajouter une nouvelle carte élémentaire à la fin de la liste existante. Le bouton "**Insère**" permet d'insérer une carte à la position

sélectionnée. Le bouton "**Supprime**" retire la carte élémentaire sélectionnée de la liste. Les bouton "**Renomme**" et "**Paramètres**" permettent de définir le nom et les paramètres de la carte élémentaire sélectionnée dans la liste.Consultez la section suivante de ce chapitre pour une description complète des paramètres d'une carte. Un équipement complexe peut grouper jusqu'à **16** cartes élémentaires. Le nom d'une carte (dans l'équipement), ne peut pas excéder **8** caractères.

A.22.4 Carte d'E/S

La librairie ISaGRAF des cartes d'E/S fournit une interface standard entre les applications développées et leur implémentation sur un matériel. Pendant la programmation de l'application, toutes les variables d'E/S seront **câblées** sur les voies des cartes d'E/S implantées dans l'automate. Une carte d'E/S ISaGRAF est définie par un **nom** et une "**Clé OEM**" qui identifie son **constructeur**. Les autres paramètres de la carte d'E/S décrivent la topologie de la carte (nombre de voies, direction et type des voies), et sa configuration logicielle ou matérielle.

Les paramètres d'une carte d'E/S

On distingue deux types de paramètres pour une carte d'E/S. Les paramètres communs sont définis pour toutes les cartes de la librairie ISaGRAF. Les paramètres OEM sont spécifiques à l'implémentation de la carte, et sont définis par son constructeur. Les paramètres communs sont saisis dans la partie supérieure de la boîte de dialogue pour la saisie des paramètres. Ces paramètres (en plus du nom de la carte d'E/S) constituent l'interface ISaGRAF standard pour une carte d'E/S.
Le "**Code clé OEM**" est un nombre qui identifie le **constructeur** de la carte. Toutes les cartes définies par le même constructeur ont la même clé OEM. La clé OEM est un **nombre non signé de 16 bits**, saisie dans un format **hexadécimal**. La clé OEM réservée pour **CJ International** est "**1**".
Les paramètres principaux définissent la topologie de la carte. Le **nombre de voies** définit le nombre de canaux d'E/S disponibles sur la carte. Le **type** de la carte indique le type des variables qui peuvent être câblées sur la carte. La **direction** définit si les variables câblées sur la carte sont des variables d'**entrée** ou de **sortie**.

Note: ISaGRAF ne permet pas de grouper des variables de type ou de direction différents sur une même carte d'E/S.

Les paramètres OEM

Les paramètres OEM sont saisis dans la partie inférieure de la boîte de dialogue. Ces paramètres sont choisis par le constructeur de la carte. Ils sont spécifiques pour chaque carte. Il y a au maximum **16** paramètres OEM pour une carte. Une carte peut ne pas avoir de paramètre OEM. Le gestionnaire de librairies d'ISaGRAF permet au concepteur de la carte de définir l'identification et le format de chacun de ces paramètres, et la façon dont ils seront renseignés par l'automaticien.
La partie gauche de la boîte de dialogue contient la liste des paramètres OEM. Chaque paramètre est identifié par un **nom** et un **numéro** logique, entre **0** et **15**. La partie droite de la boîte de dialogue contient la description détaillée du paramètre sélectionné dans la liste. Sélectionnez un paramètre dans la liste pour accéder à sa description. Pressez le bouton "**Effacer**" pour effacer la description du paramètre

OEM sélectionné, et le retirer de la liste des paramètres. Attention: cette commande ne peut pas être annulée.
Le nom d'un paramètre OEM est le nom utilisé pour identifier le paramètre pendant le câblage des E/S, si le champ doit être renseigné par l'automaticien. Le nom d'un paramètre doit respecter les règles suivantes:

- la longueur du nom ne doit pas excéder **16** caractères
- le premier caractère doit être une **lettre**
- les caractères suivants doivent être des **lettres**, des **chiffres** ou '_'
- les majuscules et les minuscules ne sont pas différenciées

Le type d'un paramètre définit son **format interne**, et son **format de saisie** pendant le câblage des E/S. Voici la liste des différents formats internes:
word ..... mot non signé de 16 bits
long ..... mot non signé de 32 bits
word hexa ..... mot non signé de 16 bits
long hexa ..... mot non signé de 32 bits
boolean ..... mot non signé de 16 bits (bit 0 seul représentatif)
character ..... mot non signé de 16 bits (octet poids faible seul représentatif)
string ..... table de 16 octets contenant une chaine terminée par 0
float ..... nombre signé flottant sur 32 bits en simple précision

Voici la liste des différents formats de saisie:
word ..... mot décimal non signé
long ..... mot décimal long
word hexa ..... mot hexadécimal non signé
long hexa ..... mot hexadécimal long
boolean ..... "true" ou "false"
character ..... caractère unique
string ..... chaîne ascii (15 caractères max)
float ..... nombre signé flottant en simple précision

Le champ "**accès**" permet de dfinir le niveau d'acessibilité du paramètre pour l'utilisateur final. Si l'option "**Modifiable**" est sélectionnée, le paramètre est visible et peut être modifié pendant le câblage des E/S. La valeur du paramètre est présentée par défaut pour la saisie. Si l'option "**Caché**" est sélectionnée, le paramètre n'est pas visible pendant le câblage, et sa valeur est une constante. Le choix "**Lecture seule**" indique que le paramètre ne peut pas être modifié (valeur constante), mais qu'il reste visible pendant le câblage.

A.22.5 Fonctions et blocs fonctionnels écrits en langage IEC

ISaGRAF gère une librairie de fonctions et de blocs fonctionnels écrits en langages IEC. Les langages disponibles pour décrire une fonction ou un bloc sont le **FBD** (Function Block Diagram ou logigramme), le **LD** (Ladder Diagram ou schéma à contact), le **ST** (Structured Text) et le **IL** (Instruction list). Notez que le LD et le FBD peuvent être mêlés dans le même diagramme. Le langage **SFC** (Sequential Function Chart ou Grafcet) ne peut pas être utilisé pour écrire une fonction ou un bloc de la librairie. Le langage associé à une fonction ou un bloc est choisi quand la fonction est créée, et ne peut pas être changé par la suite.

Compilation

Les fonctions et blocs de la librairie doivent être compilés (vérifiés) avant de pouvoir être intégrés dans des projets ISaGRAF. Aucune autre opération n'est requise pour la gestion de la librairie. Les fonctions et blocs ainsi créés apparaîtront automatiquement dans les menus de sélection de l'éditeur graphique LD/FBD.

⚠ Une fonction définie en librairie peut appeler d'autres fonctions de la librairie, mais le système ISaGRAF **ne supporte pas la récursivité** dans l'appel des fonctions.

⚠ Un bloc fonctionnel écrit en langage IEC ne peut pas appeler d'autres blocs fonctionnels (décrits en "C" ou en langage IEC).

Saisie du code source

Le code source d'une fonction ou d'un bloc fonctionnel de la librairie est entré à l'aide des outils standards d'ISaGRAF: l'éditeur graphique pour les fonctions et blocs en LD ou FBD, et l'éditeur de texte pour les fonctions et blocs en ST ou IL. Référez-vous aux sections correspondantes dans ce document pour une description détaillée de ces outils. Le générateur de code d'ISaGRAF peut être directement appelé depuis les éditeurs pour compiler les fonctions ou les blocs fonctionnels de la librairie.

Dictionnaire des variables locales

On peut associer à une fonction ou un bloc fonctionnel de la librairie des variables et définitions de mots équivalents locales. Pour accéder à la déclarartion des objets locaux, utilisez les commandes du menu "**Dictionnaire**" dans la fenêtre d'édition de la fonction ou du bloc.

⚠ Une fonction ou un bloc ne peut pas accéder à une variable globale ou une instance de bloc fonctionnel. Les variables locales d'une fonction doivent être initialisées dans le corps de la fonction.

Les variables locales d'un bloc fonctionnel en langage IEC sont copiées (instanciées) pour chaque utilisation du bloc dans un projet. La valeur des variables locales pour une instance est mémorisée d'un appel à l'autre.

Définition de l'interface

Les fonctions et les blocs fonctionnels peuvent avoir jusqu'à **32** paramètres (entrées ou sorties). Une fonction a toujours un paramètre de retour et un seul, qui doit porter le même nom que la fonction, selon les conventions d'écriture du langage ST.
La liste dans la partie supérieure de la fenêtre montre les paramètres de la fonction ou du bloc, dans l'ordre défini par son prototype d'appel: d'abord les paramètres d'appel, et finalement les paramètres de retour. La partie inférieure de la fenêtre montre la description détaillée du paramètre sélectionné dans la liste. Tous les types de données d'ISaGRAF peuvent être utilisés pour un paramètre. Les paramètres de retour doivent être les derniers de la liste. La nomenclature des paramètres doit respecter les règles suivantes:

- la longueur du nom ne peut pas excéder **16** caractères

- le premier caractère doit être une **lettre**
- les suivants doivent être des **lettres**, des **chiffres** ou le souligné
- les majuscules et les minuscules ne sont pas différenciées

La commande "**Insère**" permet d'insérer un nouveau paramètre avant le paramètre sélectionné dans la liste. La commande "**Supprime**" efface le paramètre sélectionné. La commande "**Arrange**" trie automatiquement les paramètres, et positionne les paramètres de retour en fin de liste.

A.22.6 Fonctions et blocs fonctionnel en "C"

Les fonctions et blocs fonctionnels "C" sont des **fonctions informatiques** appelables depuis une application ISaGRAF, conformément aux conventions d'appel définies par les langages ST et FBD.
Les fonctions sont des traitements **synchrones**. Le cycle d'exécution ISaGRAF est suspendu pendant l'exécution d'une fonction. Les blocs fonctionnels associent des opérations et des données mémorisées cachées. Par exemple, un bloc de comptage regroupe l'opération de comptage et l'état courant du compteur. Les fonctions et les blocs fonctionnels sont généralement utilisées pour étendre les possibilités du langage, accéder aux ressources du système.

La boîte de dialogue pour la définition des paramètres d'une fonction ou d'un bloc fonctionnel est utilisée pour définir chaque paramètre d'appel ou de retour de l'élément. Utilisez les commandes du menu "**Edition**" pour définir ou modifier les paramètres de la fonction ou du bloc fonctionnel. Une fonction peut avoir jusqu'à **31** paramètres d'appel, et a toujours **un seul** paramètre de retour. Un bloc fonctionnel peut avoir jusqu'à **32** paramètres, librement répartis en paramètres d'appel ou de retour. Voici la correspondance entre les types ISaGRAF et les types du langage C:

BOOLEEN	unsigned long	mot de 32 bits non signé: 1=true / 0=false
ANALOGIQUE	long	mot signé de 32 bits
REEL	float	valeur flottante en simple précision (32 bits)
TEMPORISATION	unsigned long	mot non signé de 32 bits (unité = 1ms)
MESSAGE	char *	chaîne de caractères

Quand un message est passé à une fonction ou un bloc fonctionnel "C", il ne peut pas contenir de caractère nul (code ASCII 00). Référez-vous au manuel d'utilisation de la cible ISaGRAF pour plus d'information sur la gestion du code "C" d'une fonction ou d'un bloc fonctionnel, et sur la procédure d'intégration avec le système cible ISaGRAF.

A.22.7 Fonctions de conversion

Une fonction de conversion est une fonction "C" appelée par le gestionnaire d'E/S d'ISaGRAF à chaque fois qu'une E/S analogique caractérisée par cette conversion est acquise ou mise à jour.
La fonction donne la relation entre la **valeur électrique** de la variable (lue sur le capteur ou envoyée vers l'actionneur) et sa **valeur physique** (utilisée dans les

programmes de l'application). Le gestionnaire de librairie d'ISaGRAF permet la gestion du code source "C" d'une fonction de conversion.
Une conversion peut être utilisée pour des variables analogiques **entières** ou **réelles**. Pour cette raison, les valeurs passées dans l'interface de la fonction sont des valeurs flottantes. L'interface est la même pour toutes les fonctions de conversion. Sa définition en langage "C" est décrite dans le fichier "**TACN0DEF**".
Référez-vous au manuel d'utilisation de la cible ISaGRAF pour plus d'information sur la gestion du code "C" d'une fonction de conversion, et sur la procédure d'intégration avec le système cible ISaGRAF.

A.23 Archivage

L'utilitaire d'archivage d'ISaGRAF permet la sauvegarde des projets et des librairies ISaGRAF sur une disquette ou un répertoire de sauvegarde. L'utilitaire d'archivage est une boîte de dialogue qui est lancée depuis le Gestionnaire de Projets ou du Gestionnaire de librairies.

⚠ Pour réaliser des sauvegardes fiables, respectez les règles suivantes:

- Ecrivez le nom et la description de l'objet sur l'étiquette de la disquette
- Ne sauvez pas les projets et les librairies sur la même disquette
- Ne sauvez pas plusieurs projets sur la même disquette

A.23.1 Lancer l'archiveur

L'utilitaire d'archivage est une boîte de dialogue qui est lancée par les commandes "**Outils / Archiver**" du Gestionnaire de Projets, pour la sauvegarde ou la restitution des projets ou des données communes à tous les projets.
L'utilitaire d'archivage peut également être lancé par la commande "**Outils / Archiver**" du Gestionnaire de Librairies, pour la sauvegarde ou la restitution des éléments de la librairie active.

Projets

Un projet est toujours archivé dans son intégralité. Tous les composants du projet (codes source et objet des programmes, dictionnaire, code exécutable de l'application) sont groupés dans le même fichier d'archive. Sélectionnez l'option "**compression**" pour réduire la taille du fichier d'archive d'un projet.

Eléments de librairie

Les éléments des librairies ISaGRAF sont sauvegardés individuellement. Tous les composants d'un élément (fiche technique, définition, interface, code source...) sont sauvés ensemble dans le même fichier d'archive.

Données communes

La commande "**Outils / Archiver / Données communes**" du Gestionnaire de Projets permet de sauvegarder ou de restituer les données de portée "commune" existant dans l'atelier ISaGRAF. Cette commande ne concerne pas le contenu des librairies ISaGRAF. Voici la liste des fichiers pouvant être copiés avec cette commande:

common.eqv ..... définitions de mots équivalents communs
oem.bat ..... fichier de commandes MS-DOS de l'utilisateur

Ces fichiers sont sauvés séparément sur la disquette d'archive, dans leur forme originale. Les fichiers d'archive correspondant ne sont jamais compressés.

A.23.2 Options

Le nom de chemin utilisé pour le stockage des archives ISaGRAF est affiché en bas de la boîte de dialogue. Pressez le bouton "**Parcourir**" pour sélectionner un autre répertoire ou une autre unité de disque.

Quand l'option "**Compression**" est sélectionnée, toutes les archives créées pendant les opérations de **sauvegarde** sont compressées. Cette option est très utile pour diminuer la taille des fichiers d'archive, et stocker de grandes archives sur une seule disquette. La compression n'est généralement pas requise pour l'archivage des éléments de librairie. L'archiveur d'ISaGRAF reconnaît automatiquement le type d'une archive (compressé ou non) lors de sa restitution. L'option "**compression**" n'a donc aucun effet sur une opération de **restitution**.

A.23.3 Sauvegarder / Restituer

La liste "**Atelier**" (à gauche) montre les objets existants dans l'atelier ISaGRAF installé sur le dique dur. La liste "**Archive**" (à droite) montre les objets sauvegardés dans l'unité et le répertoire de sauvegarde.

- ***Sauvegarder***

Sélectionnez l'objet à sauver dans la liste de **gauche** (objets de l'atelier ISaGRAF) et pressez le bouton "**Sauvegarde**" pour l'archiver. Plusieurs objets peuvent être sélectionnés pour la même opération de sauvegarde. Le bouton "**Sauvegarde**" est invalide quand un élément est sélectionné dans la liste de **droite** (mode restitution).

- ***Restituer***

Sélectionnez l'objet à restituer dans la liste de **droite** (objets stockés dans l'archive) et pressez le bouton "**Restitue**" pour le copier. Plusieurs objets peuvent être sélectionnés pour la même opération de restitution. Le bouton "**Restitue**" est invalide quand un élément est sélectionné dans la liste de **gauche** (mode sauvegarde).

A.23.4 Fichiers d'archive

L'archiveur ISaGRAF crée un fichier unique pour chaque objet sauvegardé. Le fichier a le même nom que l'objet sauvegardé. Son extension dépend du type de l'objet:

.pia ..... projet
.bia ..... Carte d'E/S
.iia ..... fonction en langage IEC
.aia ..... bloc fonctionnel en langage IEC
.uia ..... Fonction C
.fia ..... Bloc fonctionnel C
.cia ..... Fonction de conversion en C
.ria ..... Configuration d'E/S
.xia ..... Equipement complexe d'E/S

A.24 Impression du dossier

Le Générateur de Dossier d'ISaGRAF permet de construire et d'imprimer un dossier complet décrivant le projet sélectionné. Il est lancé par la commande "**Projet / Imprimer**" du Gestionnaire de projet, pour construire un dossier complet. Il est également lancé par la commande "**Imprimer**" de chaque éditeur d'ISaGRAF, pour l'impression du document en cours d'édition. Toutefois, le Générateur de Dossier offre les mêmes fonctionalités dans les deux cas.
Les commandes du menu "**Edition**" permettent de définir quels éléments du projet doivent être insérés dans le dossier. Il est ainsi possible de créer la "**table des matières**" d'un document personnalisé. Toutes les informations concernant le projet (programmes, variables, options, câblage des E/S...) peuvent être insérés dans le dossier. Aucun élément provenant des autres projets ou des librairies ISaGRAF ne peuvent apparaître dans le dossier d'un projet.

La commande "**Fichier / Imprimer**" génère le dossier correspondant à la table des matières éditée et l'imprime. L'impression peut prendre plusieurs minutes, pour construire et formatter le dossier complet. Il est vivement recommandé d'attendre la fin du travail de formattage dans la fenêtre du Générateur de Dossier avant d'utiliser d'autres outils de l'atelier ISaGRAF, pour s'assurer que le dossier a pu être généré correctement. La construction du dossier peut nécessiter un grand espace sur le disque dur. Un message d'erreur sera affiché si le disque est plein. Dans ce cas, il faudra libérer de l'espace sur le disque (en détruisant d'autres fichiers) avant de relancer l'impression. Quand la commande "**Imprimer**" est lancée, une boîte de dialogue apparaît. Elle permet la saisie d'une annotation concernant l'impression demandée. Ces notes seront stockées dans un historique, et seront imprimées sur la première page de tous les dossiers futurs (y compris celui-ci).

A.24.1 Personnaliser la table des matières

Le menu "**Edition**" regroupe les commandes disponibles pour définir la "table des matières" du dossier à générer. Ces commandes permettent d'utiliser une table standard (regroupant tous les éléments du projet), de construire une table spécifique (avec seulement certains éléments) et de déplacer des rubriques dans la table et la corriger

La liste par défaut

La commande "**Table standard**" du menu "**Edition**" définit une table des matières complète, qui inclut tous les composants du projet. Voici le contenu de cette table:

- Descripteur du projet
- Arbre hiérarchique (liens entre les programmes)
- Code source pour chaque programme
- Fichier "agenda" pour chaque programme
- Définitions communes
- Définitions globales
- Définitions locales pour chaque programme
- Variables globales

- Variables locales pour chaque programme
- Paramètres de l'application
- Câblage des E/S
- Listes de variables
- Tables de conversion
- Références croisées - représentation condensée
- Références croisées - réprésentation détaillée
- Résumé des déclarations
- Carte d'adressage "réseau" des variables
- Historique des modifications

La table des matières peut être sauvegardée sur le disque à l'aide de la commande "**Fichier / Enregistrer**". Cette commande n'est pas valide quand le générateur de dossier est appelé depuis un éditeur ISaGRAF pour l'impression d'un seul document.

Couper et coller

Utilisez les commandes "**Edition / Couper**" et "**Edition / Coller**" pour déplacer des items dans la table des matières. Le Générateur de Dossier permet la sélection de plusieurs éléments dans la table affichée.

Vider la table

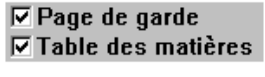
Utilisez la commande "**Edition / Effacer**" pour vider la table des matières, généralement avant de la re-construire manuellement.

Insérer des éléments dans la table

Quand la commande "**Insérer**" du menu "**Edition**" est lancée, la boîte de dialogue "**Ajouter**" apparaît. Elle permet de choisir les éléments (composants du projet) à insérer dans la table des matières.
Pour un item relatif à un programme, utilisez la boîte "**Programmes**" pour sélectionner le nom du programme. Pressez le bouton "**Ajoute**" pour insérer l'item sélectionné dans la table des matières. Le même élément ne peut pas apparaître plusieurs fois dans la table des matières.

A.24.2 Options

Utilisez les commandes du menu "**Options**" pour définir et personnaliser le format du dossier généré. D'autres options sont directement accessibles depuis la fenêtre principale du générateur de dossier:



Quand l'option "**Page de garde**" est sélectionnée, une page de garde est générée en début de dossier. Elle contient le nom du projet et l'historique des impressions. Quand cette option n'est pas validée, le premier élément à imprimer commence sur la première page.

Quand l'option "**Table des matières**" est sélectionnée, une table des matières complète est générée en fin de document.
Ces deux options ne sont pas sélectionnées par défaut quand le générateur de dossier est appelé depuis un éditeur pour l'impression d'un seul document.

Graphes SFC

L'option "**Séparer les niveaux du SFC**" permet d'imprimer, pour chaque programme SFC, d'abord le niveau 1 du SFC (dessin et commentaires), puis la programmation du niveau 2. Quand cette option n'est pas sélectionnée, les niveaux 1 et 2 d'un programme SFC apparaissent ensemble sur le document imprimé.

Mise en page

Utilisez la commande "**Mise en page**" du menu "**Options**" pour définir les paramètres principaux utilisés lors de la mise en page du dossier. Les paramètres suivants peuvent être configurés:

- **Marge gauche**: (1 ou 2 centimètres, ou pas de marge)
- **Cadre**: Quand cette option est choisie, un cadre est tracé autour de chaque folio.

Cartouche

Utilisez la commande "**Cartouche**" du menu "**Options**" pour définir le contenu du cartouche imprimé sur chaque page du dossier. Voici l'aspect standard du cartouche:

C/A	Text1	ISaGRAF - Project 'PrName'	date
	Text2		
	Text3	**User defined title**	page

La première ligne du titre (avec le nom du projet ISaGRAF), la date courante et le numéro de page sont calculés automatiquement et ne peuvent pas être modifiés. Les trois lignes de texte dans le cadre de gauche (texte1, texte2, texte3) et la seconde ligne du titre peuvent être librement définis par l'utilisateur.
Il est aussi possible de changer le logo imprimé dans le cadre de gauche. Pour utiliser un autre logo, il faut spécifier le nom et le chemin d'un fichier image (**.BMP**). L'image peut avoir des dimensions quelconques. Elle sera compressée ou étendue à l'impression, pour respecter la taille exacte du cartouche imprimé. Cliquez sur l'icone dans la boîte de dialogue pour voir apparaître le nouveau logo spécifié. Le fichier image doit exister sur le disque (avec le nom et le chemin spécifiés) quand la commande "**Imprimer**" est lancée.

Polices de caractères

Utilisez les commandes "**Police Texte**" et "**Police Titre**" du menu "**Options**" pour définir les polices de caractères à utiliser pour les titres et le texte dans le dossier généré. La taille et le style des caractères peuvent également être configurés pour le texte et les titres. La sélection des polices est faite à l'aide de la boîte de dialogue standard définie par Windows. Tous les textes (programmes littéraux, noms dans les diagrammes...) seront imprimés avec la police, la taille et le style choisi pour les textes. Seuls les titres (pour chaque composant du dossier) sont concernés par la police et le style choisis pour les titres.

Si les polices ne sont pas définies, la police standard (par défaut) de l'imprimante sera utilisée, pour les textes et les titres, avec les styles suivants:

- "Normal" pour les textes et les noms dans les diagrammes
- "Gras" pour les titres

A.25 Protection par mot de passe

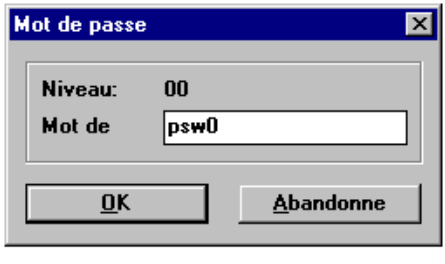
L'atelier ISaGRAF inclut un système complet de protection des données qui permet à l'utilisateur de protéger les projets et les éléments de librairie avec des mots de passe. Un élément de librairie peut être une configuration d'E/S, une carte ou un équipement complexe d'E/S, une fonction écrite en langage IEC, une fonction ou un bloc fonctionnel C, ou une fonction de conversion. Un système de protection est dédié à un projet ou à un élément de librairie, et ne peut pas être partagé entre plusieurs projets ou éléments.

Niveaux de protection

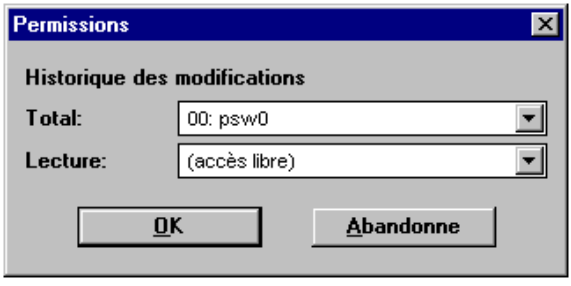
Pour un projet ou un élément de librairie, l'utilisateur peut définir jusqu'à **16** niveaux d'accès, correspondant à des mots de passe différents. Les niveaux d'accès sont ordonnés dans un système hiérarchique. Ils sont numérotés de **0** à **15**. Le plus grand niveau d'accès est numéroté **0**. Quand un utilisateur de l'atelier connait un mot de passe, il peut accéder à toutes les données correspondant aux niveaux inférieurs ou égaux à celui définit par ce mot de passe. Chaque commande ou donnée élémentaire peut être protégée par un niveau d'accès. Par exemple, la commande "Générer le code de l'application" peut être protégée indépendamment des autres commandes. Une donnée élémentaire peut être un programme, une liste d'options, une fiche technique...

Définition des mots de passe

La commande "**Mot de passe**" de l'atelier ISaGRAF permet de définir les niveaux d'accès et les mots de passe associés à un projet ou à un élément de librairie. Cette commande peut être appelée depuis le menu "**Fichiers**" du Gestionnaire de Projets d'ISaGRAF (pour un projet), ou du Gestionnaire de Librairies d' ISaGRAF (pour un élément de librairie). Aucun mot de passe n'est demandé lors du premier appel à cette fonction. Si des mots de passe sont déjà défini, il faut entrer celui de plus haut niveau pour pouvoir accéder à cette commande. Les mots de passe et protections associés à des niveaux supérieurs à celui spécifié par le mot de passe saisi ne pourront être ni visualisés ni modifiés. La commande "**Mot de passe**" permet de définir les mots de passe correspondant aux différents niveaux, et de protéger les données et commandes élémentaires avec les niveaux ainsi définis.
Les mots de passe (pour chaque niveau) sont saisis en cliquant deux fois sur la ligne d'un mot de passe dans la liste supérieure. La fenêtre suivante est ouverte:



La liste dans la partie inférieure montre les données et commandes qui peuvent être protégées, et les niveaux de protection assignés aux accès "en lecture seule" ou en "contrôle total". Assigner une protection à l'accès en lecture seule vous permet d'empêcher d'autres utilisateurs de visualiser ou d'imprimer un document.
Il faut cliquer deux fois sur une ligne de la liste inférieure pour changer les permissions assignées à un élément. La fenêtre suivante est affichée:



Les deux permissions peuvent être assignées à "accès libre" ou à un niveau de protection défini par un mot de passe. Le niveau choisi pour le contrôle total ne peut pas être moins prioritaire que le niveau choisi pour la lecture seule.
Notez que certains documents naturellement visibles dans ISaGRAF, tels que les descripteurs de projet, ne peuvent pas avoir un mode "lecture seule" protégé.

Accès aux données protégées

Aucun nom d'utilisateur ni mot de passe n'est demandé quand l'atelier ISaGRAF est lancé. Par la suite, chaque fois que l'utilisateur tente d'accéder à une donnée ou commande protégée par un mot de passe, le mot de passe est demandé dans une boîte de dialogue.
Si l'utilisateur entre le mot de passe demandé (ou un mot de passe correspondant à un niveau supérieur), il peut continuer normalement. A chaque fois qu'un mot de passe est entré, il est mémorisé; et aucun mot de passe ne sera demandé par la suite pour accéder aux items protégés par des niveaux égaux ou inférieurs. Les mots de passe saisis sont mémorisés lors de chaque appel à un nouvel outil ISaGRAF (éditeurs, debugger...) mais sont perdus quand la dernière fenêtre ISaGRAF est fermée. Les mots de passe saisis lors de la gestion des projets ne sont pas retenus pour accéder au gestionnaire de librairies et à l'archiveur (et inversement). Si un mauvais mot de passe est saisi, la fonction ou donnée ne peut pas être accédée.

Liens avec l'archiveur

Lors de la sauvegarde d'un projet ou d'un élément de librairie sur disquette, la protection de la commande nommée "**Sauvegarde sur archive**" est testée. Ceci correspond au système de protection de l'objet dans l'atelier (sur le disque dur). Aucun test n'est réalisé quant à la protection de la version archivée si elle existe déjà. La commande "**Sauvegarder**" de l'archiveur d' ISaGRAF enregistre le système de protection avec les données sauvegardées.
Lors de la restitution d'un objet existant déjà dans l'atelier (sur le disque dur), la protection de la commande nommée "**Ecraser avec une archive**" est testée. Ceci

correspond au système de protection de l'objet dans l'atelier (sur le disque dur). Aucun test n'est réalisé quant à la protection de la version archivée. Si la commande est validée, le système de protection restitué de l'archive écrase celui existant sur le disque.

Protections individuelles pour les variables et les voies d'E/S

L'atelier ISaGRAF offre un système de protection basé sur un ensemble de mots de passe hiérarchisés. La déclaration des variables et le câblage des E/S peuvent être globalement verrouillés par un mot de passe. De plus, ISaGRAF permet d'installer une protection individuelle sur chaque variable déclarée et sur chaque voie d'E/S. Ceci présume que:
- des mots de passes sont déjà définis dans le système de protection de l'application (utilisez la commande "**Projet / Mot de passe**" du gestionnaire de projets) et que des niveaux de protections valides sont disponibles pour les protections.
- vous utilisez des niveaux de protection individuelle plus prioritaires que les niveaux utilisés pour la protection globale des déclarations et du câblage.

Quand une variable ou une voie d'E/S a une protection individuelle, un icone spécial est affiché à côté de son nom dans le dictionnaire ou l'éditeur de câblage.
Utilisez les commandes "**Protéger**" et "**Retirer la protection**" du menu "**Edition**" du dictionnaire ou de l'éditeur de câblage pour installer ou retirer une protection individuelle sur la variable ou la voie sélectionnée. Ces deux commandes vous demandent de saisir un mot de passe valide correspondant à un niveau de protection correctement défini. Par la suite, chaque fois que vous voudrez changer la définition d'une variable protégée ou le câblage d'une voie protégée, vous devrez entrer un mot de passe suffisant.

Attention: si une variable ou une voie est protégée avec un niveau, et que le mot de passe correspondant est supprimé dans le système de protection, et s'il n'existe pas de mot de passe plus prioritaire, la variable ou la voie ne pourra plus être modifiée jusqu'à la définition de nouveaux mots de passe de niveau suffisant.

A.26 Techniques avancées

Ce chapitre contient des informations supplémentaires sur l'atelier ISaGRAF et le système cible. Il est dédié aux utilisateurs déjà familiarisés avec les outils et les méthodes d'ISaGRAF.

A.26.1 Au sujet des outils d'ISaGRAF

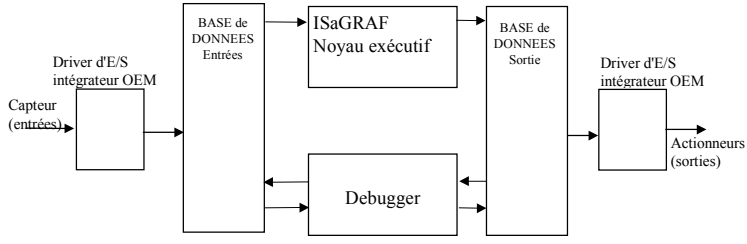
Lors de l'utilisation des outils d'édition de l'atelier ISaGRAF, il est possible de presser le **bouton droit** de la souris pour ouvrir un menu qui regroupe les principales commandes d'édition. Ce menu est ouvert à l'emplacement du curseur. Ceci réduit en particulier les "allers et retours" de la souris vers la barre de menu pendant les opérations couper / coller.
L'atelier ISaGRAF permet **l'exécution multiple** des outils. Bien qu'un outil ne puisse pas être ouvert plusieurs fois pour traiter le même objet, il est possible d'ouvrir plusieurs fenêtres pour l'édition de plusieurs objets en parallèle.
D'autres commandes permettent d'afficher la description des boutons graphiques dans les barres d'outils. Cliquer deux fois sur une zone inutilisée de la barre affiche les commandes (boutons et texte) sous forme d'un menu. Laisser le curseur de la souris sur un bouton graphique un certain temps affiche le texte correspondant à la commande associée.

A.26.2 Entrées/sorties verrouillées et virtuelles

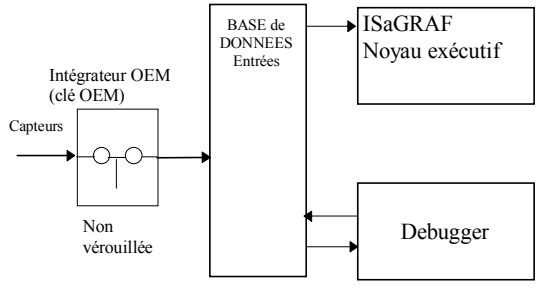
La définition d'une carte **virtuelle** pendant le câblage des E/S entraîne une déconnexion logique des variables câblées sur la carte et des dispositifs d'E/S correspondants. Quand une carte est définie comme virtuelle, les opérations du noyau ISaGRAF ne sont pas modifiées. La seule différence est que les capteurs ne sont pas acquis et que les actionneurs ne sont pas mis à jour. Les variables d'E/S de l'application peuvent être forcées ou lues avec le debugger. L'attribut "**Virtuel**" concerne une carte complète. Il est programmé pendant le câblage des E/S, **avant** la génération du code de l'application. L'attribut "**Virtuel**" est une spécification **statique**, qui est mémorisée quand l'application est arrêtée et relancée.
Une autre possibilité consiste à **verrouiller** des variables d'E/S. Le verrouillage des variables d'E/S est réservé à la maintenance. C'est la déconnexion logique du dispositif d'entrée ou de sortie associé à une variable d'E/S de l'application. Le verrouillage ou le déverrouillage d'une variable est réalisé avec le debugger. C'est une opération **dynamique**, qui ne sera pas mémorisée au redémarrage de l'application. Le **verrouillage** ne concerne qu'**une** variable (une voie d'E/S). Voici le résumé des fonctions de contrôle des E/S:

	Attribut "Virtuel"	*Verrouillage*
outil de sélection	câblage des E/S	debugger
définition	statique	dynamique
mode de sélection	par carte	par variable
application	validation et tests	maintenance

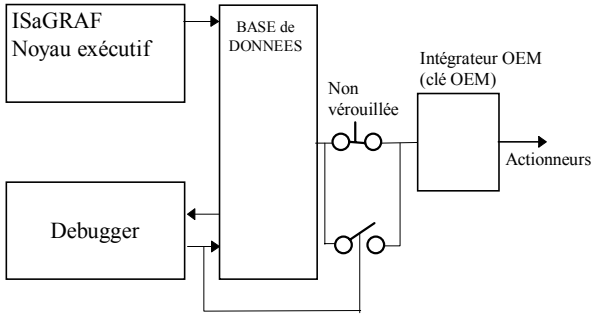
Le diagramme suivant montre le flux de données des E/S entre les différents modules d'ISaGRAF:



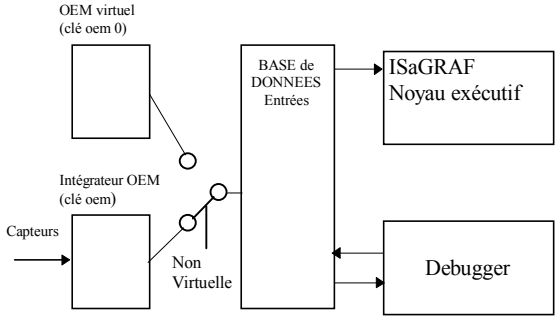
Quand une variable d'entrée est verrouillée, les accès à la base de données des entrées par le noyau ISaGRAF sont inchangés, mais la base de données et le capteur sont déconnectés. Les valeurs d'entrée pour l'application sont forcées par le debugger:



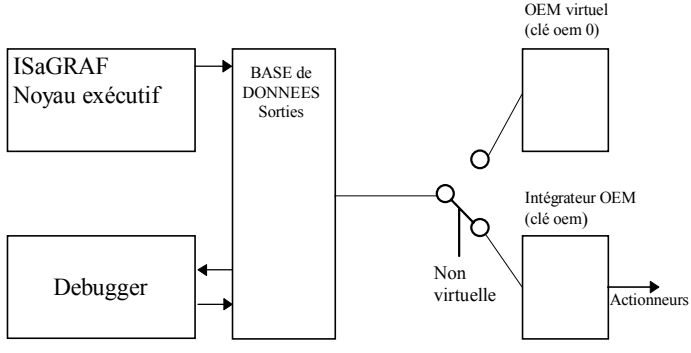
Quand une variable de sortie est verrouillée, le noyau exécutif et le driver de sortie sont déconnectés. Dans ce cas, l'accès à l'actionneur est encore possible avec le debugger d'ISaGRAF:



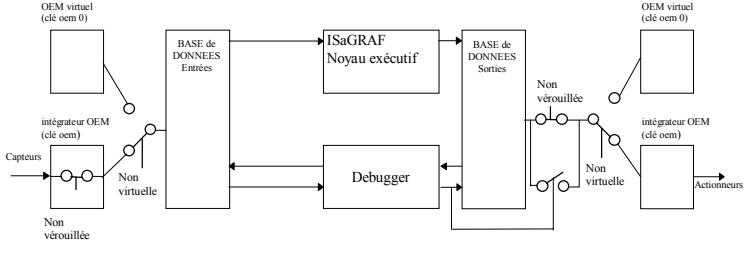
La définition d'une carte d'entrée virtuelle suit les mêmes règles que le verrouillage des entrées. La base de données des entrées et les capteurs correspondants sont déconnectés. Un driver d'E/S virtuel remplace le driver réel.



La définition d'une carte virtuelle suit les mêmes règles pour une carte d'entrée ou une carte de sortie. Pour une carte de sortie, le noyau exécutif ISaGRAF (application) continue la mise à jour de la base de données, mais celle-ci est déconnectée des actionneurs associés. Un driver d'E/S virtuel remplace le driver réel.



Pour résumer toutes les possibilités:



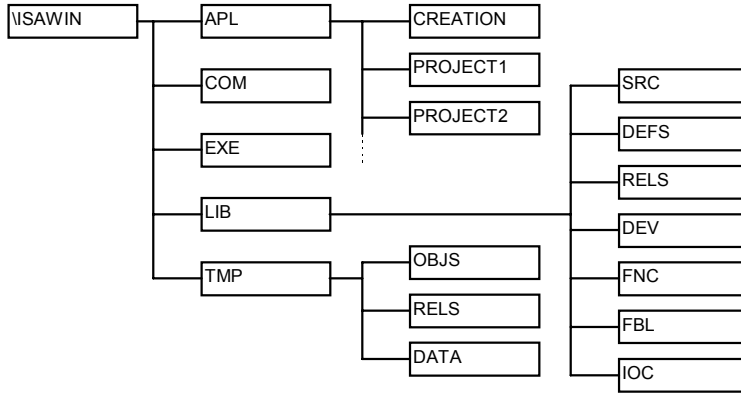
A.26.3 Validation de la liaison avec la cible

La plupart des problèmes dus à une mauvaise communication entre l'atelier ISaGRAF et l'automate cible sont annoncés par le message "**déconnecté**" dans la fenêtre du debugger. Avant tout diagnostic, il faut valider la communication, quand **aucune application n'est active** dans l'automate. On peut ainsi valider la communication série, en dehors de tout problème dû à l'exécution de l'application.
Le langage "C", utilisé pour la description des fonctions de conversion et des procédures, permet l'accès direct aux ressources du système cible. Une erreur de programmation dans un tel composant logiciel peut provoquer de graves problèmes système ou un comportement incorrecte du système ISaGRAF. Ces problèmes peuvent également apparaître lors de l'implémentation de nouveaux pilotes d'E/S, en utilisant l'option **IO Toolkit** d'ISaGRAF. Une erreur système peut survenir si une carte d'E/S est implantée à une adresse mémoire incorrecte ou invalide.
Le tableau suivant résume les principaux diagnostics des erreurs de communication entre l'atelier et la cible ISaGRAF:

statut	*contexte*	*diagnostic*
"déconnecté" (avant téléchargement)		- cible arrêtée ou hors tension - câble absent ou invalide - paramètres de comm. invalides - ISaGRAF mal installé sur la cible
"déconnecté" (après téléchargement)	démarrage en cycle à cycle	- mauvaise configuration des E/S - erreur système
	démarrage en temps réel	- mauvaise configuration des E/S - erreur système (due à la programmation "C")
"pas d'application"		- application non téléchargée - application arrêtée (à cause de la programmation "C") - confusion Intel/Motorola - version de cible incorrecte

A.26.4 Les répertoires ISaGRAF

L'atelier ISaGRAF travaille sur une architecture de répertoires dédiée. Le nom du répertoire racine pour cette architecture est spécifié par l'utilisateur lors de l'installation d'ISaGRAF. Le nom par défaut pour le répertoire racine est "**ISAWIN**". Voici les répertoires créés par le programme d'installation:



Les sous répertoires standard d'ISaGRAF sont:

Répertoire	*Contenu*
APL:	répertoire de base pour les projets ISaGRAF: chaque projet correspond à un sous répertoire qui contient toutes les données du projet D'autres répertoires peuvent exister pour d'autres groupes de projets. L'installateur d'ISaGRAF crée le répertoire **SMP** où sont installés les projets d'exemple.
COM	objets de portée "commune", utilisables par tous les projets
EXE	programmes d'ISaGRAF et fichiers d'aide
LIB	librairies ISaGRAF: - liste des éléments, - interface, paramètres pour chaque élément - fiches techniques
LIB\IOC	code source des configurations d'E/S
LIB\FNC	code source des fonctions en langages IEC
LIB\FBL	code source des blocs fonctionnels en langages IEC
LIB\SRC	code source des éléments en langage "C"
LIB\DEFS	fichiers de définitions en langage "C"
LIB\RELS	code relogeable des éléments en langage "C"
LIB\DEV	fichier de commande pour le développement "C" des librairies: makefile, fichier d'édition de liens...
TMP	fichiers temporaires. les sous-répertoires sous TMP sont réservés au générateur de code ISaGRAF et ne doivent jamais être détruits.

Les répertoires peuvent être placés ailleurs sur le disque. Si une architecture non standard est utilisée, le chemin des sous répertoires doit être décrit dans la section "**WS001**" du fichier d'initialisation "**ISA.ini**", dans le sous-répertoire **EXE** de ISaGRAF. Voici les entrées de la section "**WS001**":

Isa ..... chemin du répertoire racine d'ISaGRAF
IsaExe ..... chemin du répertoire des exécutables ISaGRAF
IsaApl ..... chemin du répertoire racine des projets ISaGRAF

IsaTmp..... chemin du répertoire pour les fichiers temporaires
IsaSrc..... chemin du répertoire pour les fichiers de code source C
IsaDefs..... chemin du répertoire pour les fichiers de définitions C

Notez que si vous changez l'entrée IsaTmp, vous devez créer les sous-répertoires OBJS, RELS et DATA dans le nouveau répertoire. L'exemple suivant montre les entrées de la section "**WS001**" pour une architecture standard:

```
;fichier c:\ISAWIN\EXE\ISA.ini

[WS001]
Isa=c:\isawin
IsaExe=c:\isawin\exe
IsaApl=c:\isawin\apl
IsaTmp=c:\isawin\tmp
IsaSrc=c:\isawin\lib\src
IsaDefs=c:\isawin\lib\defs
```

Quand vous voulez ajouter des fonctions ou des blocs fonctionnels "C" à la cible ISaGRAF, le répertoire **\ISAWIN\LIB\DEV** peut être utilisé pour les fichiers de développement: fichiers de commandes, makefiles, maps, etc... Le répertoire **\ISAWIN\LIB\RELS** peut être utilisé pour les fichiers objets générés lors de la compilation "C", et les fichiers librairies nécessaires à l'édition de lien du noyau ISaGRAF.

A.26.5 Table des symboles

Chaque objet d'une application ISaGRAF est identifié par un nom (saisi pendant la déclaration des variables) et une **adresse virtuelle** interne, calculée par le générateur de code. L'adresse virtuelle ne doit pas être confondue avec l'**adresse réseau** saisie pendant la déclaration d'une variable. Les adresses virtuelles sont utilisées dans les opérations de communication entre le noyau ISaGRAF et le debugger ou une autre tâche "C" utilisateur. Quand le générateur de code d'ISaGRAF est lancé, il génère un fichier ascii qui donne la correspondance entre les noms et les adresses virtuelles pour tous les objets (variables, programmes, étapes...) de l'application. Ce fichier peut être facilement interprété par un programme utilisateur, pour accéder aux informations symboliques de la base de données statique du noyau ISaGRAF. Ce fichier est nommé "**APPLI.TST**" et il est situé dans le répertoire du projet ISaGRAF: "**\ISAWIN\APL\nompro**" (nompro est le nom du projet). Cette section présente le format détaillé du fichier "**APPLI.TST**". Les notations suivantes sont utilisées pour la description du fichier:
VA..... adresse virtuelle
ATTR ..... attribut d'une variable
USP ..... fonction "C"

Voici les valeurs que peut prendre le champ "**attributs**" qui décrit les attributs d'une variable:
+X..... variable interne
+C..... variable interne accessible en lecture seulement
+I..... variable d'entrée
+O..... variable de sortie

Tous les nombres, à l'exception des adresses virtuelles, sont exprimés en décimal. Les adresses virtuelles (**VA**) sont exprimées en hexadécimal sur quatre chiffres, précédés du caractère "**!**". Par exemple:
`123` ..... est une valeur décimale
`!A003` ..... est une adresse virtuelle exprimée en hexadécimal

Voici le format global du fichier "**APPLI.TST**". Le fichier est structuré comme une liste de **blocs**. Un bloc est une suite d'**enregistrements**. Chaque enregistrement est décrit sur une ligne de texte. Chaque bloc commence par un en-tête décrit sur une ligne.

```
Bloc de début
Blocs de description
Bloc de fin
```

Voici le format général d'un bloc:

```
@ <nom_bloc> <arguments>
#enregistrement...
#enregistrement...
...
```

Voici la structure du premier bloc du fichier, qui regroupe des informations globales sur l'application:

```
@ISA_SYMBOLS,<appli_crc>
#NAME,<appli_name>,<version>
#DATE,<creation_date>
#SIZE,G=<nbprg>,S=<nbstep>,T=<nbtra>,L=0,P=<nbpro>,V=<nbvar>
#COMMENT,cj international
```

appli_crc ..... checksum du fichier de symboles
appli_name ..... nom de l'application
version ..... numéro de version de l'atelier ISaGRAF
creation_date.... date de génération de l'application
nbprg..... nombre de programmes
nbstep ..... nombre d'étapes SFC
nbtra..... nombre de transitions SFC
nbpro..... nombre de procédures "C" utilisées
nbvar ..... nombre total de variables d'E/S

Voici la structure du dernier bloc, qui indique la fin du fichier:

```
@END_SYMBOLS
```

Voici la structure du bloc utilisé pour décrire les programmes de l'application:

```
@PROGRAMS,<nbprg>
#<va>,<name>
#...
```

nbprg..... nombre de programmes définis dans ce bloc
va..... adresse virtuelle du programme
name..... nom du programme

Voici la structure du bloc utilisé pour décrire les étapes SFC de l'application. Une adresse virtuelle de pseudo-étape est définie pour chacun des programmes non SFC:

```
@STEPS,<nbsteps>
#<va>,<name>,<father>
#...
```

nbsteps ..... nombre d'étapes définies dans ce bloc
va..... adresse virtuelle de l'étape
name..... nom de l'étape
father..... adresse virtuelle du programme

Voici la structure du bloc utilisé pour décrire les transitions SFC de l'application:

```
@TRANSITIONS,<nbtrans>
#<va>,<name>,<father>
#...
```

nbtrans..... nombre de transitions définies dans ce bloc
va..... adresse virtuelle de la transition
name..... nom de la transition
father..... adresse virtuelle du programme

Voici la structure du bloc utilisé pour décrire les variables booléennes de l'application:

```
@BOOLEANS,<nb_boo>
#<va>,<name>,<attr>,<program>,<eq_false>,<eq_true>
#...
```

et si l'index de la variable excède 4095:

```
X#(1.<varno>),<name>,<attr>,<program>,<eq_false>,<eq_true>
```

nb_boo..... nombre de variables dans ce bloc
va..... adresse virtuelle de la variable
varno ..... index de la variable (non masqué par le type)
name..... nom de la variable
attr..... attribut de la variable
program ..... adresse virtuelle du programme père
ou "**!0000**" pour une variable globale
eq_false..... libellé équivalent à false
eq_true..... libellé équivalent à true

Voici la structure du bloc utilisé pour décrire les variables analogiques de l'application:

```
@ANALOGS,<nb_ana>
#<va>,<name>,<attr>,<program>,<format>,<unit>
#...
```

et si l'index de la variable excède 4095:

```
X#(2.<varno>),<name>,<attr>,<program>,<format>,<unit>
```

nb_ana ..... nombre de variables dans ce bloc
va..... adresse virtuelle de la variable
varno ..... index de la variable (non masqué par le type)
name..... nom de la variable
attr ..... attribut de la variable
program ..... adresse virtuelle du programme père
ou "**!0000**" pour une variable globale
format..... = "**I**" pour une variable entière
= "**F**" pour une variable réelle
unit ..... libellé d'unité

Voici la structure du bloc utilisé pour décrire les variables temporisations de l'application:

```
@TIMERS,<nb_tmr>
#<va>,<name>,<attr>,<program>
#...
```

et si l'index de la variable excède 4095:

```
X#(3.<varno>),<name>,<attr>,<program>
```

nb_tmr..... nombre de variables dans ce bloc
va..... adresse virtuelle de la variable
varno ..... index de la variable (non masqué par le type)
name..... nom de la variable
attr ..... attribut de la variable (toujours +X: interne)
program ..... adresse virtuelle du programme père
ou "**!0000**" pour une variable globale

Voici la structure du bloc utilisé pour décrire les variables messages de l'application:

```
@MESSAGES,<nb_msg>
#<va>,<name>,<attr>,<program>,<max_len>
#...
```

et si l'index de la variable excède 4095:

```
X#(4.<varno>),<name>,<attr>,<program>,<max_len>
```

nb_msg ..... nombre de variables dans ce bloc
va..... adresse virtuelle de la variable
varno ..... index de la variable (non masqué par le type)

name..... nom de la variable
attr ..... attribut de la variable
program ..... adresse virtuelle du programme père
ou "**!0000**" pour une variable globale
max_len..... longueur maximum (capacité déclarée du message)

Voici la structure du bloc utilisé pour décrire les fonctions "C" utilisées dans l'application:

```
@USP,<nb_usp>
#<va>,<name>
#...
```

nb_usp ..... nombre de fonctions dans ce bloc
va..... adresse virtuelle de la fonction
name..... nom de la fonction

Voici la structure du bloc utilisé pour décrire les instances de blocs fonctionnels "C" utilisées dans l'application:

```
@FBINSTANCES,<nb_fb>
#<va>,<inst_name>,<fb_name>
#...
```

nb_fb ..... nombre d'instances d'un bloc fonctionnel C dans ce bloc
va..... adresse virtuelle de l'instance du bloc fonctionnel
inst_name ..... nom de l'instance du bloc fonctionnel
fb_name ..... nom du bloc fonctionnel de référence

A.26.6 Limites de l'atelier ISaGRAF "LARGE" (WDL)

Voici quelques limites numériques concernant les objets traités par l'atelier ISaGRAF. Bien sûr, d'autres limites pratiques sont imposées par la configuration du matériel utilisé (mémoire disponible, espace disque), et les capacités du système cible ISaGRAF (mémoire, ressources matérielles et logicielles disponibles...). Les tableaux suivants donnent des limites purement théoriques qui ne pourront en aucun cas être dépassées.

- **Concernant un projet**

Objet	*Maximum*	*Notes*
Programmes	255	pour l'ensemble des programmes principaux, fils, et sous-programmes
Niveaux de hiérarchie	20	

Le nombre de projets installés dans l'atelier n'est limité que par l'espace disque disponible.

- **Concernant les noms**

Nom pour:	*Maximum*	*Notes*
Projet	8 char	
Programme	8 char	
Variable	16 char	+ 60 caractères de commentaire

Nom de mot équivalent	16 char	
Equivalence	255 char	+ 60 caractères de commentaire
Table de conversion	16 char	
Liste de variables	16 char	
fonction / bloc fonc	8 char	fonctions ou blocs fonctionnels en C ou en langage CEI
fonction paramètre	16 char	fonctions ou blocs fonctionnels en C ou en langage CEI
Carte d'E/S	8 char	
Configuration d'E/S	8 char	
Paramètre OEM	16 char	
Fonction de conversion	8 char	

- **Edition (pour un programme)**

Objet	*Maximum*	*Notes*
Lignes SFC	600	
Colonnes SFC	20	
Etapes SFC	4095	pour l'application entière, groupant les étapes, étapes initiales, étapes de début et de fin
Transitions SFC	4095	pour l'application entière
LD/FBD	200 cols	
	2000 lignes	taille de l'espace d'édition en pas de la grille
Quick LD	pas limité	les limites sont imposées par la capacité du PC
Etiquettes IL	251	dans le même programme IL
Edition de texte	40Kbytes	ou moins selon la configuration du système

- **Concernant le dictionnaire (pour un projet)**

Objet	*Maximum*	*Notes*
Variables booléennes	65535	
Variables analogiques	65535	variables entières et réelles
Temporisations	65535	
Variables message	65535	
Mots équivalents	4095	dans la même liste (même portée)
Mots équivalents	255	utilisés dans le même programme
Tables de conversion	127	utilisées dans l'application
Points dans une table	32	définis dans la même table

Les limites données pour les variables booléennes, analogiques et messages regroupent les variables internes, d'entrée et de sortie. Il prend également en compte toutes les variables temporaires allouées par les compilateurs. Le nombre de variables éditées ensemble dans le dictionnaire (même type, même portée) ne peut pas excéder 16000. Cette limite peut être inférieure à 16000 selon la configuration de votre PC. L'application ne peut pas fonctionner sur une cible 3.21 ou antérieure si le nombre total de variable pour un type excède 4095. Le lien "MODBUS" utilisant les adresses réseau ne peut pas être utilisé si le nombre de variables d'un type excède 4095.

- **Concernant le câblage des E/S**

Objet	*Maximum*	*Notes*

Cartes d'E/S	256	définies dans la même application (cartes ou E/S complexes)
Voies d'E/S	128	sur la même carte

Le nombre de carte est la somme des cartes simples et des éléments de chaque équipement complexe.

Concernant les librairies

Objet	*Maximum*	*Notes*
Fonctions (lang. CEI)	255	installées ensemble dans la librairie
B.Fs (lang. CEI)	255	installés ensemble dans la librairie
Fonctions C	255	installées ensemble dans la librairie
Blocs fonctionnels C	255	installés ensemble dans la librairie
Instances de B.F	4095	pour le même type de bloc dans la même application
Paramètres fonction	31	en entrée - fonctions en C ou en langage CEI
Paramètres B.F	32	répartis en entrée et sortie (au moins une sortie est requise)
Fonction de conversion	128	installées ensemble dans la librairie
Configurations d'E/S	255	installées ensemble dans la librairie
Cartes d'E/S	255	installées ensemble dans la librairie
Equipt. complexes d'E/S	255	installés ensemble dans la librairie
Paramètres OEM	16	

B. Référence des langages

B.1 Architecture du projet

Une application ISaGRAF est divisée en plusieurs unités de programmation appelés **programmes**. Les programmes d'une application sont organisés en une architecture hiérarchisée. Les programmes peuvent être décrits avec les langages graphiques ou textuels **SFC**, **FC (Flow Chart)**, **FBD**, **LD, ST** ou **IL**.

B.1.1 Programmes

Un **programme** est une unité logique de programmation qui décrit des opérations entre les **variables** de l'application. Les programmes décrivent des opérations **séquentielles** ou **cycliques**. Les programmes cycliques sont exécutés systématiquement à chaque cycle ISaGRAF. L'exécution des programmes séquentiels respecte les règles d'évolution des langages **SFC** et **FC**.
Les programmes sont organisés dans un système hiérarchique arborescent. Les programmes placés au sommet, **programmes principaux**, de la hiérarchie sont activés par le système. Les **sous-programmes** (de niveau inférieur dans la hiérarchie) sont activés par leur programme père. La description d'un programme peut être réalisée à l'aide des langages graphiques ou textuels disponibles:

- **Sequential Function Chart** (SFC) ou GRAFCET pour les traitements séquentiels
- **Flow Chart** (FC) pour la structuration des opérations
- **Function Block Diagram** (FBD) ou Logigramme pour les traitements cycliques
- **Ladder Diagram** (LD) ou schéma à relais pour les traitements cycliques booléens
- **Structured Text** (ST) pour les traitements cycliques
- **Intruction List** (IL) pour les traitements cycliques de bas niveau

Plusieurs langages ne peuvent pas être mixés dans le même programme, à l'exception des langages LD et FBD qui peuvent être mêlés dans le même diagramme.

B.1.2 Opérations cycliques et séquentielles

La **hiérarchie** des programmes est organisée en quatre principales **sections** ou groupes:

DEBUT	programmes exécutés au début de chaque cycle
SEQUENTIEL	programmes exécutés en fonction des règles SFC ou FC
FIN	programmes exécutés à la fin de chaque cycle
FONCTIONS	ensemble de sous-programmes non dédiés
BLOCS FONCTIONNELS	ensemble de blocs fonctionnels

Les programmes des sections "**Début**" et "**Fin**" décrivent des opérations **cycliques**. Ils sont indépendants de l'état du procédé. Les programmes de la section "**Sequentiel**" décrivent des opérations séquentielles, où l'état du procédé est explicitement représenté dans la programmation, selon les conventions **SFC** ou **FC**.
Les programmes de la section "**Fonctions**" sont des sous-programmes qui peuvent être appelés par tous les autres programmes du projet. Un programme de la section "**Fonction**" peut appeler un programme de la même section.

Les programmes de la section Blocs fonctionnels peuvent être appelé par tous les autres programmes du projet. Un programme de la section Blocs fonctionnels peut appeler des programmes de la section fonction, mais pas d'autres blocs fonctionnels.
Les programmes principaux de la section "**Début**" sont exécutés systématiquement au début de chaque cycle. Les programmes principaux de la section "**Fin**" sont exécutés systématiquement à la fin de chaque cycle. Les programmes principaux de la section "**Séquentiel**" sont activés au lancement de l'application, et exécutés conformément aux règles d'évolution du **SFC** ou du **FC**.
Les programmes principaux et fils de la section séquentielle doivent être décrits avec le langage **SFC** ou le langage **FC**. Les programmes principaux des sections cycliques ne peuvent pas être décrits avec le langage **SFC** ni avec le langage **FC**. Chaque programme de chaque section peut avoir un ou plusieurs sous-programmes. Chaque programme **SFC** de la section séquentielle peut contrôler un ou plusieurs programmes **SFC** fils. Chaque programme **FC** de la section séquentielle peut contrôler un ou plusieurs sous-programmes **FC**.
Les sous-programmes (non **FC**) ne peuvent pas être décrits avec les langage **SFC** ou **FC**.
Les programmes de la section "**Début**" sont typiquement utilisés pour décrire les opérations préliminaires effectuées sur les variables d'entrée, pour construire des données de plus haut niveau, qui seront reprises dans la section "**Séquentiel**". Les programmes de la section "**Fin**" sont typiquement utilisés pour les opérations de contrôle et de mise en forme des données de haut niveau (ordres) fabriqués par la section "**Séquentiel**" avant la mise à jour des actionneurs.

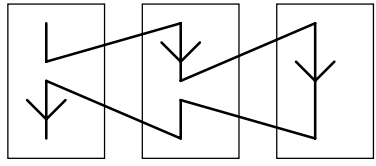
B.1.3 Fils SFC et sous-programmes FC

Chaque programme **SFC** de la section séquentielle peut contrôler d'autres programmes SFC, appelés ses **programmes fils**. Un **programme SFC fils** est un programme exécuté en parallèle, qui peut être lancé, tué, figé ou relancé par son programme père. Le programme père et le programme fils doivent tous deux être décrits avec le langage **SFC**. Un programme fils peut avoir des variables et définitions locales.
Quand un programme père lance un programme **SFC** fils, il place un **jeton** (activation) dans chacune de ses étapes initiales. Une telle commande peut être décrite par l'énoncé **GSTART**.
Quand un programme père tue un programme **SFC** fils, il enlève tous les jetons existant dans ses étapes. Une telle commande peut être décrite par l'énoncé **GKILL**.
Quand un programme père fige un programme **SFC** fils, il suspend son exécution. Le programme figé peut ensuite être relancé par l'énoncé **GRST**.
Tout programme **FC** de la section séquentielle peut contrôler des sous-programmes **FC**. L'exécution d'un programme père **FC** est suspendue (le flux est bloqué) pendant toute l'exécution du sous-programme **FC**, c'est-à-dire jusqu'à ce que le flux dans le sous-programme rencontre le symbole END. Il ne peut en aucun cas y avoir d'exécution simultanée d'un programme **FC** et d'un de ses sous-programmes **FC**.

B.1.4 Fonctions et sous-programmes

L'exécution d'un sous-programme ou d'un fonction est demandée par son programme père. L'exécution du père est suspendue jusqu'à la fin du déroulement du sous-programme ou de la fonction:

progr. **sous-programmes**



Chaque programme de chaque section peut avoir un ou plusieurs sous-programmes. Un sous-programme ne peut être activé que par un seul programme père. Un sous-programme peut avoir des variables et des définitions locales. Tous les langages à l'exception du **SFC** et du **FC** peuvent être utilisés pour décrire un sous-programme. Les programmes de la section "**Fonctions**" sont des sous-programmes qui peuvent être appelés par n'importe quel autre programme du projet. Contrairement aux autres sous-programmes, ils ne sont pas dédiés à un programme père unique. Un programme de la section "**Fonction**" peut appeler un autre sous-programme de cette section.

Attention: Le système ISaGRAF ne supporte pas la **récursivité** dans les appels de fonctions. Une erreur surviendra à l'exécution si un programme de la section "**Fonctions**" est appelé par lui-même ou un de ses descendants.

Attention: Une fonction ou un sous-programme ne "stocke" pas la valeur de ses variables locales. Une fonction ou un sous-programme n'est pas instancié et ne peut donc pas appelé de blocs fonctionnels.

L'interface d'un sous-programme doit être défini explicitement, avec un **type** et un **nom** pour chacun de ses paramètres d'appel et son paramètre de retour. Afin de respecter les conventions d'écriture du langage **ST**, le paramètre de retour d'un sous-programme doit porter le même nom que le sous-programme. La table ci-après montre comment forcer le paramètre de retour dans le corps d'un sous-programme, dans les différents langages:

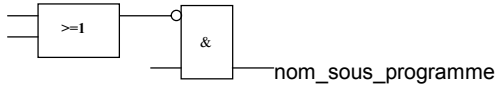
ST: assigner une valeur au paramètre de retour en utilisant son nom (le même nom que le sous-programme):

```
nom_sous_programme:= <expression>;
```

IL: la valeur du résultat courant (registre IL) à la fin de la séquence est stocké dans le paramètre de retour:

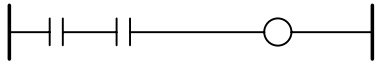
```
LD    10
ADD   20   (* valeur du paramètre de retour = 30 *)
```

FBD: affecter le paramètre de retour en utilisant son nom:



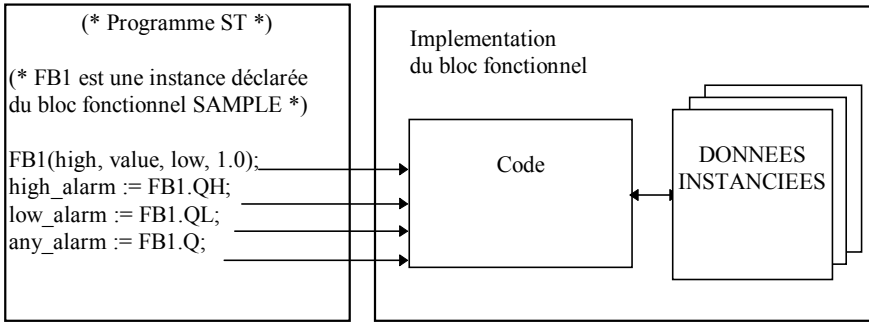
LD: utiliser un relais ayant le nom du sous-programme:

nom_sous_programme



B.1.5 Blocs fonctionnels

Un bloc fonctionnel peut être programmé dans les langages: LD, FBD, ST ou IL. Les blocs fonctionnels sont instanciés. Ce qui signifie que les variables locales d'un bloc fonctionnel sont copiées pour chaque instance. Lorsqu'un programme appelle un bloc, il appelle en fait l'instance du bloc: le même code est appelé mais les données utilisées sont celles qui ont été allouées pour l'instance. Les valeurs des variables de l'instance sont conservées d'un cycle à l'autre.



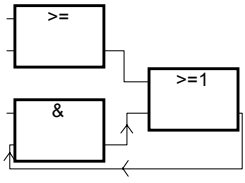
Attention:
- Un bloc fonctionnel écrit dans l'un des langages de la CEI ne peuvent pas appeler d'autres blocs fonctionnels: le mécanisme d'instanciation gère uniquement les variables locales du bloc en question. Voici la liste des blocs fonctionnels standard que vous ne pouvez pas utiliser à l'intérieur d'un bloc fonctionnel écrit en langage CEI:
SR, RS, R_Trig, F_Trig, SEMA, CTU, CTD, CTUD, TON, TOF, TP, CMP, StackInt, AVERAGE, HYSTER, LIM_ALRM, INTEGRAL, DERIVATE, BLINK, SIG_GEN

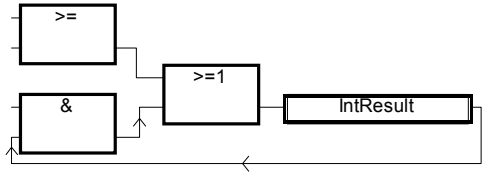
- Pour les mêmes raisons, vous ne pouvez pas utiliser des contacts ou des relais de type Positif ou Négatif, ou des relais de type Set ou Reset.

- Les fonctions TSTART et TSTOP qui gère le démarrage ou l'arrêt de temporisation ne peuvent pas être utilisées si les cibles de version 3.0x. Mais peuvent être utilisées avec les cibles dont la version est au moins 3.20.

- Lorsque vous écrivez des boucles dans votre bloc fonctionnel, vous devez utiliser une variable locale avant la boucle, comme dans l'exemple ci-dessous:

Ceci ne marche pas: Ceci marche:





B.1.6 Langages

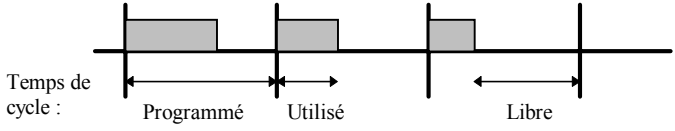
Un programme peut être décrit avec un des langages graphiques ou littéraux suivants:

- **Sequential Function Chart** (SFC) ou GRAFCET pour les traitements séquentiels
- **Flow Chart** (FC) pour la structuration des opérations
- **Function Block Diagram** (FBD) ou Logigramme pour les traitements cycliques
- **Ladder Diagram** (LD) ou schéma à relais pour les traitements cycliques booléens
- **Structured Text** (ST) pour les traitements cycliques
- **Intruction List** (IL) pour les traitements cycliques de bas niveau

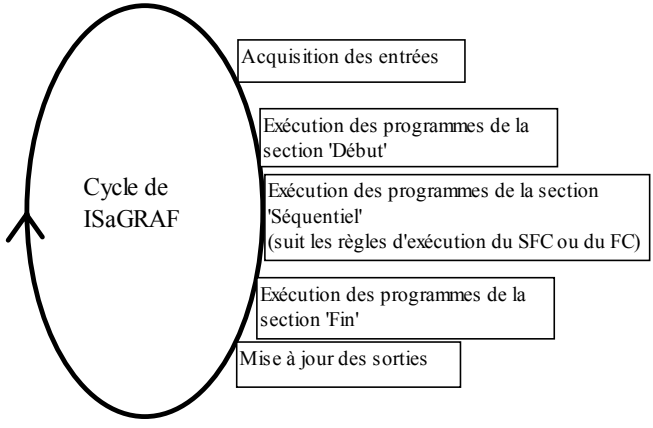
Le même programme ne peut pas utiliser plusieurs langages. Le langage utilisé pour décrire un programme est choisi quand le programme est créé. Toutefois, les éléments graphiques des langages LD et FBD peuvent être mêlés dans le même diagramme.

B.1.7 Règles d'exécution

ISaGRAF est un système **synchrone**. Toutes les opérations sont échantillonnées. La durée d'un échantillon est appelée le **temps de cycle**:



Les opérations réalisées dans un cycle sont:



Ce système assure:
- qu'une variable **d'entrée** n'évolue pas pendant un cycle
- que les **sorties** ne sont mises à jour qu'une fois dans le cycle
- le partage fiable de variables **globales** entre les programmes
- l'estimation du temps de réponse de l'application complète

B.2 Objets communs à tous les langages

Ce chapitre présente les **objets** de la base de données de programmation d'ISaGRAF. Ces objets sont manipulés par les programmes écrits en **SFC**, **FC**, **FBD**, **LD**, **ST** ou **IL**.

B.2.1 Types de base

Toute expression, constante ou variable utilisée dans un programme doit être caractérisée par un **type**. La cohérence des types doit être respectée dans les réseaux graphiques et les énoncés textuels. Voici les types de base prédéfinis par ISaGRAF:
- **BOOLEEN**: valeur logique (vrai ou faux)
- **ANALOGIQUE**: valeur continue entière ou réelle
- **TEMPORISATION**: valeur de temps
- **MESSAGE**: chaîne de caractères

Note: les temporisations contiennent des valeurs inférieures à un jour, et ne peuvent pas être utilisées pour stocker des dates.

B.2.2 Expressions constantes

Une **expression constante** est toujours relative à un type. La même notation ne peut pas être utilisée pour représenter des expressions constantes de type différent.

B.2.2.1 Expressions constantes booléennes

Il existe seulement deux expressions constantes de type **booléen**:
- **TRUE** est équivalent à la valeur entière 1
- **FALSE** est équivalent à la valeur entière 0

Les minuscules et les majuscules ne sont pas différenciées pour l'écriture des mots clé "True" et "False".

B.2.2.2 Expressions constantes analogiques entières

Une expression constante **analogique entière** représente un nombre signé entre **-2147483647** et **+2147483647**. Les expressions constantes entières peuvent être exprimées dans l'une ou l'autre des **bases** présentées dans le tableau suivant. L'expression constante doit commencer par un **préfixe** qui identifie la base utilisée.

base	préfixe	exemple
DECIMAL	(aucun)	-908
HEXADECIMAL	"16#"	16#1A2B3C4D
OCTAL	"8#"	8#1756402
BINAIRE	"2#"	2#1101_0001_0101_1101

Le caractère souligné ('**_**') peut être librement utilisé pour séparer des groupes de chiffres. Il n'a aucune signification sémantique, mais augmente la lisibilité des expressions longues.

B.2.2.3 Expressions constantes analogiques réelles

Les expressions **analogiques réelles** peuvent utiliser une représentation **décimale** ou **scientifique**. Le **point décimal** ('**.**') sépare la partie entière de la partie décimale. Il doit obligatoirement être présent car il permet la différenciation des expressions constantes entières et réelles. La représentation scientifique utilise la lettre '**E**' ou '**F**' pour séparer la **mantisse** de l'**exposant**. L'exposant d'une expression constante scientifique est un entier signé compris entre **-37** et **+37**. Voici des exemples d'expressions constantes réelles:

3.14159	**-1.0E+12**
+1.0	**1.0F-15**
-789.56	**+1.0E-37**

L'expression "**123**" ne représente pas une valeur réelle. Sa notation correcte est "**123.0**".

B.2.2.4 Expressions constantes temporelles

Les expressions constantes **temporelles** représentent des valeurs de temps entre **0 seconde** et **23h59m59s999ms**. La plus petite **unité** autorisée est la milli-seconde. Voici les unités standard de temps utilisées dans les expressions constantes temporelles:

- Heure: la lettre "**h**" doit suivre le nombre d'heures
- Minute: la lettre "**m**" doit suivre le nombre de minutes
- Seconde: la lettre "**s**" doit suivre le nombre de secondes
- Milli-seconde: les lettres "**ms**" doivent suivre le nombre de milli-secondes

L'expression doit commencer par le préfixe "**T#**" ou "**TIME#**". Les préfixes et les lettres d'unité peuvent être écrits en minuscule ou en majuscule. Certaines unités peuvent ne pas apparaître. Voici des exemples d'expressions constantes temporelles:

T#1H451MS	1 heure et 451 milli-secondes
time#1H3M	1 heure et 3 minutes

L'expression "0" ne représente pas une expression constante temporelle, mais analogique.

B.2.2.5 Expressions constantes messages

Les expressions constantes de type **message** représentent **des chaînes de caractères**. Les caractères de la chaîne doivent être précédés et suivis par une apostrophe. Par exemple:

'CECI EST UN MESSAGE'

Attention: le caractère apostrophe ne peut pas être utilisé à l'intérieur de l'expression. Une expression constante de type message doit être exprimée sur une seule ligne de programme, et ne doit jamais excéder 255 caractères.
Une chaîne vide est représentée par deux apostrophes, sans espace ou caractère de tabulation entre elles:

'' (* ceci est une chaîne vide *)

Le caractère spécial dollar ('**$**'), suivi par d'autres caractères spéciaux, peut être utilisé pour insérer des caractères non imprimables dans une chaîne de caractères:

Séquence	Signification	Ascii	Exemple
$$	caractère '$'	16#24	'Le $$ est coté...'
$'	apostrophe	16#27	'Entrez $'O$' pour YES'
$L	ligne suivante	16#0a	'ligne $L suivante'
$R	retour chariot	16#0d	' jour $R Bon'
$N	fin de ligne	16#0d0a	'Ceci est une ligne$N'
$P	fin de page	16#0c	'fin page $P début page'
$T	tabulation	16#09	'nom$Ttaille$Tdate'
$hh (*)	tout caractère	16#hh	'ABCD = $41$42$43$44'

(*) "**hh**" est le code ascii du caractère, exprimé sur deux chiffres en hexadécimal.

B.2.3 Variables

Les variables peuvent être **LOCALES** à un programme, ou **GLOBALES**. Les variables locales ne peuvent être utilisées que par un seul programme. Les variables globales peuvent être utilisées par tous les programmes de l'application. Le nom d'une variable doit respecter les règles suivantes:

- la longueur du nom ne doit pas excéder **16** caractères
- le premier caractère doit être une **lettre**
- les caractères suivants doivent être des **lettres**, des **chiffres** ou '**_**'

B.2.3.1 Mots clés réservés

Voici la liste des **mots clés** réservés. Ces identificateurs ne doivent pas être utilisés en tant pour nommer un programme, une variable, une fonction ou un bloc fonctionnel.

A	ANA, ABS, ACOS, ADD, ANA, AND, AND_MASK, ANDN, ARRAY, ASIN, AT, ATAN,
B	BCD_TO_BOOL, BCD_TO_INT, BCD_TO_REAL, BCD_TO_STRING, BCD_TO_TIME, BOO, BOOL, BOOL_TO_BCD, BOOL_TO_INT, BOOL_TO_REAL, BOOL_TO_STRING, BOOL_TO_TIME, BY, BYTE,
C	CAL, CALC, CALCN, CALN, CALNC, CASE, CONCAT, CONSTANT, COS,
D	DATE, DATE_AND_TIME, DELETE, DINT, DIV, DO, DT, DWORD,
E	ELSE, ELSIF, EN, END_CASE, END_FOR, END_FUNCTION, END_IF, END_PROGRAM, END_REPEAT, END_RESSOURCE, END_STRUCT, END_TYPE, END_VAR, END_WHILE, ENO, EQ, EXIT, EXP, EXPT,
F	FALSE, FEDGE, FIND, FOR, FUNCTION,
G	GE, GFREEZE, GKILL, GRST, GSTART, GSTATUS, GT,
I	IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING, INT_TO_TIME,
J	JMP, JMPC, JMPCN, JMPN, JMPNC,
L	LD, LDN, LE, LEFT, LEN, LIMIT, LINT, LN, LOG, LREAL, LT, LWORD,

M MAX, MID, MIN, MOD, MOVE, MSG, MUL, MUX,

N NE, NOT,

O OF, ON, OPERATE, OR, OR_MASK, ORN,

P PROGRAM

R R, REDGE, READ_ONLY, READ_WRITE, REAL, REAL_TO_BCD, REAL_TO_BOOL, REAL_TO_INT, REAL_TO_STRING, REAL_TO_TIME, REDGE, REPEAT, REPLACE, RESSOURCE, RET, RETAIN, RETC, RETCN, RETN, RETNC, RETURN, RIGHT, ROL, ROR,

S S, SEL, SHL, SHR, SIN, SINT, SQRT, ST, STN, STRING, STRING_TO_BCD, STRING_TO_BOOL, STRING_TO_INT, STRING_TO_REAL, STRING_TO_TIME, STRUCT, SUB, SYS_ERR_READ, SYS_ERR_TEST, SYS_INITALL, SYS_INITANA, SYS_INITBOO, SYS_INITTMR, SYS_RESTALL, SYS_RESTANA, SYS_RESTBOO, SYS_RESTTMR, SYS_SAVALL, SYS_SAVANA, SYS_SAVBOO, SYS_SAVTMR, SYS_TALLOWED, SYS_TCURRENT, SYS_TMAXIMUM, SYS_TOVERFLOW, SYS_TRESET, SYS_TWRITE, SYSTEM,

T TAN, TASK, THEN, TIME, TIME_OF_DAY, TIME_TO_BCD, TIME_TO_BOOL, TIME_TO_INT, TIME_TO_REAL, TIME_TO_STRING, TMR, TO, TOD, TRUE, TSTART, TSTOP, TYPE,

U UDINT, UINT, ULINT, UNTIL, USINT,

V VAR, VAR_ACCESS, VAR_EXTERNAL, VAR_GLOBAL, VAR_IN_OUT, VAR_INPUT, ,VAR_OUTPUT,

W WHILE, WITH, WORD,

X XOR, XOR_MASK, XORN

Tous les mots clé commençant par le caractère souligné ('_') sont des mots clés internes et ne doivent pas être utilisés dans des instructions textuelles.

B.2.3.2 Variables représentées directement

ISaGRAF permet l'utilisation des 'variables représentées directement' dans le code des programmes pour accéder à une voie d'E/S libre. Les voies libres sont celles qui ne sont pas liées à une variable d'E/S déclarée. L'identificateur d'une variable représentée directement commence toujours pas le caractère "**%**" .
Voici la nomenclature à suivre pour accéder à une variable représentée directement pour une voie de carte simple. "**s**" est le numéro d'emplacement de la carte (slot). "**c**" est le numéro de la voie (channel).

`%IXs.c` voie libre pour une carte d'entrée booléenne
`%IDs.c` voie libre pour une carte d'entrée analogique entière
`%ISs.c` voie libre pour une carte d'entrée de type message
`%QXs.c` voie libre pour une carte de sortie booléenne
`%QDs.c` voie libre pour une carte de sortie analogique entière
`%QSs.c` voie libre pour une carte de sortie de type message

Voici la nomenclature à suivre pour accéder à d'une variable représentée directement pour une voie d'un équipement complexe. "**s**" est le numéro d'emplacement de l'équipement (slot). "**b**" est le numéro de la carte dans l'équipement (board). "**c**" est le numéro de la voie (channel).

`%IXs.b.c`	voie libre pour une carte d'entrée booléenne
`%IDs.b.c`	voie libre pour une carte d'entrée analogique entière
`%ISs.b.c`	voie libre pour une carte d'entrée de type message
`%QXs.b.c`	voie libre pour une carte de sortie booléenne
`%QDs.b.c`	voie libre pour une carte de sortie analogique entière
`%QSs.b.c`	voie libre pour une carte de sortie de type message

Exemples:

`%QX1.6`	$6^{ième}$ voie de la carte #1 (sortie booléenne)
`%ID2.1.7`	$7^{ième}$ voie de la carte #1 de l'équipement #2 (entrée entière)

Une variable représentée directement ne peut pas avoir le type "**réel**".

B.2.3.3 Variables booléennes

Booléen signifie **logique**. Ces variables ne peuvent prendre que les valeurs **TRUE** ou **FALSE**. Les variables booléennes sont typiquement utilisées dans les expressions de type booléen. Une variable booléenne peut avoir un des **attributs** suivants:

Interne:	variable mémoire mise à jour par le programme
Constant:	variable mémoire avec une valeur initiale, autorisée en lecture seulement
Entrée:	variable connectée à un capteur (rafraîchie par le système)
Sortie:	variable connectée à un actionneur

Attention: Lors de la déclaration d'une variable booléenne, les libellés définis pour remplacer les valeurs 'true' et 'false' sont destinés à la mise au point. Ils ne peuvent pas être utilisés dans la programmation, tant qu'ils ne font pas l'objet de **définitions** de mots équivalents.

B.2.3.4 Variables analogiques

Analogique signifie **continu**. Ces variables prennent des valeurs signées entières ou réelles (flottantes). Voici les formats disponibles pour une variable analogique:

Entier:	entier signé sur 32 bits: de **-2147483647** à **+2147483647**
Réel:	valeur flottante sur 32 bits - standard IEEE (simple précision) 1 bit de signe + 23 bits de mantisse + 8 bits d'exposant

L'exposant d'une variable analogique REELLE doit être compris entre **-37** et **+37**. Une variable analogique peut avoir un des **attributs** suivants:

Interne:	variable mémoire mise à jour par le programme
Constant:	variable mémoire avec une valeur initiale, autorisée en lecture seulement
Entrée:	variable connectée à un capteur (rafraîchie par le système)
Sortie:	variable connectée à un actionneur

Note: Quand une variable réelle est connectée à un dispositif d'E/S, le pilote d'E/S correspondant traite l'équivalent entier de sa valeur.

Attention: Les variables entières et réelles ne peuvent pas être mélangées dans la même expression analogique.

B.2.3.5 Variables temporisation

Une **temporisation** est un chronomètre, (ou horloge ou compteur de temps). Ces variables prennent des valeurs temporelles, comprises entre **0s** et **23h59m59s999ms**, et ne peuvent jamais être négatives. Les variables temporisations sont stockées sur 32 bits. Leur représentation interne indique un nombre entier de milli-secondes.
Une variable temporisation peut avoir un des **attributs** suivants:

Interne: variable mémoire mise à jour par le programme
Constant: variable mémoire avec une valeur initiale, autorisée en lecture seulement

Attention: Les variables temporisations ne peuvent pas prendre les attributs ENTREE ou SORTIE.

Les variables temporisations peuvent être automatiquement rafraîchies par le système ISaGRAF. Quand une temporisation est **active**, sa valeur est automatiquement incrémentée en accord avec l'horloge du système. Les énoncés suivants peuvent être utilisés pour contrôler une temporisation:

TSTART: lance le rafraîchissement automatique d'une temporisation
TSTOP: arrête le rafraîchissement automatique d'une temporisation

B.2.3.6 Variables messages

Les variables **messages** contiennent des **chaînes de caractères**. La longueur de la chaîne peut changer en fonction des opérations réalisées. La longueur d'un message ne peut jamais excéder sa capacité (longueur maximale) spécifiée pendant sa déclaration. La capacité d'un message ne peut pas excéder 255 caractères. Une variable message peut avoir un des **attributs** suivants:

Interne: variable mémoire mise à jour par le programme
Constant: variable mémoire avec une valeur initiale, autorisée en lecture seulement
Entrée: variable connectée à un capteur (rafraîchie par le système)
Sortie: variable connectée à un actionneur

Les variables messages peuvent contenir tous les caractères de la table ascii (codes de **0** à **255**). Le caractère nul (code 0) peut exister à l'intérieur d'une chaîne. Certaines procédures "C" de la librairie standard d'ISaGRAF ne peuvent pas traiter correctement des messages contenant des caractères nuls.

B.2.4 Commentaires

Des blocs de **commentaires** peuvent être librement insérés dans le code source des programmes textuels **ST** et **IL**. Un bloc de commentaire doit commencer par les caractères "**(***" et se terminer par les caractères "***)**". Les commentaires peuvent être insérés partout

dans un programme **ST**, et peuvent être exprimés sur plusieurs lignes. Voici des exemples de commentaires:

```
counter:= ivalue;   (* assignation compteur principal *)
(* voici un commentaire exprimé
sur deux lignes *)
c:= counter (* compteur principal *) + base_value + 1;
```

Les blocs de commentaires ne peuvent pas être imbriqués. Les caractères "**(***" sont interdits dans un bloc de commentaires.

Attention: Le langage IL n'accepte un bloc de commentaire que comme dernier composant d'une ligne d'instruction.

B.2.5 Définition de mots équivalents

Le système ISaGRAF permet de redéfinir des expressions constantes, des libellés **équivalents** à "true" ou "false", des mots clé du langage ou des expressions **ST** complexes. Pour ceci, il faut affecter un **identificateur** à l'expression correspondante. Par exemple:

OUI	signifie	**TRUE**
PI	signifie	**3.14159**
OK	signifie	**(mode_auto AND NOT (alarme))**

Quand une équivalence est définie, son **identificateur** peut être utilisé pour remplacer l'expression dans le code source d'un programme. Voici un exemple de programme ST utilisant les équivalences définies ci-avant:

```
If OK Then
        angle:= PI / 2.0;
        isdone:= OUI;
End_if;
```

Les définitions peuvent être **LOCALES** à un programme, ou **GLOBALES**, ou **COMMUNES**.
Les définitions locales ne peuvent être utilisées que dans un seul programme.
Les définitions globales peuvent être utilisées dans tous les programmes de l'application.
Les définitions communes peuvent être utilisées dans tous les programmes de toutes les applications.
Notez que les définitions communes peuvent être stockées séparément en utilisant l'utilitaire d'archivage.

Attention: Quand le même identificateur est utilisé pour plusieurs équivalences, il remplace seulement la dernière expression définie. Par exemple:

Définition:	**OPEN**	équivalent à	**FALSE**
	OPEN	équivalent à	**TRUE**
signifie:	**OPEN**	équivalent à	**TRUE**

La nomenclature des définitions doit respecter les règles suivantes:

- la longueur du nom ne doit pas excéder **16** caractères
- le premier caractère doit être une **lettre**
- les caractères suivants doivent être des **lettres**, des **chiffres**, ou '**_**'

Attention:
Un mot équivalent ne peut pas utiliser un mot équivalent dans sa définition. Par exemple, vous ne pouvez pas avoir:

PI	équivalent à	**3.14159**
PI2	équivalent à	**PI*2**

Il faut écrire l'équivalence en utilisant uniquement des constantes des variables ou des opérations:

PI2	équivalent à	**6.28318**

B.3 Langage SFC

Le langage **SFC** (**Sequential Function Chart**) ou GRAFCET est un langage **graphique** utilisé pour décrire les opérations **séquentielles**. Le procédé est représenté comme une suite connue d'**étapes** (états stables), reliées entre elles par des **transitions**. Une **condition booléenne** est attachée à chaque transition. Les **actions** dans les étapes sont décrites avec les langages **ST**, **IL**, **LD** ou **FDB**.

B.3.1 Format du graphe SFC

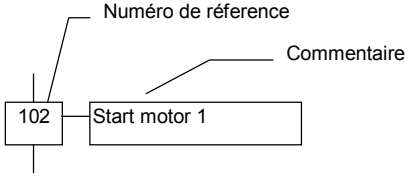
Un programme SFC est un réseau graphique d'**étapes** et de **transitions**, reliées par des **liaisons orientées**. Les liens de connexion multiples sont représentés par des divergences et des convergences. Certaines parties du graphe peuvent être isolées, et représentées dans le graphe principal par un seul symbole, nommé **macro-étape**. Les principales **règles graphiques** du langage SFC sont:
- une étape ne peut pas être suivie d'une autre étape
- une transition ne peut pas être suivie d'une autre transition

B.3.2 Composants de base du graphe SFC

Voici les composants de base (symboles graphiques) du graphe SFC: étapes et étapes initiales, transitions, liaisons orientées et renvois à une étape.

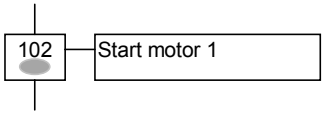
B.3.2.1 Etapes et étapes initiales

Une **étape** est représentée par un **carré**. Chaque étape est **référencée** par un **numéro**. Ce numéro est inscrit dans le cadre de l'étape. Une description générale de l'étape est écrite dans un rectangle relié au symbole de l'étape. Cette description est un **commentaire libre** (pas de langage de programmation). Toutes ces informations constituent le **niveau 1** de l'étape:

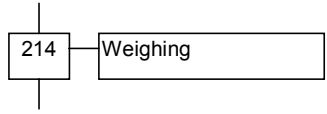


A l'exécution, un **jeton** indique si l'étape est **active**:

Etape active:

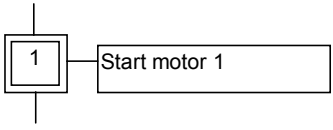


Etape inactive:



La **situation initiale** d'un programme SFC est représentée par ses **étapes initiales**. Le symbole d'une étape initiale a un **cadre double**. Un jeton est automatiquement posé dans chaque étape initiale quand le programme est lancé.

Etape initiale:



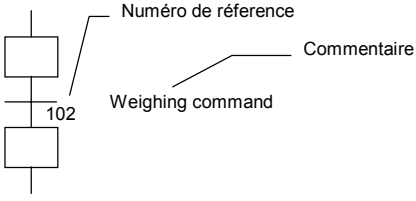
Un programme SFC doit contenir **au moins une** étape initiale. Voici les attributs d'une étape. Ces champs peuvent être testés depuis tous les langages:

GSnnn.x.....**activité de l'étape** (valeur booléenne)
GSnnn.t.....**durée d'activité** de l'étape (valeur temporelle)

(**nnn** est le numéro de référence de l'étape)

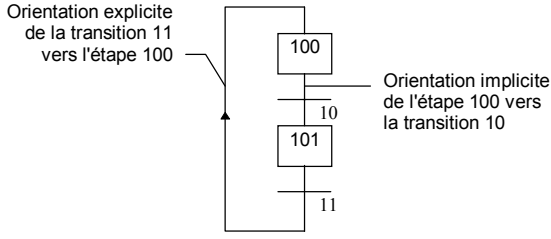
B.3.2.2 Transitions

Une **transition** est représentée par une barre horizontale qui croise un arc de liaison. Chaque transition est **référencée** par un **numéro**. Ce numéro est inscrit à côté du symbole de la transition. Une description générale de la transition est inscrite à la droite de son symbole. C'est un **commentaire libre** (pas de langage de programmation). Toutes ces informations constituent le **niveau 1** de la transition:



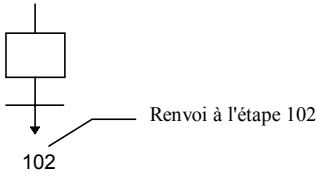
B.3.2.3 Liaisons orientées

De simple lignes sont utilisées pour représenter les **liaisons** orientées entre les étapes et les transitions. Quand l'orientation n'est pas explicitement donnée, le lien est orienté du haut vers le bas.

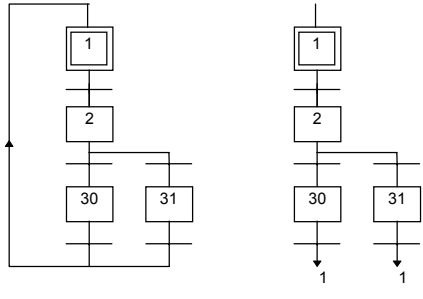


B.3.2.4 Renvoi à une étape

Un symbole de **renvoi** peut être utilisé pour représenter un arc de liaison d'une transition vers une étape, sans tracer le lien. Le symbole du renvoi doit être référencé avec le **numéro de référence** de l'étape de destination:



Vous ne pouvez pas utiliser de symbole de renvoi vers une transition. Exemple de renvois - les graphes suivants sont équivalents:

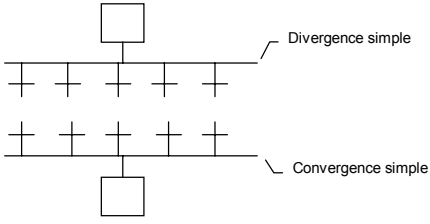


B.3.3 Divergences et convergences

Les **divergences** sont des liaisons multiples depuis un symbole SFC (étape ou transition) vers plusieurs autres symboles SFC. Les **convergences** sont des liaisons multiples depuis plusieurs symboles SFC vers un même symbole. On distingue deux types de divergences et de convergences: simples ou doubles.

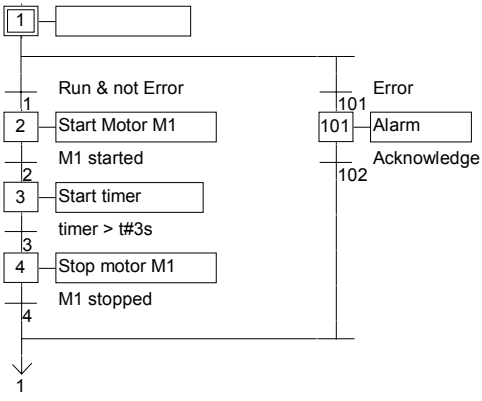
B.3.3.1 Divergences simples

Une divergence simple est une liaison multiple depuis une étape vers plusieurs transitions. Elle représente plusieurs possibilités dans le séquencement du procédé. Une convergence simple est une liaison multiple depuis plusieurs transitions vers la même étape. Une convergence simple est généralement utilisée pour regrouper les branches ouvertes sur une divergence simple. Les divergences et convergences simples sont représentées par des lignes horizontales simples.



Attention: Les conditions attachées aux différentes transitions qui débutent les branches de la divergence **ne sont pas implicitement exclusives**. Il faut détailler l'exclusivité explicitement dans chacune de ces conditions pour s'assurer qu'un seul jeton SFC sera créé, dans l'une des branches de la divergence, quand celle-ci sera franchie. Voici un exemple de divergence et de convergence simples:

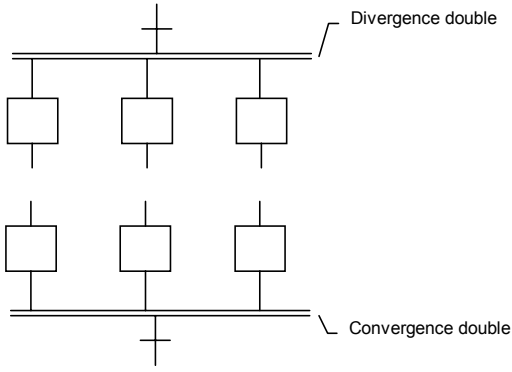
(* Programme SFC avec divergence et convergence simples *)



B.3.3.2 Divergences doubles

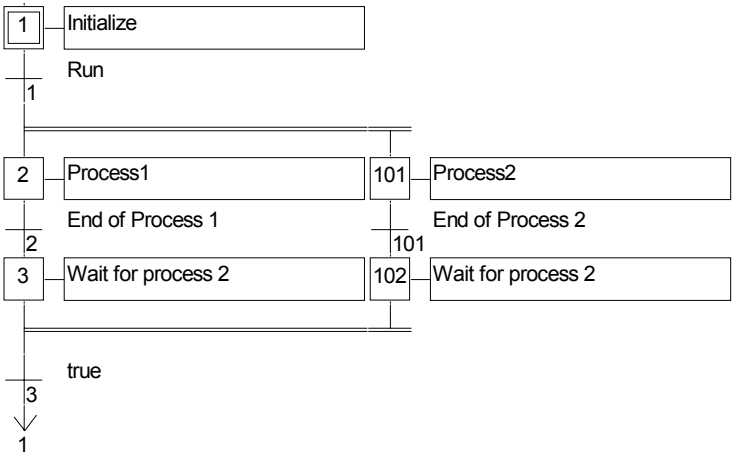
Une divergence double est une liaison multiple depuis une transition vers plusieurs étapes. Elle représente généralement des opérations parallèles dans le séquencement du procédé. Une convergence double est une liaison multiple depuis plusieurs étapes vers la même transition. Une convergence double est généralement utilisée pour regrouper les branches

ouvertes sur une divergence double. Les divergences et convergences doubles sont représentées par des lignes horizontales doubles.



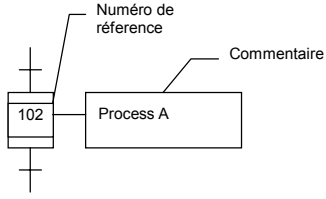
Voici un exemple de divergence et de convergence doubles:

(* Programme SFC avec divergence et convergence doubles *)



B.3.4 Macro-étapes

Une **macro-étape** est une représentation par un symbole unique d'un groupe unique d'étapes et de transitions. Le corps de la macro-étape est décrit séparément, ailleurs dans le même programme SFC. Elle apparaît comme un symbole unique dans le graphe principal. Voici le symbole utilisé pour représenter une macro-étape:



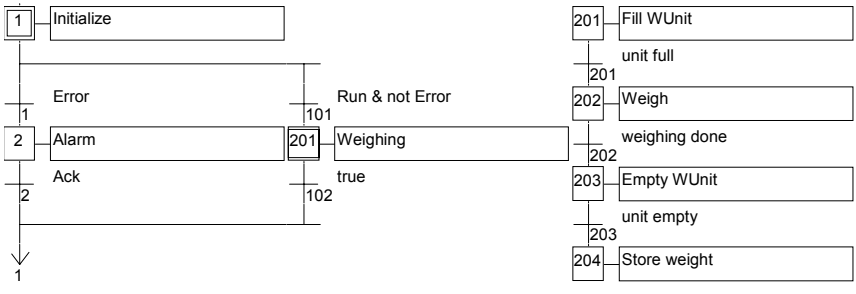
Le **numéro de référence** écrit dans le symbole de la macro-étape est le numéro de référence de la première étape du corps de la macro. Le corps d'une macro-étape doit commencer par une **étape de début** et se terminer par une **étape de fin**. Le graphe doit être connexe. Une étape de début n'a pas de liaison en amont (pas de transition précédente). Une étape de fin n'a pas de liaison en aval (pas de transition suivante). Un symbole de macro-étape peut apparaître dans le corps d'une autre macro-étape.

Attention: Puisqu'une macro-étape représente un groupe **unique** d'étapes et de transitions, la même macro-étape ne peut pas apparaître plusieurs fois dans le même programme SFC.

Voici un exemple de macro-étape:

(* Programme SFC incluant une macro-étape *)

(* Schéma principal *) **(* Corps de la macro-étape *)**



B.3.5 Actions dans les étapes

Le **niveau 2** d'une étape SFC est la description détaillée des **actions** exécutées **quand l'étape est active**. Cette description est réalisée à l'aide des **structures littérales du SFC**, et des autres langages, tels que le **ST** (Structured Text). Voici les différents types d'action:
- Actions booléennes
- Actions impulsionnelles programmées en ST
- Actions normales programmées en ST
- Actions SFC

Plusieurs actions (de même type ou de type différent) peuvent être décrites dans la même étape. Les structures suivantes permettent d'intégrer des traitements décrits dans d'autres langages:

- Appel de sous-programme
- Insertion de code IL (Instruction List)

B.3.5.1 Actions booléennes

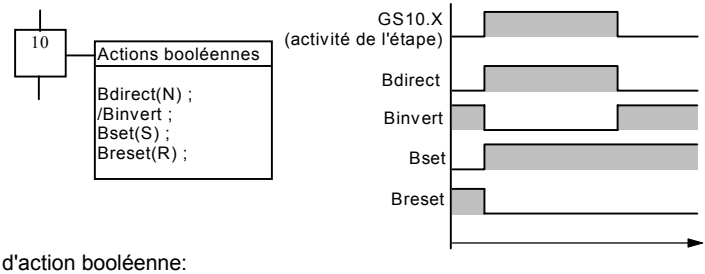
Une **action booléenne** consiste à assigner à une variable booléenne le signal d'activité d'une étape (drapeau d'étape). La variable booléenne doit être interne ou de sortie. Elle est forcée à chaque fois que le signal d'activité de l'étape change d'état. Voici la syntaxe des actions booléennes non mémorisées:

<variable_bool> (N);	copie le signal d'activité de l'étape dans la variable
<variable_bool> ;	même effet (l'attribut N est optionnel)
/ <variable_bool> ;	copie l'inverse logique du signal d'activité de l'étape dans la variable

D'autres types d'actions consistent à forcer la variable (à faux ou à vrai) quand l'étape devient active. Voici la syntaxe de ces actions:

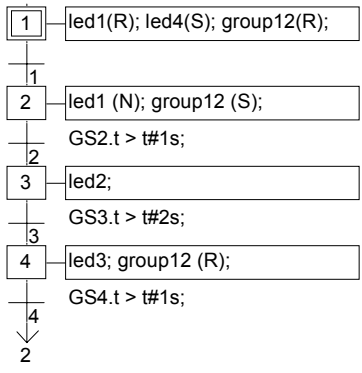
<variable_bool> (S) ;	force la variable à TRUE quand le signal d'activité de l'étape prend l'état TRUE
<variable_bool> (R) ;	force la variable à FALSE quand le signal d'activité de l'étape prend l'état TRUE

La variable booléenne doit être INTERNE ou de SORTIE. La programmation SFC suivante représente le comportement indiqué dans le chronogramme:



Exemple d'action booléenne:

(* Programme SFC avec actions booléennes *)

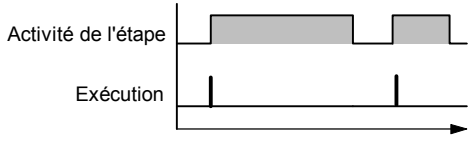


B.3.5.2 Actions impulsionnelles

Une **action impulsionnelle** est une liste d'instructions ST ou IL, exécutée **une seule fois** quand l'étape **devient active**. Les instructions sont écrites selon la syntaxe SFC suivante:

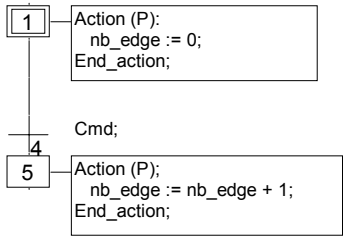
```
ACTION (P):
        (* Enoncés ST *)
END_ACTION ;
```

Voici le chronogramme d'une action de type **P**:



Exemple d'action impulsionnelle:

(* Programme SFC avec actions impulsionnelles *)

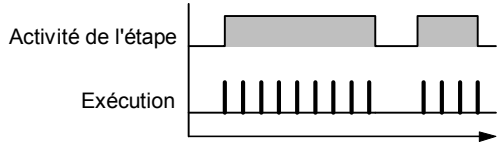


B.3.5.3 Actions normales

Une **action normale** (ou non mémorisée) est une liste d'instructions ST ou IL, exécutée **à chaque cycle** pendant toute la durée d'**activité** de l'étape. Les instructions sont écrites selon la syntaxe SFC suivante:

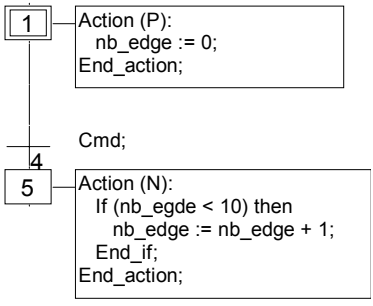
```
ACTION (N):
        (* Enoncés ST *)
END_ACTION ;
```

Voici le chronogramme d'une action de type **N**:



Exemple d'action normale:

(* Programme SFC avec actions normales *)



B.3.5.4 Actions SFC

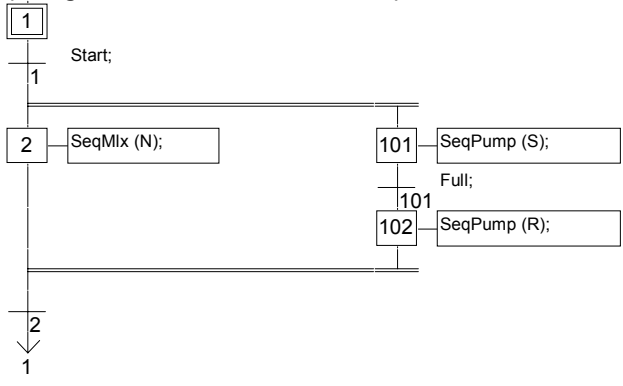
Une action SFC est une séquence fille SFC, lancée ou tuée selon les évolutions du signal d'activité de l'étape. Une action SFC peut être décrite avec les qualificatifs d'action **N** (Non mémorisée), **S** (Set), ou **R** (Reset). Voici la syntaxe des actions SFC:

<progr_fils> (N) ;	lance la séquence fille quand l'étape devient active, et la tue lorsque l'étape redevient inactive
<progr_fils> ;	même effet (l'attribut N est optionnel)
<progr_fils> (S) ;	lance la séquence fille quand l'étape devient active. rien n'est fait lorsque l'étape redevient inactive
<progr_fils> (R) ;	tue la séquence fille quand l'étape devient active. rien n'est fait lorsque l'étape redevient inactive

La séquence SFC spécifiée dans l'action doit être un **programme SFC fils** du programme en cours d'édition. Notez que l'utilisation des qualificatifs **S** (Set) ou **R** (Reset) pour une action SFC a exactement le même effet que les énoncés **GSTART** et **GKILL**, programmés dans une action de type **P** (impulsionnelle) en **ST**.

Voici un exemple d'action SFC. Le programme SFC principal est nommé **Princ**. Il a deux programmes SFC fils, nommés **SeqMlx** et **SeqPomp**. Voici la programmation en SFC du programme principal:

(* Programme SFC avec actions SFC *)



B.3.5.5 Appel de fonction ou de bloc fonctionnel dans une action

Les sous-programmes, fonctions ou blocs fonctionnels (écrits en ST, IL, LD ou FBD), ou les fonctions ou blocs fonctionnels "C" peuvent être directement appelés depuis un bloc d'action, selon la syntaxe suivante:

Pour les sous-programmes, fonctions ou fonctions "C":

```
ACTION (P):
    <resultat>:= <sous_programme> ( ) ;
END_ACTION;
```

ou

```
ACTION (N):
    <resultat>:= <sous_programme> ( ) ;
END_ACTION;
```

Pour les blocs fonctionnels écrits en "C" ou en ST, IL, LD, FBD:

```
ACTION (P):
    Fbinst(in1, in2);
    result1:= Fbinst.out1;
    result2:= Fbinst.out2;
END_ACTION;
```

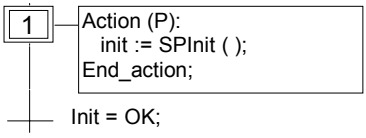
ou

```
ACTION (N):
        Fbinst(in1, in2);
        result1:= Fbinst.out1;
        result2:= Fbinst.out2;
END_ACTION;
```

Vous trouverez la syntaxe détaillées de ces appels dans le chapitre concernant le langage ST.

Exemple d'appel de sous-programme dans un bloc d'action:

(* Appel de sous-programme dans une action SFC*)



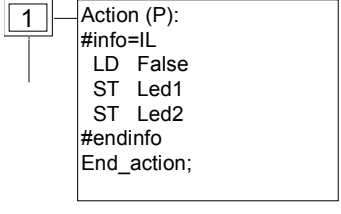
B.3.5.6 Convention d'écriture en langage IL

La programmation en **IL** (Instruction List) peut être directement insérée dans un bloc **d'action**, selon la syntaxe suivante:

```
ACTION (P):  (* ou N *)
#info=IL
        <instruction>
        <instruction>
        ....
#endinfo
END_ACTION;
```

Les mots clés "**#info=IL**" et "**#endinfo**" doivent être entrés exactement sous cette forme. Respectez les majuscules et les minuscules. N'insérez aucun caractère blanc ou tabulation avant ou après ces mots clés. Voici un exemple d'action programmée en IL:

(* Programme SFC avec action programmée en IL *)



B.3.6 Conditions attachées aux transitions

A chaque transition est attachée une **expression booléenne**, qui conditionne le franchissement de la transition. Les **conditions** sont généralement décrites en langage ST. C'est le **niveau 2** de la transition. Mais d'autres conventions d'écriture sont autorisées:
- Convention d'écriture en ST
- Convention d'écriture en LD
- Convention d'écriture en IL
- Appel de fonctions

Attention: Quand aucune condition n'est attachée à une transition, sa **réceptivité** par défaut est **TRUE** (vraie).

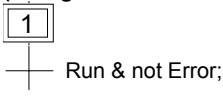
B.3.6.1 Convention d'écriture en langage ST

Vous pouvez utiliser le langage **ST** (Structured Text) pour décrire la **condition** attachée à une transition. L'expression doit être du type **booléen** et doit être terminée par un **point virgule**, selon la syntaxe suivante:

< expression_booleenne > ;

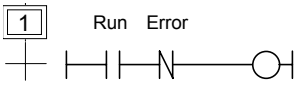
L'expression peut être une valeur constante TRUE ou FALSE, une variable booléenne interne ou d'entrée, ou une combinaison de variables représentant une grandeur booléenne. Voici un exemple de transition programmée en ST:

(* Programme SFC avec conditions programmées en ST *)



B.3.6.2 Convention d'écriture en langage LD

Le langage LD (schéma à relais) peut être utilisé pour décrire une condition associée à une transition. Le schéma est alors composé d'un seul échelon contenant un seul relais. La valeur du relais représente la valeur de la transition.
Voici un exemple de transition programmée en LD:



B.3.6.3 Convention d'écriture en langage IL

Le langage **IL** (Instruction List) peut directement être utilisé pour programmer la condition d'une transition, selon la syntaxe suivante:

#info=IL

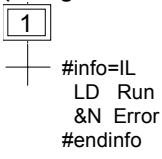
```
        <instruction>
        <instruction>
        ....
#endinfo
```

La valeur contenue dans le **résultat courant** (registre IL) à la fin de la séquence programmée indique le résultat de la condition attachée à la transition:

résultat courant = 0	➔	condition = **FALSE**
résultat courant <> 0	➔	condition = **TRUE**

Les mots clés "**#info=IL**" et "**#endinfo**" doivent être entrés exactement sous cette forme. Respectez les majuscules et les minuscules. N'insérez aucun caractère blanc ou tabulation avant ou après ces mots clés. Voici un exemple de condition programmée en IL:

(* Programme SFC avec transitions programmées en IL *)



B.3.6.4 Appel de fonction dans une transition

Un **sous-programme** ou une **fonction** (écrit en FBD, LD, ST ou IL) peut être appelé pour évaluer la condition attachée à une transition, selon la syntaxe suivante:

< sous_programme > ( ) ;

La valeur retournée par le sous-programme indique le résultat de la condition de franchissement de la transition:

valeur retournée = FALSE	➔	condition = **FALSE**
valeur retournée = TRUE	➔	condition = **TRUE**

Voici un exemple d'appel de sous-programme dans une transition:

(* Programme SFC avec appel de sous-programme *)

B.3.7 Règles d'évolution du SFC

Voici les **cinq règles d'évolution** définies par le langage SFC:

Situation initiale

La **situation initiale** est caractérisée par les **étapes initiales** qui définissent l'état du procédé au lancement de l'application. Il doit y avoir **au moins une** étape initiale dans un programme SFC.

Franchissement d'une transition

Une transition est soit **valide** soit **invalide**. Elle est valide quand toutes les étapes immédiatement précédentes sont **actives**.
Une transition ne peut être **franchie** que si:
- elle est valide, et
- la condition associée est vraie.

Evolution des étapes actives

Le franchissement d'une transition entraîne simultanément **l'activation** de toutes les étapes immédiatement suivantes et la **désactivation** de toutes les étapes immédiatement précédentes.

Franchissement simultané de transitions

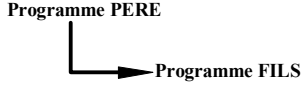
Les lignes doubles peuvent être utilisées pour représenter les transitions qui doivent être franchies simultanément. Si ces transitions sont représentées séparément, le test de l'activité des étapes précédentes (GSnnn.x) peut être inclus dans leur condition.

Activation et désactivation simultanées

Si, durant le séquencement, une étape doit être simultanément activée et désactivée, la priorité est donnée à l'activation.

B.3.8 Hiérarchie des programmes SFC

Le système ISaGRAF permet la description d'une structure verticale des programmes SFC. Les programmes SFC sont organisés dans un système **hiérarchique**. Chaque programme peut contrôler (lancer, tuer...) d'autres programmes SFC. Chaque programme SFC peut contrôler d'autres programmes SFC appelés ses **fils**. Les programmes SFC sont reliés dans un **arbre de hiérarchie**, par des relations de type "**père - fils**":



Voici les règles de base imposées par la hiérarchie:

- Les programmes SFC qui n'ont pas de père sont appelés "**principaux**"
- Les programmes principaux sont activés au lancement de l'application
- Un programme père peut avoir plusieurs programmes fils
- Un programme fils ne peut avoir qu'un seul programme père
- Un programme fils ne peut être contrôlé que par son programme père
- Un programme ne peut pas contrôler les fils de ses programmes fils

Voici les actions qu'un programme SFC père peut effectuer pour contrôler un programme SFC fils:

- Lancer (**GSTART**) Lance le programme fils: active chacune de ses étapes initiales. Les fils du programme lancé ne sont pas automatiquement lancés avec lui.
- Tuer (**GKILL**) Tue le programme en désactivant chacune de ses étapes actives. Tous ses programmes fils sont tués avec lui.
- Figer (**GFREEZE**) Suspend l'exécution du programme (désactive les actions des étapes actives et le calcul des transitions), et mémorise l'état des étapes actives pour le programme puisse être relancé ultérieurement. Les fils du programme figés sont également figés.
- Relancer (**GRST**) Relance un programme SFC en réactivant toutes les étapes qui ont été désactivées quand le programme a été figé. Les fils du programme relancé ne sont pas relancés automatiquement.
- Tester (**GSTATUS**) Rend l'état (figé, actif, inactif) du programme.

B.4 Flow Chart

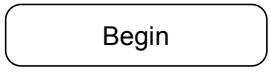
Le **Flow Chart** (**FC**) est un langage graphique qui décrit des opérations séquentielles et un flux de décision. Un diagramme Flow Chart est composé d' **Actions** et de **Décisions**. Les actions et décisions sont reliés par des liaisons orientées représentant le flux. Les actions et les décisions peuvent être décrits en langage ST, LD ou IL. Des fonctions et blocs fonctionnels écrits en tout langage (à l'exception du SFC et du FC) peuvent être appelés depuis les actions et les décisions. Un programme Flow Chart peut appeler un autre programme Flow Chart. Le programme FC appelé est appelé **sous-programme FC** du programme appelant.

B.4.1 Composants du langage FC

Voici les composants graphiques élémentaires du langage Flow Chart:

Début de graphe

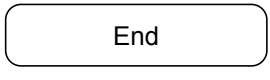
Un symbole "**begin**" doit apparaître au début du diagramme Flow Chart. Il est unique et ne peut mas être omis. Il représente l'état initial du diagramme lors de son activation. Voici le dessin du symbole "begin":



Le symbole "Begin" a toujours une connection (en aval) vers un autre objet du diagramme. Un diagramme flow chart est invalide si aucune liaison n'est tracée depuis le symbole "Begin" vers un autre élément.

Fin du graphe

Un symbole "**end**" doit apparaître à la fin du diagramme Flow Chart program. Il est unique et ne peut pas être omis. Toutefois il est possible qu'aucune liaison ne mène au symbole "End" (diagramme bouclant indéfiniment), mais le symbole "End" est tout de même présent dans le diagramme. Il représente l'état final du diagramme, quand son exécution est terminées. Voici le dessin d'un symbole "End":



Le symbole "End" a généralement une connection (en amont) provenant d'un autre objet du diagramme.

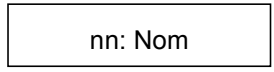
Flux

Une **liaison** est une ligne orientée qui représente le flux de contrôle entre deux points du diagramme. Une liaison est toujours terminée par une flèche qui indique la direction du flux. Voici le dessin d'une liaison:

Deux liens ne peuvent pas être conectés à partir du même point.

Actions

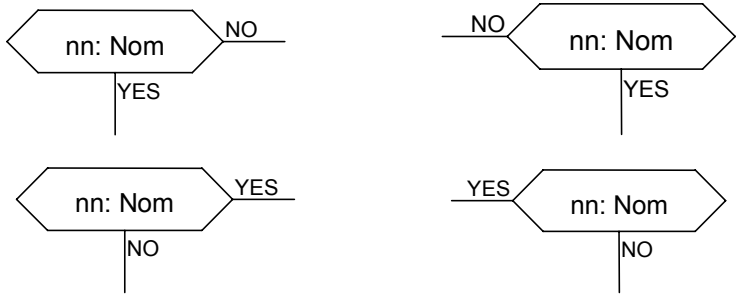
Le symbole d'une **action** représente des actions à exécuter. Une action est identifiée par un numéro et un nom. Voici le dessin d'un symbole d'action:



Deux objets du diagramme ne peuvent pas avoir le même numéro. Le langage de programmation d'une action peut être le ST, le LD ou le IL. Une action doit toujours être connectée en amont et en aval aux autres objets du diagramme.

Décisions

Une **décision** représente un **test** booléen. Une décision est identifiée par un numéro et un nom. Selon l'expression de la condition, exprimée en ST, LD ou IL, le flux est dirigé soit vers la sortie "YES" soit vers la sortie "NO". Voici les différentes possibilités pour le dessin d'une décision:



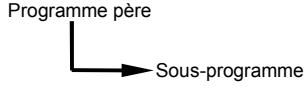
Deux objets du diagramme ne peuvent pas avoir le même numéro. La programmation de la condition peut être réalisée avec:
- une expression booléenne en ST, ou
- une échelle unique en LD, sans symbole attaché à la bobine, ou
- plusieurs instructions en IL. Le résultat courant IL en fin de liste contient le résultat de la condition.

Quand la condition est programmée en ST, l'expression peut optionnellement être terminée par un point-virgule. Quand la condition est programmée en LD, la bobine unique représente la valeur de la condition. Une condition peut être égale à:
- FALSE qui représente la décision NO
- TRUE qui représente la décision YES

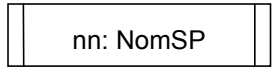
Un test doit toujours avoir une connection en amont et deux connections aval reliées à ses orties "Yes" et "No".

Sous-programme

ISaGRAF permet la représentation d'une structure verticale de programmes FC. Les programmes FC sont organisés dans un **arbre de hiérarchie**. Chaque programme FC peut appeler un ou plusieurs **sous-programmes FC**. Les programmes FC sont reliés dans l'arbre de hiérarchie par une relation de type "père / fils" matérialisé par un lien terminé par une flèche:



Un symbole de **sous-programme** dans un diagramme Flow Chart représente un appel au sous programme. L'exécution du programme appelant est suspendue pendant toute l'exécution du sous-programme. Un sous-programme Flow Chart est identifié ar son nom et par un numéro. Voici le dessin d'un appel de sous-programme:

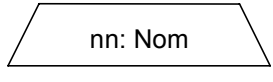


Deux objets du diagramme ne peuvent pas avoir le même numéro. Les règles suivantes doivent être respectées pour la mise en œuvre de la hiérarchie des programmes FC:
- Les programmes FC n'ayant pas de père sont appelés programmes principaux.
- Les programmes principaux FC sont activés au lancement de l'application.
- Un programme peut avoir un ou plusieurs sous-programmes.
- Un sous-programme ne peut pas avoir plusieurs programmes pères.
- Un sous-programme ne peut être appelé que par son programme père.
- Un programme ne peut pas appeler un fils d'un de ses fils

L'appel au même sous-programme peut apparaître plusieurs foid dans le même diagramme. Un appel de sous-programme FC représente l'exécution totale du sous-programme. Le programme père est suspendu pendant l'exécution du sous-programme. Un appel de sous-programme doit avoir des connections en amont et en aval vers les autres éléments du diagramme.

Action spécifique

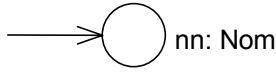
Un symbole d'**action spécifique** représente une série d'actions à exécuter. Comme les autres symboles actions, il est identifié par un numéro et un nom. L'exécution d'une action spécifique n'a pas de différence avec celle d'une action normale. Le but des actions spécifiques est simplement de rendre le diagramme plus lisible en marquand les parties non portables du programme. L'utilisation des symboles d'action spécifique est optionnelle. Voici le dessin d'une action spécifique:



Les actions spécifiques ont rigoureusement le même comportement que les autres actions, qu'il s'agisse du mode de programmation, des règles de connection aux autres éléments du diagramme, ou du comportement dynamique.

Connecteurs

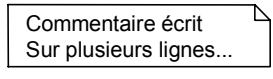
Les **connecteurs** permettent d'exprimer une liaison entre deux points du diagramme, sans que le trait de la liaison complète soit tracé. Un connecteur est représenté par un cercle, connecté par un trait de liaison à la source du lien à exprimer. Le dessin du connecteur est complété, à sa droite, par le nom de l'élément qui est la destination du lien. Voici le dessin d'un connecteur:



Un connecteur se réfère toujours à un élément défini dans le même diagramme. La destination de la liaison est exprimée par le numéro de l'élément de destination.

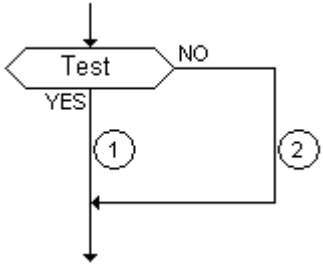
Commentaires

Les blocs de **commentaires** n'ont pas d'influence sur le comportement du diagramme. Ils peuvent être insérés partout dans le diagramme, and permettent de documenter le programme. Voici le dessin d'un bloc de commentaire:



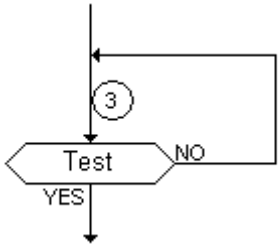
B.4.2 Structures complexes

Cette section montre des exemples de **structures complexes** pouvant apparaître dans les diagrammes FC. Ces combinaisons de base permettent la construction de diagrammes structurés.



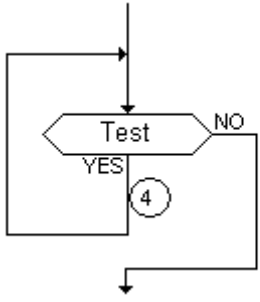
IF / THEN / ELSE

(1) les actions pour "THEN" sont insérées ici
(2) les actions pour "ELSE" sont insérées ici



REPEAT / UNTIL

(3) les actions à répéter sont insérées ici



WHILE / DO

(4) les actions à répéter sont insérées ici

B.4.3 Exécution du Flow Chart

Voici les règles de base de l'**exécution** d'un diagramme Flow Chart:

- L'activation du symbole Begin nécesite un cycle sur la cible
- L'activation du symbole End nécessite un cycle sur la cible et engendre la fin de l'exécution du diagramme. Aucune action n'est plus exécutée après le symboel End.
- L'exécution est poursuivie, dans le même cycle, jusqu'à ce qu'un élément déjà rencontré soit rencontré à nouveau (boucle).

Note: Contrairement au SFC, une action n'est pas un état stable. Il n'y a jamais de répétition des instructions quand le symbole d'une action est rencontré.

B.4.4 Règles syntaxiques

En plus des règles syntaxiques des actions et conditions programmées en ST, LD ou IL, d'autres **règles syntaxiques** s'appliquent au diagramme lui-même:
- Tous les "points de connection" des symboles doivent être correctement reliés aux autres éléments du graphe. (seule la connection entrée du symbole "End" peut être omise)
- Tous les symboles doivent être reliés ensemble (pas de partie de graphe isolée)
- Tous les connecteurs doivent avoir une destination valide

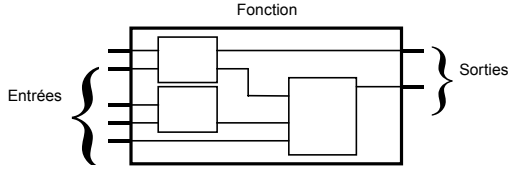
Il est à noter que:
- Les actions vides (sans programmation) sont autorisées
- Les décisions vides (sans programmation) sont considérés comme toujours égales à TRUE.

B.5 Langage FBD

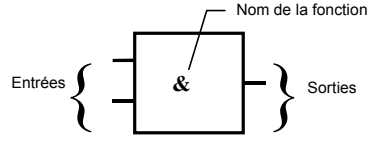
Le **FBD** (Function Block Diagram) est un langage graphique. Il permet la construction d'équations complexes à partir des fonctions élémentaires des librairies ISaGRAF.

B.5.1 Format du diagramme FBD

Le diagramme FBD décrit une fonction entre des **variables d'entrée** et des **variables de sortie**. Une fonction est décrite comme un réseau de **fonctions élémentaires**. Les variables d'entrée et de sortie sont connectées aux boîtes fonctions par des **arcs de liaison**. Une sortie d'une boîte peut être connectée sur une entrée d'une autre boîte.



La fonction complète représentée par un programme FBD est construite à l'aide de **boîtes fonctions élémentaires** de la librairie ISaGRAF. Chaque boîte fonction élémentaire a une nombre prédéfini de **points de connexion** en entrée et en sortie. Une boîte fonction est représenté par un **rectangle**. Ses entrées sont connectées sur le bord **gauche** du rectangle. Ses sorties sont connectées sur le bord **droit**. Une boîte fonction élémentaire réalise une **fonction** élémentaire entre ses entrées et ses sorties. Le nom de la fonction réalisée est inscrite dans le rectangle de la boîte. Chaque entrée ou sortie de la boîte est caractérisée par un **type**.



Les variables d'entrée du diagramme FBD doivent être connectées aux entrées des boîtes fonctions. Le type de chaque variable doit être cohérent avec le type de l'entrée de la boîte correspondante. Une entrée pour le diagramme FBD peut être une expression **constante**, toute variable **interne**, d'**entrée** ou de **sortie**

Les variables de sortie du diagramme FBD doivent être connectées aux sorties des boîtes fonctions. Le type de chaque variable doit être cohérent avec le type de la sortie de la boîte correspondante. Une sortie pour le diagramme FBD peut être une variable **interne** ou de **sortie**, ou le nom du programme édité (pour les **sous-programmes** seulement) Quand le

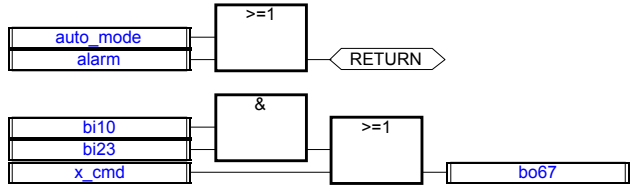
nom du sous-programme édité est utilisé comme sortie du diagramme, il représente une assignation de la valeur retournée par le sous-programme.
Les variables d'entrée et de sortie, les entrées et les sorties des boîtes fonctions sont reliées par des **arcs de liaison**. Des lignes sont utilisées pour représenter les liaisons entre deux points du diagramme:

- Une variable d'entrée et une entrée de boîte
- Une sortie d'une boîte et une entrée d'un autre boîte
- Une sortie de boîte et une variable de sortie

Les liaisons sont **orientées**: une liaison transporte une information depuis son extrémité gauche vers son extrémité droite. Les extrémités gauche et droite doivent être connectées à des éléments de **même type**.
Des **liaisons multiples** à droite sont utilisées pour transmettre une information depuis une extrémité à gauche vers plusieurs points à droite. Toutes les extrémités de la liaison doivent avoir le **même type**.

B.5.1.1 L'énoncé RETURN

Le mot clé "**<RETURN>**" peut apparaître comme une sortie du diagramme. Il doit être connecté à une sortie booléenne d'une boîte fonction. L'énoncé RETURN représente une **terminaison conditionnelle** de l'exécution du programme: si la liaison connectée prend l'état booléen **TRUE**, la fin du diagramme n'est pas interprétée. Voici un exemple d'énoncé RETURN:



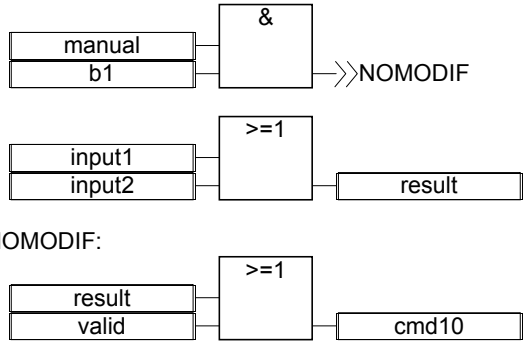

```
(* Equivalence ST *)
If auto_mode OR alarm Then
        Return;
End_if;
bo67:= (bi10 AND bi23) OR x_cmd;
```

B.5.1.2 Sauts et étiquettes

Les **étiquettes** et les **sauts** conditionnels, sont utilisés pour contrôler l'exécution du diagramme. Aucune connexion ne peut être réalisée à droite d'un symbole d'étiquette ou de saut.

>>LAB saut à une étiquette (le nom de l'étiquette est "LAB")
LAB: définition d'une étiquette (nommée "LAB")

Si la liaison **à gauche** du symbole de saut prend l'état booléen **VRAI**, l'exécution du programme est déroutée après l'étiquette correspondante. Voici un Exemple d'utilisation de sauts et étiquettes:

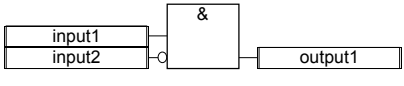



```
(* Equivalence IL: *)
                ld      manual
                and     b1
                jmpc    NOMODIF
                ld      input1
                or      input2
                st      result
NOMODIF:        ld      result
                or      valid
                st      cmd10
```

B.5.1.3 Inversion booléenne

Un arc de liaison simple, connecté à une entrée booléenne d'une boîte fonction, peut être terminé par un symbole d'**inversion booléenne**. L'inversion est représentée par un petit cercle. Quand une inversion booléenne est utilisée, les extrémités gauche et droite de l'arc concerné doivent être de type **booléen**. Voici un exemple d'utilisation d'une inversion booléenne:

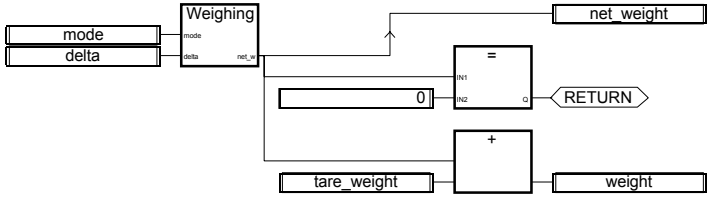
(* Liaison avec inversion booléenne *)




```
(* Equivalence ST: *)
output1:= input1 AND NOT (input2);
```

B.5.1.4 Appel de fonctions ou blocs fonctionnels depuis le FBD

Le langage FBD permet l'appel de **sous-programmes**, de **fonctions** ou de **blocs fonctionnels**. Un sous-programme, une fonction ou un bloc fonctionnel est représenté comme une boîte fonction. Le nom de la boîte est le nom du sous-programme, de la fonction ou du bloc fonctionnel.
Dans le case d'un sous-programme ou d'une fonction, la valeur rendue par le sous-programme est la seule sortie de la boîte. Un bloc fonctionnel peut avoir plusieurs sorties.
Voici un exemple d'appel de sous-programme:




```
(* Equivalence ST *)
net_weight:= Weighing (mode, delta); (* appel d'un sous-programme *)
If (net_weight = 0) Then Return; End_if;
weight:= net_weight + tare_weight;
```

B.6 Langage LD

Le langage **LD** (Ladder Diagram) est une représentation graphique d'équations booléennes combinant des **contacts** (en entrée) et des **relais** (en sortie). Le langage LD permet la manipulation de données **booléennes**, à l'aide de **symboles graphiques** organisés dans un diagramme. Les symboles du diagramme LD sont organisés comme les éléments d'un schéma électrique à contacts. Les diagrammes LD sont limités à gauche et à droite par des **barres d'alimentation**. Voici les composants graphiques élémentaires d'un diagramme LD:


---------- Barre d'alimentation à gauche


---------- Barre d'alimentation à droite


---------- Arc de liaison horizontal


---------- Arc de liaison vertical

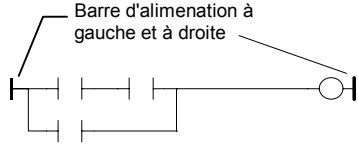

---------- Liaisons multiples verticales et horizontales


---------- Contact associé à une variable

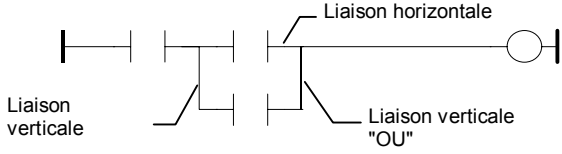

---------- Relais associé à une variable de sortie ou interne

B.6.1 Barres d'alimentation

Un diagramme LD est limité sur la gauche et la droite par des lignes verticales appelées respectivement **barre d'alimentation à gauche** et **barre d'alimentation à droite**.



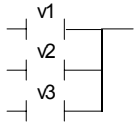
Les symboles du diagramme LD sont reliés ensemble et aux barres d'alimentation par des **arcs de liaison**. Les liaisons doivent être verticales ou horizontales.



Chaque segment de liaison peut prendre l'état booléen **FALSE** ou **TRUE**. L'état d'une liaison est propagé dans toutes ses extrémités à droite. Toute liaison horizontale connectée à la **barre d'alimentation à gauche** a l'état booléen **TRUE**.

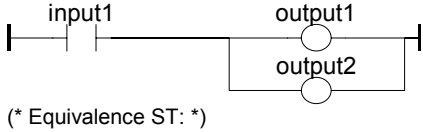
B.6.2 Liaison multiple

L'état booléen d'un arc de liaison horizontal est le même à son extrémité gauche et à son extrémité droite. La combinaison de liaisons horizontales et verticales permet la construction de **liaisons multiples**. L'état aux extrémités d'une liaison multiple respecte des règles logiques.
Une **liaison multiple à gauche** combine **plusieurs** liaisons horizontales connectées à **gauche** d'une liaison verticale, et **une seule** liaison horizontale à **droite**. L'état de l'extrémité à droite est le résultat du **OU LOGIQUE** entre les états des extrémités à gauche.
(* Exemple de liaison multiple à gauche *)



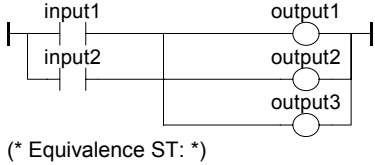
(* Etat de l'extrémité droite: (v1 OR v2 OR v3) *)

Une **liaison multiple à droite** combine **une seule** liaison horizontale connectées à **gauche** d'une liaison verticale, et **plusieurs** liaisons horizontales à **droite**. L'état de l'extrémité à gauche est propagé dans toutes les extrémités à droite.
(* Exemple de liaison multiple à droite *)



(* Equivalence ST: *)
output1:= input1;
output2:= input1;

Une **liaison multiple à gauche** combine **plusieurs** liaisons horizontales connectées à **gauche** d'une liaison verticale, et **plusieurs** liaisons horizontales à **droite**. L'état de chacune des extrémités à droite est le résultat du **OU LOGIQUE** entre les états des extrémités à gauche.
(* Exemple de liaison multiple à gauche et à droite *)



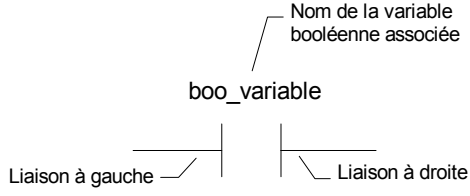
(* Equivalence ST: *)
output1:= input1 OR input2;
output2:= input1 OR input2;
output3:= input1 OR input2;

B.6.3 Contacts et relais

Les types de **contacts** suivants peuvent être utilisés dans un diagramme:
- Contacts directs
- Contacts inversés
- Contacts à détection de front

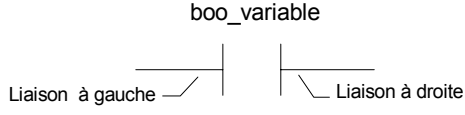
Les types de relais suivants peuvent être utilisés dans un diagramme:
- Relais directs
- Relais inversés
- Relais à forçage de type SET
- Relais à forçage de type RESET
- Relais à détection de front

Le nom de la variable associée est inscrit au dessus du symbole graphique:



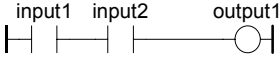
B.6.3.1 Contacts directs

Un **contact direct** représente une opération entre l'état d'un **arc de liaison** et une **variable** booléenne.



L'état de la liaison à droite est le résultat du **ET LOGIQUE** entre l'état de la liaison à gauche et la valeur de la variable associée au contact.

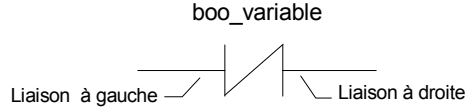
(* Exemple de contacts directs: *)



(* Equivalence ST: *)
output1:= input1 AND input2;

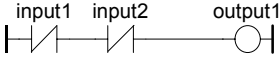
B.6.3.2 Contacts inversés

Un **contact inversé** représente une opération entre l'état d'un **arc de liaison** et l'**inverse logique** d'une **variable** booléenne.



L'état de la liaison à droite est le résultat du **ET LOGIQUE** entre l'état de la liaison à gauche et l'**INVERSE LOGIQUE** de la valeur de la variable associée au contact.

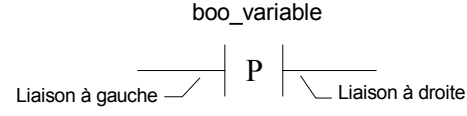
(* Exemple de contacts inversés *)




```
(* Equivalence ST: *)
output1:= NOT (input1) AND NOT (input2);
```

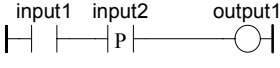
B.6.3.3 Contacts à détection de front positif

Un tel contact représente une opération entre l'état d'un **arc de liaison** et le passage à l'état **VRAI** d'une **variable** booléenne: **front montant**.



L'état de la liaison à droite prend l'état **VRAI** quand l'état de la liaison à gauche et à **VRAI** et que l'état de la variable associée passe de **FAUX** à **VRAI**. Il prend l'état **FAUX** dans tous les autres cas.

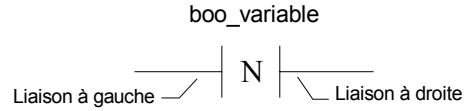
(* Exemple de contacts à détection de front positif *)



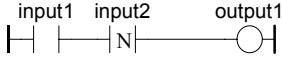

```
(* Equivalence ST: *)
output1:= input1 AND (input2 AND NOT (input2prev));
(* input2prev est la valeur de input2 au cycle précédent *)
```

B.6.3.4 Contacts à détection de front négatif

Un tel contact représente une opération entre l'état d'un **arc de liaison** et le passage à l'état **FAUX** d'une **variable** booléenne: **front descendant**.



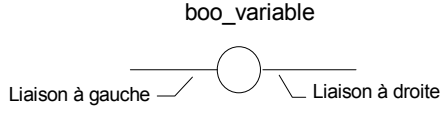
L'état de la liaison à droite prend l'état **VRAI** quand l'état de la liaison à gauche et à **VRAI** et que l'état de la variable associée passe de **VRAI** à **FAUX**. Il prend l'état **FAUX** dans tous les autres cas.
(* Exemple de contacts à détection de front négatif *)



(* Equivalence ST: *)
output1:= input1 AND (NOT (input2) AND input2prev);
(* input2prev est la valeur de input2 au cycle précédent *)

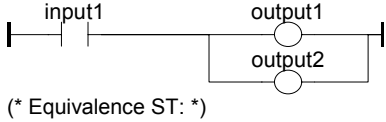
B.6.3.5 Relais direct

Une relais direct permet le forçage d'une **sortie booléenne** avec l'état d'un **arc de liaison**.



La variable associée est forcée à l'**état de la liaison à gauche**. L'état de la liaison à gauche est propagé dans la liaison à droite. La liaison à droite peut être connectée à la barre d'alimentation à droite. La variable booléenne associée doit être de **SORTIE** ou **INTERNE**. Le nom de la variable associée peut être le nom du programme édité (pour les **sous-programmes** seulement). Dans ce cas le relais représente l'affectation de la valeur renvoyée par le sous-programme.

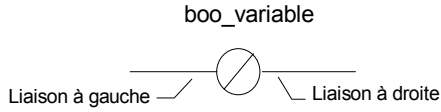
(* Exemple de relais directs *)



(* Equivalence ST: *)
output1:= input1;
output2:= input1;

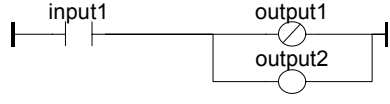
B.6.3.6 Relais inversés

Une **relais inversé** permet le forçage d'une **sortie booléenne** avec l'**inverse logique** de l'état d'un **arc de liaison**.



La variable associée est forcée à l'**inverse logique** de l'**état de la liaison à gauche**. L'état de la liaison à gauche est propagé dans la liaison à droite. La liaison à droite peut être connectée à la barre d'alimentation à droite. La variable booléenne associée doit être de **SORTIE** ou **INTERNE**. Le nom de la variable associée peut être le nom du programme édité (pour les **sous-programmes** seulement). Dans ce cas le relais représente l'affectation de la valeur renvoyée par le sous-programme.

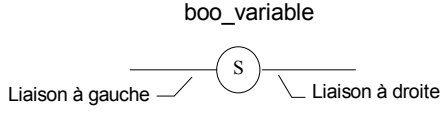
(* Exemple de relais inversés *)



(* Equivalence ST: *)
output1:= NOT (input1);
output2:= input1;

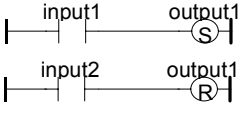
B.6.3.7 Relais à action "SET"

Un relais à action "Set" permet le forçage d'une **sortie booléenne** en fonction de l'état d'un **arc de liaison**.



La variable associée est **FORCEE A TRUE** quand la **liaison à gauche** prend l'état TRUE. La variable garde cette valeur jusqu'à un ordre inverse donné par un relais de type "RESET". L'état de la liaison à gauche est propagé dans la liaison à droite. La liaison à droite peut être connectée à la barre d'alimentation à droite. La variable booléenne associée doit être de **SORTIE** ou **INTERNE**.

(* Exemple de relais à action SET et RESET *)

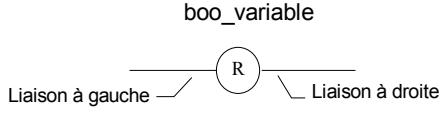


(* Equivalence ST: *)

```
IF input1 THEN
  output1:= TRUE;
END_IF;
IF input2 THEN
  output1:= FALSE;
END_IF;
```

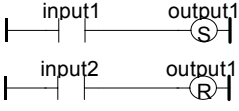
B.6.3.8 Relais à action "RESET"

Un relais à action "Reset" permet le forçage d'une **sortie booléenne** en fonction de l'état d'un **arc de liaison**.



La variable associée est **FORCEE A FALSE** quand la **liaison à gauche** prend l'état TRUE. La variable garde cette valeur jusqu'à un ordre inverse donné par un relais de type "SET". L'état de la liaison à gauche est propagé dans la liaison à droite. La liaison à droite peut être connectée à la barre d'alimentation à droite. La variable booléenne associée doit être de **SORTIE** ou **INTERNE**.

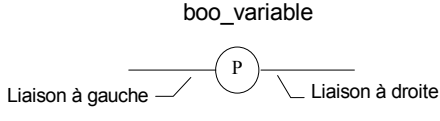
(* Exemple de relais à action SET et RESET *)




```
(* Equivalence ST: *)
IF input1 THEN
  output1:= TRUE;
END_IF;
IF input2 THEN
  output1:= FALSE;
END_IF;
```

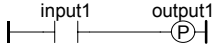
B.6.3.9 Relais à détection de front montant

Les relais "Positif" permet le forçage d'une **sortie booléenne** en fonction de l'état d'un **arc de liaison**. Ce type de relais est disponible uniquement dans l'éditeur Quick LD.



La variable est **FORCEE A TRUE** quand la **liaison à gauche** passe de l'état FALSE à l'état TRUE: **front montant**. La variable est remise à FALSE dans tous les autres cas. L'état de la liaison à gauche est propagé dans la liaison à droite. La liaison à droite peut être connectée à la barre d'alimentation à droite. La variable booléenne associée doit être de **SORTIE** ou **INTERNE**.

(* Exemple d'utilisation d'un relais "Positif" *)



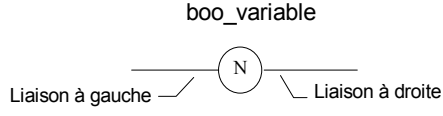
(* Equivalence ST: *)

```
IF (input1 and NOT(input1prev)) THEN
  output1:= TRUE;
ELSE
  output1:= FALSE;
END_IF;
```

(* input1prev est la valeur de input1 au cycle précédent *)

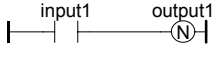
B.6.3.10 Relais à détection de front descendant

Les relais "Négatifs" permet le forçage d'une **sortie booléenne** en fonction de l'état d'un **arc de liaison**. Ce type de relais est disponible uniquement dans l'éditeur Quick LD.



La variable est **FORCEE A TRUE** quand la **liaison à gauche** passe de l'état TRUE à l'état FALSE: **front descendant**. La variable est remise à FALSE dans tous les autres cas. L'état de la liaison à gauche est propagé dans la liaison à droite. La liaison à droite peut être connectée à la barre d'alimentation à droite. La variable booléenne associée doit être de **SORTIE** ou **INTERNE**.

(* Exemple d'utilisation d'un relais "Négatif" *)



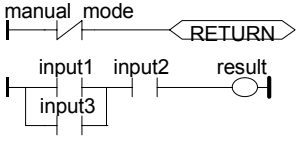

```
(* Equivalence ST: *)
IF (NOT(input1) and input1prev) THEN
  output1:= TRUE;
ELSE
  output1:= FALSE;
END_IF;
(* input1prev est la valeur de input1 au cycle précédent*)
```

B.6.4 L'énoncé RETURN

Le symbole **RETURN** peut être utilisé pour représenter une fin conditionnelle de l'exécution du programme. Aucune liaison ne doit être tracée à droite du symbole RETURN.

Si l'état de la **liaison à gauche** est **TRUE**, la suite du diagramme LD (lignes suivantes) n'est pas interprétée.

(* Exemple d'utilisation du symbole RETURN *)



(* Equivalence ST: *)
If Not (manual_mode) Then RETURN; End_if;
result:= (input1 OR input3) AND input2;

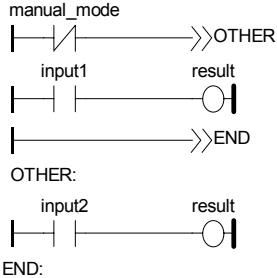
B.6.5 Sauts et étiquettes

Les **étiquettes**, les **sauts** conditionnels ou inconditionnels, sont utilisés pour contrôler l'exécution du diagramme. Aucune connexion ne peut être réalisée à droite d'un symbole d'étiquette ou de saut.

>>LAB ..... saut à l'étiquette de nom "LAB"
LAB: ..... définition de l'étiquette "LAB"

Si la liaison **à gauche** du symbole de saut prend l'état **TRUE**, l'exécution du programme est déroutée après l'étiquette correspondante.

(* Exemple d'utilisation de sauts et étiquettes *)

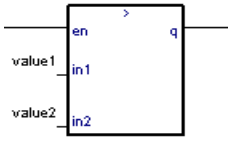


```
(* Equivalence IL: *)
                ldn         manual_mode
                jmpc        other
                ld          input1
                st          result
                jmp         END
OTHER:          ld          input2
                st          result
END:            (* fin du programme *)
```

B.6.6 Blocs en LD

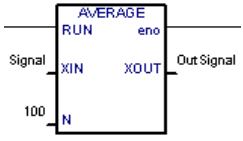
Lorsque vous utilisez l'éditeur Quick LD, des boites fonctions peuvent être connectées aux lignes booléennes. Une fonction peut être un opérateur, un bloc fonctionnel ou une fonction. Comme un bloc n'a pas forcément une entrée ou une sortie booléenne, l'insertion d'un bloc dans un diagramme LD peut entraîner l'ajout automatique de nouveaux paramètres à l'interface de ce bloc: EN, ENO. Ces paramètres EN / ENO ne sont pas ajoutés si vous utilisez l'éditeur FBD / LD, car avec cet éditeur vous avez la possibilité de connecter les variables du type requis.

• L'entrée "EN"
La première entrée de certains opérateurs, bloc fonctionnels ou fonctions, n'est pas du type booléen, et ne peut donc pas être connecté à une ligne LD booléenne. Donc, pour ces blocs, une entrée supplémentaire booléenne est ajoutée comme premier paramètre d'entrée. Cette entrée a le nom "**EN**". Dans ce cas, le bloc est exécuté uniquement si l'entrée **EN** reçoit un signal TRUE. Voici l'exemple du bloc comparateur et son équivalent en ST:



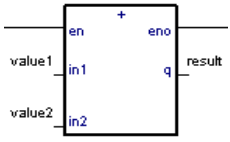

```
IF etat_ligne THEN
   q:= (value1 > value 2);
ELSE
   q:= FALSE;
END_IF;
(* la liaison à droite prend l'état de q *)
```

• La sortie "ENO"
La première sortie de certains opérateurs, bloc fonctionnel ou fonctions, n'est pas du type booléen, et ne peut donc pas être connectée à une ligne LD booléenne. Donc, pour ces blocs, une sortie supplémentaire booléenne est ajoutée comme premier paramètre de sortie. Cette sortie a le nom "**ENO**". La sortie **ENO** prend toujours l'état de la première entrée du bloc. Voici l'exemple du bloc AVERAGE et son équivalent en ST.




```
AVERAGE(etat_ligne, Signal, 100);
OutSignal:= AVERAGE.XOUT;
eno:= etat_ligne;
(* la liaison à droite prend l'état de eno *)
```

• Les paramètres "EN" et "ENO"
Dans certains cas, les deux paramètres **EN** et **ENO** sont ajoutés. Voici l'exemple d'un opérateur arithmétique et son équivalent en ST.




```
IF etat_ligne THEN
   result:= (value1 + value2);
END_IF;
eno:= etat_ligne;
(* la liaison à droite prend l'état de eno *)
```

B.7 Langage ST

Le langage **ST** (Structured Text) est un langage textuel de haut niveau dédié aux applications d'automatisation. Ce langage est principalement utilisé pour décrire les procédures complexes, difficilement modélisables avec les langages graphiques. Le ST est le langage par défaut pour la programmation des actions dans les étapes et des conditions associées aux transitions du **SFC**.

B.7.1 Syntaxe du ST

Un programme ST est une suite d'**énoncés**. chaque énoncé est terminé par un point virgule (**";"**).Les noms utilisés dans le code source (identificateurs de variables, constantes, mots clés du langage...) sont délimités par des **séparateurs passifs** (espace, fin de ligne ou tabulation) ou des **séparateurs actifs**, qui ont un rôle d'opérateur. Des **commentaires** peuvent être librement insérés dans la programmation. Un commentaire doit commencer par les caractères **"(*"** et se terminer par **"*)"**. Chaque énoncé est terminé par un point virgule (**";"**). Voici les types d'énoncés standards du ST:
- **assignation** (variable:= expression;)
- appel de **sous-programme** ou de **fonction**
- appel de **blocs fonctionnel**
- énoncés de **sélection** (IF, THEN, ELSE, CASE)
- énoncés d'**itération** (FOR, WHILE, REPEAT)
- énoncés de **contrôle** (RETURN, EXIT)
- énoncés spéciaux pour le lien avec le langage **SFC**

Des **séparateurs** passifs peuvent être librement insérés entre les séparateurs actifs, les expressions constantes et les identificateurs. Ces séparateurs sont l'**espace** (caractère blanc), les caractère de **tabulation** et de **fin de ligne**. Contrairement aux langages formatés en lignes tels que le **IL**, des caractères de fin de ligne peuvent être insérés partout dans le programme. Il est recommandé de respecter les règles suivantes quand vous utilisez les séparateurs passifs, pour assurer une bonne lisibilité du code source:
- Ne pas écrire plusieurs énoncés sur la même ligne
- Utiliser les tabulations pour indenter les structures de contrôle
- Insérer des commentaires

B.7.2 Expressions

Les expressions ST combinent des **opérateurs** et des **opérandes** variables ou constants. Pour chaque expression simple (combinant des opérandes avec un opérateur ST), le **typage** des opérandes doit être cohérent. L'expression simple a un type donné par ses opérandes et l'opérateur mis en oeuvre, et pourra être utilisée dans une expression plus complexe.
Par exemple:

(boo_var1 AND boo_var2)	a le type BOOLEEN
not (boo_var1)	a le type BOOLEEN
(sin (3.14) + 0.72)	a le type analogique REEL
(t#1s23 + 1.78)	est une expression invalide

Les **parenthèses** sont utilisées pour isoler des parties d'une expression, et donner explicitement un ordre d'évaluation des opérations. Quand aucune parenthèses n'est insérée, l'ordre d'évaluation est donné par la **priorité** entre les opérateurs du ST. Par exemple:

2 + 3 * 6	égale 2+18=20	parce que la multiplication est plus prioritaire
(2+3) * 6	égale 5*6=30	priorité indiquée par les parenthèses

Attention: Vous ne pouvez pas imbriquer plus de **8** niveaux de parenthèses dans la même expression.

B.7.3 Appel de fonction ou de bloc fonctionnel

La convention d'appel du ST pour les fonctions concerne les objets suivants:
- Sous-programmes
- Fonctions ou blocs fonctionnels de la librairie écrits en langage IEC
- Fonctions et blocs fonctionnels en "C"
- Fonctions de conversion de type

B.7.3.1 Appel de sous-programme ou de fonction

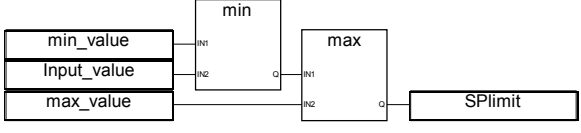
Nom:	nom du sous-programme appelé (ou fonction de librairie en langage IEC ou en "C")
Fonction:	appelle un sous-programme ou une fonction en ST, IL, LD ou FBD et rend sa valeur de retour
Syntaxe:	**<var_ana>:= <ssprog> (<par1>,** ... **<parN>);**
Opérandes:	la valeur de retour et les valeurs passées doivent respecter le typage des paramètres
Retour:	valeur retournée par le sous-programme

Un appel de sous-programme peut être inclus dans une expression complexe ou une condition SFC.

Exemple1: appel de sous-programme

```
(* Programme appelant, écrit en ST *)
(* acquisition d'une valeur analogique et conversion en temporisation *)
ana_timeprog:= SPlimit ( tprog_cmd );
appl_timer:= tmr (ana_timeprog * 100);
```

(* Programme appelé, nommé 'SPlimit', écrit en FBD *)



Exemple2: appel de fonction

(* les fonctions utilisées dans les expressions complexes: min, max, right, mlen et left sont des fonctions "C" standard *)
limited_value:= min (16, max (0, input_value));
rol_msg:= right (message, mlen (message) - 1) + left (message, 1);

B.7.3.2 Appel de bloc fonctionnel

Nom:	nom de l'instance du bloc fonctionnel
Fonction:	appelle un bloc fonctionnel de la librairie ISaGRAF ou de la librairie utilisateur et accède à ses paramètres de retour
Syntaxe:	**(* appel du bloc *)** **<nom_bloc> (<pa1>, <pa2> ...);** **(* récupère les paramètres de sortie *)** **<result>:= <nom_bloc>. <param_ret_1>;** ... **<result>:= <nom_bloc>. <nom_ret_N>;**
Opérandes:	les paramètres d'appel et l'affectation des paramètres de retour doivent respecter les types définis dans l'interface du bloc fonctionnel.
Retour:	Voir la syntaxe pour récupérer les paramètres de sortie.

Consultez la librairie ISaGRAF pour connaître le rôle et l'interface précis de chaque bloc fonctionnel. L'instance du bloc fonctionnel (nom de la copie) doit être déclarée dans le dictionnaire.

Exemple:

(* Programme ST avec appel de bloc fonctionnel *)
(* déclaration de l'instance dans le dictionnaire: *)
(* trigb1: bloc R_TRIG - détection de front montant *)

(* activation du bloc fonctionnel en ST *)
trigb1 (b1);
(* accès aux paramètres de retour *)
If (trigb1.Q) Then nb_fronts:= nb_fronts + 1; End_if;

B.7.4 Opérateurs booléens spécifiques au ST

Les opérateurs suivants sont spécifiques au langage ST:
- REDGE détection de front montant
- FEDGE détection de front descendant

Les opérateurs booléens suivants:
- NOT inversion booléenne
- AND (&) ET logique
- OR OU logique
- XOR OU exclusif logique

peuvent être utilisés. Leur description est détaillée dans le chapitre "Opérateurs, blocs fonctionnels et fonctions standard".

B.7.4.1 Opérateur "REDGE"

Nom:	**REDGE**
Fonction:	évalue le **front montant** d'une expression booléenne
Syntaxe:	<**front**>:**= REDGE (<expression_boo>, <variable_memo>);**
Opérandes:	le premier opérande est une variable booléenne ou toute expression booléenne complexe le second opérande est une variable booléenne interne utilisée pour mémoriser l'état précédent de l'expression
Retour:	TRUE quand l'expression passe de FALSE à TRUE FALSE dans tous les autres cas

Le front montant d'une expression ne peut être détecté qu'une seule fois, avec la fonction REDGE, dans le même cycle d'exécution.

Attention: La variable "mémoire" utilisée pour stocker l'état précédent de l'expression ne doit pas être utilisée dans les calculs de front pour d'autres expressions.

Quand l'expression est une variable booléenne nommée "**xxx**", déclarez une variable interne nommée "**EDGE_xxx**" dédiée aux opérations REDGE et FEDGE pour cette variable. Cette méthode évite les conflits entre variables pendant les calculs de détection de fronts.

Exemple:

```
(* Programme ST avec opérateur REDGE *)
(* ce programme compte les fronts montants d'une entrée booléenne *)
(* Bi120 est le nom de la variable d'entrée *)
(* Edge_Bi120 est la variable utilisée pour mémoriser l'état de Bi120 *)

If REDGE (Bi120, Edge_Bi120) Then
        Counter:= Counter + 1;
End_if;
```

Remarque: cet opérateur ne fait pas partie de la norme CEI 1131-3. Préférez le bloc standard R_TRIG. Cet opérateur a été conservé pour des raisons de compatibilité.

B.7.4.2 Opérateur "FEDGE"

Nom:	**FEDGE**
Fonction:	évalue le **front descendant** d'une expression booléenne
Syntaxe:	<**front**>:**= FEDGE (<expression_boo>, <variable_memo>);**
Opérandes:	le premier opérande est une variable booléenne ou toute expression booléenne complexe le second opérande est une variable booléenne interne utilisée pour mémoriser l'état précédent de l'expression
Retour:	TRUE quand l'expression passe de TRUE à FALSE FALSE dans tous les autres cas

Le front descendant d'une expression ne peut être détecté qu'une seule fois, avec la fonction FEDGE, dans le même cycle d'exécution.

Attention: La variable "mémoire" utilisée pour stocker l'état précédent de l'expression ne doit pas être utilisée dans les calculs de front pour d'autres expressions.

Quand l'expression est une variable booléenne nommée "**xxx**", déclarez une variable interne nommée "**EDGE_xxx**" dédiée aux opérations REDGE et FEDGE pour cette variable. Cette méthode évite les conflits entre variables pendant les calculs de détection de fronts.

Exemple:

```
(* Programme ST avec opérateur FEDGE *)
(* ce programme compte les fronts descendants d'une entrée booléenne *)
(* Bi120 est le nom de la variable d'entrée *)
(* Edge_Bi120 est la variable utilisée pour mémoriser l'état de Bi120 *)
If FEDGE (Bi120, Edge_Bi120) Then
        Counter:= Counter + 1;
End_if;
```

Remarque: cet opérateur ne fait pas partie de la norme CEI 1131-3. Préférez le bloc standard F_TRIG. Cet opérateur a été conservé pour des raisons de compatibilité.

B.7.5 Enoncés du langage ST

Voici les types d'énoncés du langage ST:
- Assignation
- Enoncé RETURN
- Sélection IF-THEN-ELSIF-ELSE
- Itération CASE
- Itération WHILE
- Itération REPEAT
- Itération FOR
- Enoncé EXIT

B.7.5.1 Assignation

Nom:	**:=**
Fonction:	assigne une variable avec une expression
Syntaxe:	<**variable**>:**=** <**expression**> **;**
Opérandes:	la variable doit être interne ou de sortie la variable et l'expression doivent avoir le même type

L'expression peut être un appel à un sous-programme ou à une fonction de la librairie ISaGRAF

Exemple:

```
(* Programme ST - exemple d'assignations *)
(* variable <<= variable *)
bo23:= bo10;
```

```
(* variable <<= expression *)
bo56:= bx34 OR alrm100 & (level >= over_value);
result:= (100 * input_value) / scale;

(* assignation avec retour de sous-programme *)
rc:= PSelect ( );

(*assignation avec retour de fonction *)
limited_value:= min (16, max (0, input_value) );
```

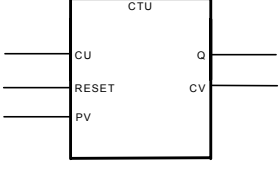
B.7.5.2 Enoncé "RETURN"

Nom: **RETURN**
Fonction: Termine l'exécution du programme
Syntaxe: **RETURN ;**
Opérandes: (aucun)

Si l'énoncé RETURN apparaît dans un bloc d'action SFC, la terminaison ne concerne que ce bloc d'action.

Exemple:

```
(* Spécifications FBD du programme: compteur programmable *)
```




```
(* implémentation ST du programme, avec énoncé RETURN *)

If not (CU) then
        Q:= false;
        CV:= 0;
        RETURN; (* fin du programme *)
end_if;

if R then
        CV:= 0;
else
        if (CV < PV) then
                CV:= CV + 1;
        end_if;
end_if;
Q:= (CV >= PV);
```

B.7.5.3 Enoncé "IF-THEN-ELSIF-ELSE"

Nom: **IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF**

Fonction: Exécute une parmi deux listes d'énoncés ST.
La sélection est effectuée en fonction de l'état d'une expression booléenne

Syntaxe:

```
IF <expression_booleenne> THEN
  <enonce> ;
  <enonce> ;
  ...
ELSIF <expression_booleenne> THEN
  <enonce> ;
  <enonce> ;
  ...
ELSE
  <enonce> ;
  <enonce> ;
  ...
END_IF;
```

Les énoncés ELSE et ELSIF sont optionnels. Si ELSE est omis, aucune instruction n'est exécutée quand l'expression de sélection prend la valeur FALSE.

Exemple:

```
(* Programme ST avec énoncé IF *)
IF manual AND not (alarm) THEN
        level:= manual_level;
        bx126:= bi12 OR bi45;
ELSIF over_mode THEN
        level:= max_level;
ELSE
        level:= (lv16 * 100) / scale;
END_IF;
(* Structure IF sans ELSE *)
If overflow THEN
        alrm_level:= true;
END_IF;
```

B.7.5.4 ENONCE "CASE"

Nom: **CASE ... OF ... ELSE ... END_CASE**

Fonction: exécute une parmi plusieurs liste d'énoncés ST
la sélection est faite en accord avec la valeur d'une expression analogique entière

Syntaxe:

```
CASE <expression_entiere> OF
  <valeur>: <enonces> ;
  <valeur> , <valeur>: <enonces> ;
  ...
ELSE
  <enonces> ;
END_CASE;
```

Les valeurs de sélection doivent être des expressions constantes entières. Plusieurs valeurs de sélection, séparées par des virgules, peuvent mener à la même liste d'énoncés. L'énoncé ELSE est optionnel.

Exemple:

```
(* Programme ST avec énoncé CASE *)

CASE error_code OF
        255:    err_msg:= 'Division by zero';
                fatal_error:= TRUE;
        1:      err_msg:= 'Overflow';
        2, 3:   err_msg:= 'Bad sign';
ELSE
        err_msg:= 'Unknown error';
END_CASE;
```

B.7.5.5 Enoncé "WHILE"

Nom: **WHILE ... DO ... END_WHILE**
Fonction: Enoncé d'itération avec test en début de boucle.
Syntaxe:

```
WHILE <expression_booleenne> DO
  <enonce> ;
  <enonce> ;
  ...
END_WHILE ;
```

Attention: ISaGRAF étant un système **synchrone**, les variables d'entrées ne sont pas rafraîchies pendant les itérations d'une boucle WHILE. Il ne faut jamais tester l'évolution d'une variable d'entrée dans la condition d'une structure WHILE.

Exemple:

```
(* Programme ST avec énoncé WHILE *)
(* ce programme utilise des procédures "C" spécifiques pour lire *)
(* des caractères sur un port de communication série *)
string:= ''; (* chaîne vide *)
nbchar:= 0;
WHILE ((nbchar < 16) & ComIsReady ( )) DO
        string:= string + ComGetChar ( );
        nbchar:= nbchar + 1;
END_WHILE;
```

B.7.5.6 Enoncé "REPEAT"

Nom: **REPEAT ... UNTIL ... END_REPEAT**
Fonction: Enoncé d'itération avec test d'itération en fin de boucle.
Syntaxe:

```
REPEAT
  <enonce> ;
  <enonce> ;
```

```
    ...
UNTIL <condition_booleenne>
END_REPEAT ;
```

Attention: ISaGRAF étant un système **synchrone**, les variables d'entrées ne sont pas rafraîchies pendant les itérations d'une boucle REPEAT. Il ne faut jamais tester l'évolution d'une variable d'entrée dans la condition d'une structure REPEAT.

Exemple:

```
(* Programme ST avec énoncé REPEAT *)
(* ce programme utilise des procédures "C" spécifiques pour lire      *)
(* des caractères sur un port de communication série                  *)
string:= ''; (* chaîne vide *)
nbchar:= 0;
IF ComIsReady ( ) THEN
        REPEAT
                string:= string + ComGetChar ( );
                nbchar:= nbchar + 1;
        UNTIL ( (nbchar >= 16) OR NOT (ComIsReady ( )) )
        END_REPEAT;
END_IF;
```

B.7.5.7 Enoncé "FOR"

Nom:	**FOR ... TO ... BY ... DO ... END_FOR**	
Fonction:	Structure d'itération utilisant une variable d'index de type analogique entier	
Syntaxe:	**FOR <index>:= <mini> TO <maxi> BY <pas>** **DO** <enonce> ; <enonce> ; **END_FOR;**	
Opérandes:	index:	variable analogique interne incrémentée à chaque itération
	mini:	valeur initiale de la variable d'index (valeur lors de la première itération)
	maxi:	valeur maximum autorisée pour l'index
	pas:	pas d'incrémentation de l'index (à chaque itération)

La mention [BY pas] est optionnelle. Lorsqu'il n'est pas spécifié, le pas d'incrément par défaut est 1.

Attention: ISaGRAF étant un système **synchrone**, les variables d'entrées ne sont pas rafraîchies pendant les itérations d'une boucle FOR.

Voici l'équivalent "While" d'un énoncé FOR:

```
index:= mini;
while (index <= maxi) do
```

```
    <enonce> ;
    <enonce> ;
    index:= index + pas;
  end_while;
```

Exemple:

```
(* Programme ST avec énoncé FOR *)
(* Ce programme extrait les caractères "chiffres" d'une chaîne *)
length:= mlen (message);
target:= ''; (* chaîne vide *)
FOR index:= 1 TO length BY 1 DO
        code:= ascii (message, index);
        IF (code >= 48) & (code <= 57) THEN
                target:= target + char (code);
        END_IF;
END_FOR;
```

B.7.5.8 Enoncé "EXIT"

Nom: **EXIT**
Fonction: quitte une boucle d'itération FOR, WHILE ou REPEAT
Syntaxe: **EXIT;**

L'énoncé EXIT est généralement utilisé dans un énoncé de sélection IF, à l'intérieur d'un bloc FOR, WHILE ou REPEAT.

Exemple:

```
(* Programme ST avec énoncé EXIT *)
(* ce programme recherche un caractère dans une chaîne *)

length:= mlen (message);
found:= NO;
FOR index:= 1 TO length BY 1 DO
        code:= ascii (message, index);
        IF (code  = searched_char) THEN
                found:= YES;
                EXIT;
        END_IF;
END_FOR;
```

B.7.6 Extension au langage ST

Les énoncés et fonctions suivants sont des extensions du langage ST.
- TSTART / TSTOP: Gestion de temporisations

Les énoncés suivants permettent à un programme SFC père, dans ses actions, de contrôler un **programme SFC fils**. Ils peuvent être utilisés dans les blocs ACTION(): ... END_ACTION; des étapes SFC.
- GSTART lance un programme SFC

- GKILL tue un programme SFC
- GFREEZE fige un programme SFC
- GRST relance un programme SFC
- GSTATUS rend l'état d'un programme SFC

Attention: Ces fonctions ne font pas partie de la norme CEI 1131-3.
Voici des équivalents simples pour les énoncés GSTART et GKILL dans les étapes SFC:

```
sfc_fils(S); (* équivalent à GSTART(sfc_fils); *)
sfc_fils(R); (* équivalent à GKILL(sfc_fils); *)
```

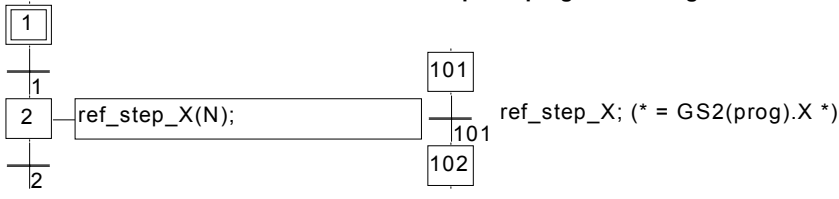
Les champs suivants d'une étape SFC peuvent être testés dans des expressions:

GSnnn.x est une valeur booléenne qui représente **l'activité de l'étape**
GSnnn.t est une valeur temporelle qui représente le temps écoulé depuis que l'étape est active: **durée d'activation**.
("**nnn**" est le numéro de référence de l'étape SFC)

Il est également possible de tester l'activité d'une étape déclarée dans un autre programme SFC, en utilisant la syntaxe suivante:
GSnnn(nomprog).x

Attention: référencer une étape d'un autre programme, en utilisant cette syntaxe de fait pas partie de la norme CEI 1131-3. Une façon simple d'obtenir le même résultat en respectant la norme, est de déclarer une variable booléenne globale dans le dictionnaire. Cette variable représentera l'activité de l'étape à tester (par exemple ref_step_X). Ensuite vous insérez cette variable dans l'étape avec le qualificatif N (ref_step_X(N);). Puis dans le programme qui a besoin de tester l'activité de l'étape, vous utilisez la variable.

programme Prog **programme qui a besoin de tester l'activité de l'étape du programme Prog**



B.7.6.1 Enoncé "TSTART"

Nom: **TSTART**
Fonction: Valide le rafraîchissement automatique d'une tempo.
La valeur courante de la temporisation n'est pas modifiée par l'énoncé TSTART, i.e. le comptage démarre à partir de la valeur courante.
Syntaxe: **TSTART (<variable_tempo>);**
Opérandes: toute variable temporisation inactive
Retour: (aucun)

Exemple:

```
(* Exemple d'utilisation des énoncés TSTART et TSTOP *)
```

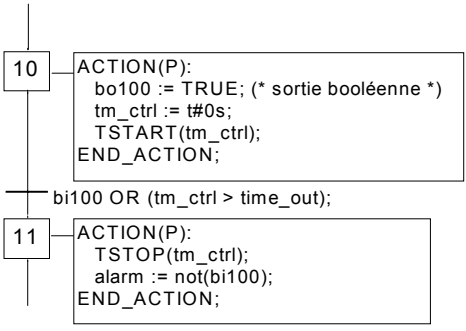
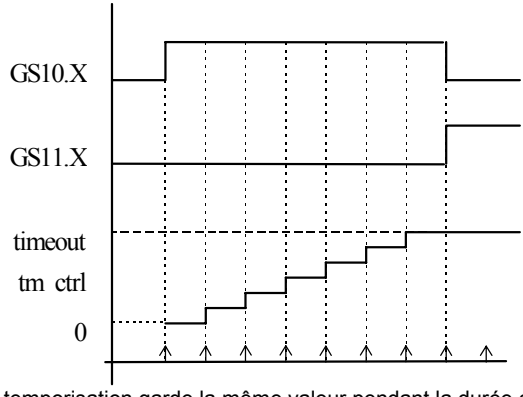


Diagramme de temps si bi100 est toujours à FALSE:



La temporisation garde la même valeur pendant la durée du cycle.

B.7.6.2 Enoncé "TSTOP"

Nom:	**TSTOP**
Fonction:	Stoppe le rafraîchissement automatique d'une tempo. La valeur de la variable n'est pas modifiée par la commande TSTOP.
Syntaxe:	**TSTOP (<variable_tempo>);**
Opérandes:	toute variable temporisation activée par TSTART
Retour:	(aucun)

Exemple: Voir TSTART (fonction décrite ci-dessus)

B.7.6.3 ENONCE "GSTART"

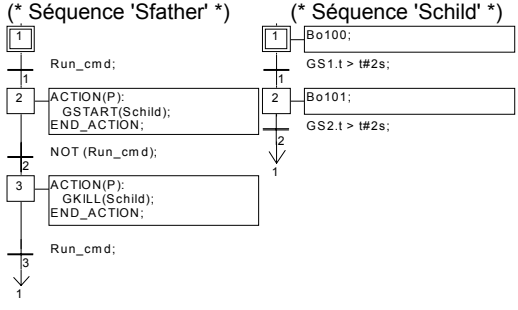
Nom:	**GSTART**
Fonction:	Lance un programme SFC fils en plaçant un jeton dans chacune de ses étapes initiales
Syntaxe:	**GSTART (<programme_fils>);**

Opérandes: Le programme SFC spécifié doit être le fils du programme incluant l'énoncé GSTART.
Retour: (aucun)

Les fils du programme lancé par GSTART ne sont pas automatiquement lancés avec lui.
Remarque: comme GSTART ne fait pas partie de la norme CEI 1131-3, préférez l'utilisation du qualificatif S et la syntaxe suivante pour lancer un programme SFC fils:

```
SFC_fils(S);
```

Exemple d'utilisation de GSTART et GKILL:



B.7.6.4 Enoncé "GKILL"

Nom: **GKILL**
Fonction: Tue un programme SFC fils en supprimant tous les jetons existant dans ses étapes
Syntaxe: **GKILL (<programme_fils>);**
Opérandes: Le programme SFC spécifié doit être le fils du programme incluant l'énoncé GKILL.
Retour: (aucun)

Les fils du programme tué sont automatiquement tués avec lui.
Remarque: comme GKILL ne fait pas partie de la norme CEI 1131-3, préférez l'utilisation du qualificatif R et la syntaxe suivante;

```
SFC_fils(R);
```

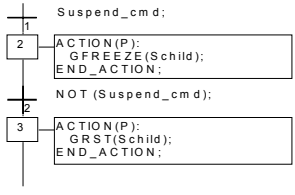
Exemple: voir GSTART (fonction décrite ci-dessus).

B.7.6.5 Enoncé "GFREEZE"

Nom: **GFREEZE**
Fonction: Suspend l'exécution d'un programme fils SFC. Le programme figé pourra être relancé en utilisant la commande GRST.
Syntaxe: **GFREEZE (<programme_fils>);**
Opérandes: Le programme SFC spécifié doit être le fils du programme incluant l'énoncé GFREEZE.
Retour: (aucun)

Les fils du programme fils sont automatiquement "gelés" en même temps que le programme spécifié.
Remarque: GFREEZE ne fait pas partie de la norme CEI 1131-3.

Exemple:



B.7.6.6 Enoncé "GRST"

Nom:	**GRST**
Fonction:	Relance un programme SFC fils figé par une commande GFREEZE.
Syntaxe:	**GRST (<programme_fils>);**
Opérandes:	Le programme SFC spécifié doit être le fils du programme incluant l'énoncé GRST.
Retour:	(aucun)

Les fils du programme fils sont automatiquement "relancés" en même temps que le programme spécifié.
Remarque: GRST ne fait pas partie de la norme CEI 1131-3.

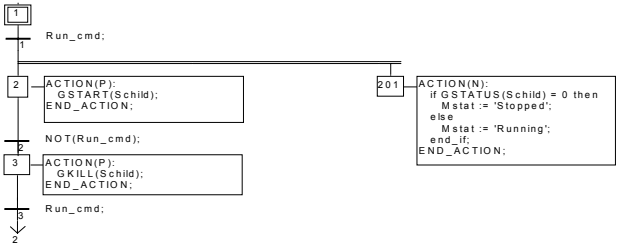
Exemple: voir GFREEZE (fonction décrite ci-dessus)

B.7.6.7 Enoncé "GSTATUS"

Nom:	**GSTATUS**
Fonction:	Rend l'état courant d'un programme SFC.
Syntaxe:	**<var_ana>:= GSTATUS (<programme_fils>);**
Opérandes:	Le programme SFC spécifié doit être le fils du programme incluant l'appel à GSTATUS.
Retour:	**0** = le programme est **inactif** (tué) **1** = le programme est **actif** (lancé) **2** = le programme est **figé**

Remarque: GSTATUS ne fait pas partie de la norme CEI 1131-3.

Exemple:


1
Run_cmd;
1
2
ACTION(P):
GSTART(Schild);
END_ACTION;
201
ACTION(N):
if GSTATUS(Schild) = 0 then
Mstat := 'Stopped';
else
Mstat := 'Running';
end_if;
END_ACTION;
NOT(Run_cmd);
2
3
ACTION(P):
GKILL(Schild);
END_ACTION;
Run_cmd;
3
2

B.8 Langage IL

Le langage **IL** (Instruction List), est un langage textuel de bas niveau. Il est particulièrement adapté aux applications de petite taille, ou à l'optimisation de portions réduites d'une application. Les instructions travaillent toujours sur un **résultat courant** (ou **registre IL**). L'opérateur indique le type d'opération à effectuer entre le résultat courant et l'opérande. Le résultat de l'opération est stocké à son tour dans le résultat courant.

B.8.1 Syntaxe du langage IL

Un programme IL est une liste d'**instructions**. Chaque instruction doit commencer par une ligne nouvelle, et doit contenir un **opérateur**, complété éventuellement par des **modificateurs** et, si c'est nécessaire pour l'opération, un ou plusieurs **opérandes**, séparés par des virgules (**','**). Une **étiquette** suivie de deux points (**':'**) peut précéder l'instruction. Si un **commentaire** est attaché à l'instruction, il doit être le dernier élément de la ligne. Un commentaire commence toujours par les caractères **'(*'** et se termine par **'*)'**. Des lignes vides peuvent être insérées entre des instructions. Un commentaire peut être posé sur une ligne sans instruction.

Exemples de ligne d'instruction:

Etiquette	*Opérateur*	*Opérande*	*Commentaire*
Debut:	**LD**	**IX1**	**(* bouton poussoir *)**
	ANDN	**MX5**	**(* commande valide *)**
	ST	**QX2**	**(* lance moteur *)**

B.8.1.1 Etiquettes

Une **étiquette** suivie du caractère **':'** peut précéder une instruction. Une étiquette peut être posée sur une ligne sans instruction. Les étiquettes sont utilisées comme opérandes par certaines instructions telles que les sauts. La nomenclature des étiquettes doit respecter les règles suivantes:
- la longueur du nom ne doit pas excéder **16** caractères
- le premier caractère doit être une **lettre**
- les caractères suivants doivent être des **lettres**, des **chiffres** ou **'_'**

Deux étiquettes d'un même programme IL ne peuvent pas avoir le même nom. Une étiquette peut avoir le même nom qu'une variable.

B.8.1.2 Modificateurs d'instructions

Voici la liste des **modificateurs** autorisés pour les instructions du langage IL. Le caractère modificateur complète le nom de l'instruction, sans aucun caractère de séparation.

N inversion booléenne de l'opérande
(opération différée
C opération conditionnelle

Le modificateur **'N'** indique que l'opérande doit être inversé avant d'être utilisé par l'instruction. Par exemple, l'instruction **ANDN IX12** est interprétée comme: **resultat:= resultat AND NOT (IX12)**
Le modificateur **parenthèse** ouvrante **'('** indique que l'évaluation de l'instruction doit être différée jusqu'à la prochaine instruction parenthèse fermante **')'**.
Le modificateur **'C'** indique que l'instruction ne doit être exécutée que si le résultat courant a la valeur booléenne TRUE (ou différent de 0 pour une valeur non booléenne). Le modificateur **'C'** peut être combiné avec l'opérateur **'N'** pour indiquer que l'instruction ne doit être exécutée que si le résultat courant vaut FALSE (ou 0 pour une instruction non booléenne).

B.8.1.3 Opérations différées

Parce que le langage IL ne traite qu'un seul registre (**résultat courant**), certaines opérations doivent être différées, pour changer l'ordre naturel d'exécution des instructions. Les parenthèses sont utilisées pour représenter les **opérations différées**.

'('	est un modificateur	indique que l'opération doit être différée
')'	est une instruction	exécute l'opération différée

Le modificateur **parenthèse** ouvrante **'('** indique que l'évaluation de l'instruction doit être différée jusqu'à la prochaine instruction parenthèse fermante **')'**. Par exemple, la séquence:

```
AND(        IX12
OR          IX35
)
```

est interprétée:

```
resultat:= resultat AND ( IX12 OR IX35 );
```

B.8.2 Opérateur du IL

La table suivante résume l'ensemble des opérateurs du langage IL:

opérateur	modificateurs	opérande	description
LD	N	variable, constante	charge l'opérande
ST	N	variable	stocke le résultat courant
S		variable BOO	forçage à TRUE
R		variable BOO	forçage à FALSE
AND	N (	BOO	ET logique
&	N (	BOO	ET logique
OR	N (	BOO	OU logique
XOR	N (	BOO	OU exclusif
ADD	(	variable, constante	addition
SUB	(	variable, constante	soustraction
MUL	(	variable, constante	multiplication
DIV	(	variable, constante	division

GT GE EQ LE LT NE	((((((	variable, constante variable, constante variable, constante variable, constante variable, constante variable, constante	test >= test >= test = test <= test < test <>
CAL JMP RET	C N C N C N	instance de bloc fonc. étiquette	appel un bloc fonctionnel saut à une étiquette retour de sous-programme
)			exécute instruction différée

Dans les sections qui suivent seuls les opérateurs spécifiques au IL sont décrits. Les autres opérateurs utilisables sont détaillés dans le chapitre "Opérateurs, blocs fonctionnels et fonctions standard".

B.8.2.1 Opérateur "LD"

Opération Charge une valeur dans le résultat courant
Modificateurs N
Opérande Expression constante
Variable interne d'entrée ou de sortie

Exemple:

```
                                (* EXEMPLES D'OPERATIONS LD *)
LDex:       LD      false       (* resultat:= constante booléenne FALSE *)
            LD      true        (* resultat:= constante booléenne TRUE *)
            LD      123         (* resultat:= constante analogique entière*)
            LD      123.1       (* resultat:= constante analogique réelle *)
            LD      t#3s        (* resultat:= constante tempo *)
            LD      boo_var1    (* resultat:= variable booléenne *)
            LD      ana_var1    (* resultat:= variable analogique *)
            LD      tmr_var1    (* resultat:= variable temporisation *)
            LDN     boo_var2    (* resultat:= NOT ( variable booléenne ) *)
```

B.8.2.2 Opérateur "ST"

Opération Stocke le résultat courant dans une variable
Le résultat courant n'est pas modifié par cette instruction
Modificateurs N
Opérande Variable interne ou de sortie
Exemple:

```
                                (* EXEMPLES D'OPERATIONS ST *)
STboo:      LD      false
            ST      boo_var1    (* boo_var1:= FALSE *)
            STN     boo_var2    (* boo_var2:= TRUE *)
STana:      LD      123
            ST      ana_var1    (* ana_var1:= 123 *)
STtmr:      LD      t#12s
            ST      tmr_var1    (* tmr_var1:= t#12s *)
```

B.8.2.3 Opérateur "S"

Opération Force une variable booléenne à TRUE, si le résultat courant vaut TRUE. Aucune opération n'est effectuée si le résultat courant vaut FALSE. Le résultat courant n'est pas modifié par cette instruction.
Modificateurs (aucun)
Opérande Variable booléenne interne ou de sortie

Exemple:

```
                                (* EXEMPLES D'OPERATIONS S *)
SETex:      LD      true        (* résultat courant:= TRUE *)
            S       boo_var1    (* boo_var1:= TRUE *)
                                (* résultat courant inchangé *)
            LD      false       (* résultat courant:= FALSE *)
            S       boo_var1    (* aucune action - boo_var1 inchangé *)
```

B.8.2.4 Opérateur "R"

Opération Force une variable booléenne à FALSE, si le résultat courant vaut TRUE. Aucune opération n'est effectuée si le résultat courant vaut FALSE. Le résultat courant n'est pas modifié par cette instruction.
Modificateurs (aucun)
Opérande Variable booléenne interne ou de sortie

Exemple:

```
                                (* EXEMPLES D'OPERATIONS R *)
RESETex:    LD      true        (* résultat courant:= TRUE *)
            R       boo_var1    (* boo_var1:= FALSE *)
                                (* résultat courant inchangé *)
            ST      boo_var2    (* boo_var2:= TRUE *)
            LD      false       (* résultat courant:= FALSE *)
            R       boo_var1    (* aucune action - boo_var1 inchangé *)
```

B.8.2.5 Opérateur "JMP"

Opération Saut à l'instruction repérée par une étiquette
Modificateurs C N
Opérande Etiquette définie dans le même programme IL

Exemple:

```
(* l'exemple suivant teste la valeur d'un sélecteur analogique (0 ou 1 ou 2) *)
(* et force une parmi 3 variables booléennes. Le test "égal à 0" est réalisé *)
(* par l'instruction JMPC *)
JMPex:      LD      selector    (* selector = 0 ou 1 ou 2 *)
            BOO                 (* conversion en booléen *)
            JMPC    test1       (* si selector = 0 alors *)
            LD      true
            ST      bo0         (* bo0:= true *)
```

```
            JMP     JMPend  (* fin du programme *)
test1:      LD      selector (* décrémente selector *)
            SUB     1       (* selector vaut 0 ou 1 *)
            BOO             (* conversion en booléen *)
            JMPC    test2   (* si selector = 0 alors *)
            LD      true
            ST      bo1     (* bo1:= true *)
            JMP     JMPend  (* fin du programme *)
                            (* dernière possibilité *)
test2:      LD      true
            ST      bo2     (* bo2:= true *)
JMPend:                     (* fin du programme *)
```

B.8.2.6 Opérateur "RET"

Opération Termine l'exécution du programme. Si la séquence IL est un sous-programme, le résultat courant sera la valeur retournée au programme appelant.

Modificateurs C N

Opérande (aucun)

Exemple:

```
(* l'exemple suivant teste la valeur d'un sélecteur analogique (0 ou 1 ou 2) *)
(* et force une parmi 3 variables booléennes. Le test "égal à 0" est réalisé *)
(* par l'instruction JMPC *)
RETex:      LD      selector (* selector = 0 ou 1 ou 2 *)
            BOO             (* conversion en booléen *)
            JMPC    test1   (* si selector = 0 alors *)
            LD      true
            ST      bo0     (* bo0:= true *)
            RET             (* fin - retourne 0 *)
test1:      LD      selector (* décrémente selector *)
            SUB     1       (* selector = 0 ou 1 *)
            BOO             (* conversion en booléen *)
            JMPC    test2   (* si selector = 0 alors *)
            LD      true
            ST      bo1     (* bo1:= true *)
            LD      1       (* valeur de selector *)
            RET             (* fin - retourne 1 *)
                            (* dernière possibilité *)
test2:      RETNC           (* retour si selector contient *)
                            (* une valeur invalide *)
            LD      true
            ST      bo2     (* bo2:= true *)
            LD      2       (* valeur de selector *)
                            (* fin - retourne 2 *)
```

B.8.2.7 Opérateur ")"

Opération Exécute une opération différée

	L'opération différée a été modifiée par "("
Modificateurs	(aucun)
Opérande	(aucun)

Exemple:

```
(* Exemple d'opérations différées imbriquées *)
(* res:= a1 + (a2 * (a3 - a4) * a5) + a6; *)
Delayed:    LD      a1      (* résultat:= a1; *)
            ADD(    a2      (* ADD différé - résultat:= a2; *)
            MUL(    a3      (* MUL différé - résultat:= a3; *)
            SUB     a4      (* résultat:= a3 - a4; *)
            )               (* exécute MUL différé - résultat:= a2 * (a3-a4); *)
            MUL     a5      (* résultat:= a2 * (a3 - a4) * a5; *)
            )               (* exécute ADD différé *)
                            (* résultat:= a1 + (a2 * (a3 - a4) * a5); *)
            ADD     a6      (* résultat:= a1 + (a2 * (a3 - a4) * a5) + a6; *)
            ST      res     (* res:= résultat courant *)
```

B.8.2.8 Appel de sous-programme ou fonction

Pour appeler un sous-programme ou une fonction (écrit en langage IL, ST, LD ou FBD) depuis un programme IL, il suffit d'utiliser son nom comme une opération.

Opération	Exécute un sous-programme ou une fonction. La valeur retournée par le sous-programme ou la fonction est chargée dans le résultat courant.
Modificateurs	(aucun)
Opérande	Le premier paramètre d'appel doit être stocké dans le résultat courant. Les paramètres suivants sont exprimés dans le champ opérande, séparés par des virgules.

Exemple:

```
(* Appel de sous-programme: conversion analogique > temporisation *)
Main:       LD      bi0
            SUBPRO  bi1,bi2 (* Appel sous-programme pour avoir la valeur ana *)
            ST      result  (* resultat:= valeur rendue par le sous-programme *)
            GT      vmax    (* test de dépassement *)
            RETC            (* retour si dépassement *)
            LD      result
            MUL     1000    (* conversion secondes > milli-secondes *)
            TMR             (* conversion en temporisation *)
            ST      tmval   (* stockage dans une variable temporisation *)

(* Sous-programme nommé 'SUBPRO': évalue la valeur analogique *)
(* valeur binaire codée sur trois bits donnés par trois variables d'entrée: in0, in1, in2. Ce sont
les paramètres d'entrée du sous-programme *)
            LD      in2
            ANA             (* result = ana (in2); *)
            MUL     2       (* result:= 2*ana (in2); *)
```

```
ST      temporary  (* temporary:= result *)
LD      in1
ANA
ADD     temporary  (* result:= 2*ana (in2) + ana (in1); *)
MUL     2          (* result:= 4*ana (in2) + 2*ana (in1); *)
ST      temporary  (* temporary:= result *)
LD      in0
ANA
ADD     temporary  (* result:= 4*ana (in2) + 2*ana (in1)+ana (in0); *)
ST      SUBPRO     (* retourne le résultat courant *)
```

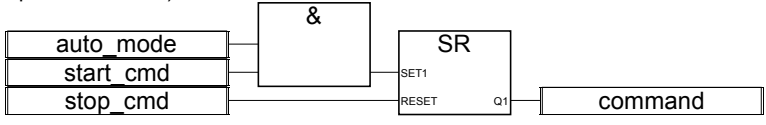
B.8.2.9 Appel de bloc fonctionnel: opérateur CAL

Operation appel un bloc fonctionnel
Modificateur C N
Opérande Nom de l'instance du bloc fonctionnel
Vous devez assignez les paramètres d'entrée du bloc avant de l'appeler en utilisant les séquences d'opérations LD/ST.
Les paramètres de sortie seront connus s'ils sont utilisés.

Exemple1:

```
(* Appel du bloc fonctionnel SR: SR1 est une instance de SR *)
LD      auto_mode
AND     start_cmd
ST      SR1.set1
LD      stop_cmd
ST      SR1.reset
CAL     SR1
LD      SR1.Q1
ST      command
```

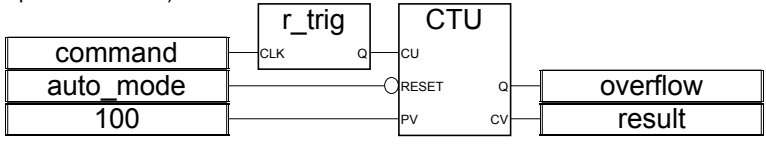
(* Equivalent FBD: *)



Exemple 2

```
(* R_TRIG1 est une instance du bloc R_TRIG et CTU1 est une instance du bloc CTU *)
LD      command
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ST      CTU1.cu
LDN     auto_mode
ST      CTU1.reset
LD      100
ST      CTU1.pv
CAL     CTU1
```

```
LD      CTU1.Q
ST      overflow
LD      CTU1.cv
ST      result

(* Equivalent FBD: *)
```



B.9 Opérateurs, fonctions et blocs fonctionnels

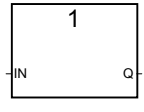
B.9.1 OPERATEURS STANDARDS

Voici la liste des boîtes fonctions disponibles, correspondant aux opérateurs et fonctions définies en standard par le langage **ST**:

- Manipulation de données.....Assignation, Négation analogique
- Opérations booléennes.....ET logique
  OU logique
  OU exclusif logique
- Opérations arithmétiques.....Addition
  Soustraction
  Multiplication
  Division
- Opérations de masquage.....Masque ET analogique bit à bit
  Masque OU analogique bit à bit
  Masque OU exclusif analogique bit à bit
  Inversion bit à bit
- Comparaisons .....Plus petit que
  Plus petit ou égal à
  Plus grand que
  Plus grand ou égal à
  Egal à
  Différent de
- Conversion de données .....Conversion en booléen
  Conversion en analogique entier
  Conversion en analogique réel
  Conversion en temporisation
  Conversion en message
- Autres.....Concaténation de messages
  Accès aux paramètres du système
  Traitement des E/S

Les notations suivantes sont utilisés pour le type des paramètres:
BOO: booléen
ENT: analogique entier
REEL: analogique réel
TMP: temporisation
MSG: message

1 gain



Arguments:

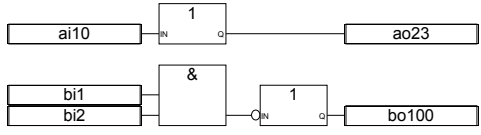
IN	tout type
Q	tout type

Description:

assignation d'une variable dans une autre

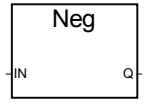
Ce bloc est très utile pour relier directement une entrée et une sortie du diagramme. Il peut également être utilisé pour insérer une inversion booléenne avant la connexion d'un arc à une variable de sortie du diagramme.

(* Programme FBD avec blocs "assignation" *)




```
(* Equivalence ST: *)
ao23:= ai10;
bo100:= NOT (bi1 AND bi2);

(* Equivalence IL: *)
LD      ai10
ST      ao23
LD      bi1
AND     bi2
STN     bo100
```

NEG



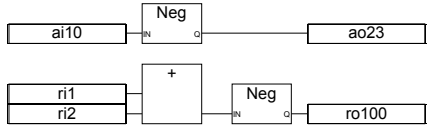
Arguments:

IN	ENT-REEL	L'entrée et la sortie doivent avoir le même format
Q	ENT-REEL	

Description:

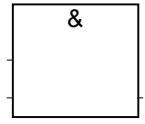
Assignation de la négation d'une variable.

(* Programme FBD avec blocs de négation *)




```
(* Equivalence ST: *)
ao23:= - (ai10);
ro100:= - (ri1 + ri2);

(* Equivalence IL: *)
LD          ai10
MUL         -1
ST          ao23
LD          ri1
ADD         ri2
MUL         -1.0
ST          ro100
```

& AND



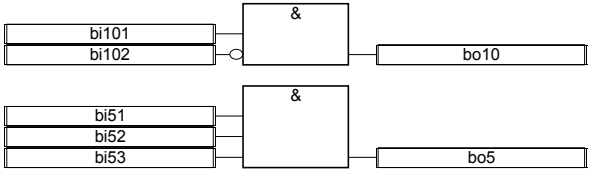
Remarque: Vous pouvez étendre le nombre d'entrées de cet opérateur à plus de deux.

Arguments:

(inputs)	BOO	
output	BOO	ET logique entre les termes en entrée

Description:
ET logique entre deux ou plusieurs termes.

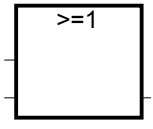
```
(* Programme FBD avec blocs "ET" *)
```




```
(* Equivalence ST: *)
bo10:= bi101 AND NOT (bi102);
bo5:= (bi51 AND bi52) AND bi53;

(* Equivalence IL: *)
            LD          bi101       (* résultat courant:= bi101 *)
```

```
ANDN    bi102    (* résultat courant:= bi101 AND not(bi102) *)
ST      bo10     (* bo10:= résultat courant *)
LD      bi51     (* résultat courant:= bi51;
&       bi52     (* résultat courant:= bi51 AND bi52 *)
&       bi53     (* résultat courant:= (bi51 AND bi52) AND bi53 *)
ST      bo5      (* bo5:= résultat courant *)
```

>=1 OR



Remarque: Vous pouvez étendre le nombre d'entrées de cet opérateur à plus de deux.

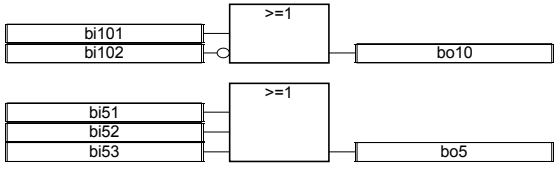
Arguments:

(inputs)	BOO	
output	BOO	OU logique entre les termes en entrée

Description:

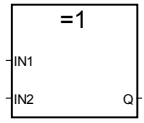
OU logique entre deux ou plusieurs termes.

(* Programme FBD avec blocs "OU" *)




```
(* Equivalence ST: *)
bo10:= bi101 OR NOT (bi102);
bo5:= (bi51 OR bi52) OR bi53;

(* Equivalence IL: *)
LD      bi101
ORN     bi102
ST      bo10
LD      bi51
OR      bi52
OR      bi53
ST      bo5
```

=1 XOR

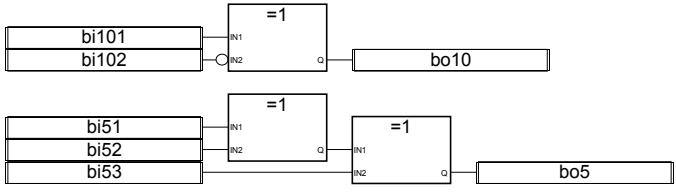


Arguments:

IN1	BOO	
IN2	BOO	
Q	BOO	OU logique exclusif des deux termes en entrée

Description:
OU logique exclusif entre deux termes.

(* Programme FBD avec blocs "OU Exclusif" *)

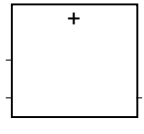


(* Equivalence ST: *)
bo10:= bi101 XOR NOT (bi102);
bo5:= (bi51 XOR bi52) XOR bi53;

(* Equivalence IL: *)

LD	bi101
XORN	bi102
ST	bo10
LD	bi51
XOR	bi52
XOR	bi53
ST	bo5

+



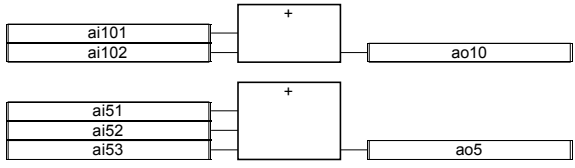
Remarque: Vous pouvez étendre le nombre d'entrées de cet opérateur à plus de deux.

Arguments:

(entrées)	ENT-REEL	les entrées peuvent être entières ou réelles (toutes les entrées doivent avoir le même format)
sortie	ENT-REEL	addition signée des termes en entrée

Description:
Addition de deux ou plusieurs termes analogiques.

(* Programme FBD avec blocs "addition" *)

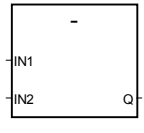


(* Equivalence ST *)
ao10:= ai101 + ai102;
ao5:= (ai51 + ai52) + ai53;

(* Equivalence IL *)

LD	ai101
ADD	ai102
ST	ao10
LD	ai51
ADD	ai52
ADD	ai53
ST	ao5

-

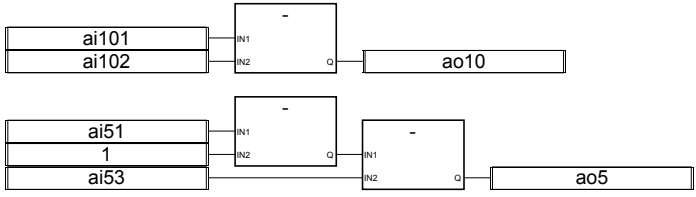


Arguments:

IN1	ENT-REEL	peut être entier ou réel
IN2	ENT-REEL	(IN1 et IN2 doivent avoir le même format)
Q	ENT-REEL	soustraction (IN1 - IN2)

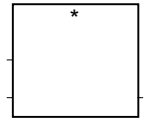
Description:
Soustraction de deux termes analogiques (premier - second).

(* Programme FBD avec blocs "soustraction" *)




```
(* Equivalence ST *)
ao10:= ai101 - ai102;
ao5:= (ai51 - 1) - ai53;

(* Equivalence IL *)
LD          ai101
SUB         ai102
ST          ao10
LD          ai51
SUB         1
SUB         ai53
ST          ao5
```

*



Remarque: Vous pouvez étendre le nombre d'entrées de cet opérateur à plus de deux.

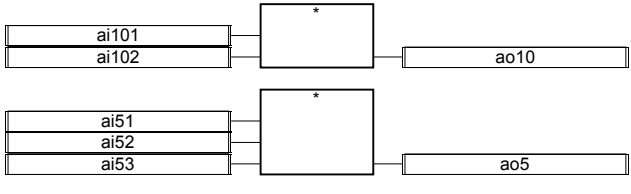
Arguments:

(entrées)	ENT-REEL	les entrées peuvent être entières ou réelles (toutes les entrées doivent avoir le même format)
sortie	ENT-REEL	multiplication signée des termes en entrée

Description:

Multiplication de deux ou plusieurs termes analogiques.

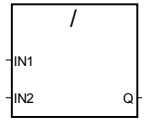
(* Programme FBD avec blocs "multiplication" *)




```
(* Equivalence ST *)
ao10:= ai101 * ai102;
```

```
ao5:= (ai51 * ai52) * ai53;

(* Equivalence IL *)
LD          ai101
MUL         ai102
ST          ao10
LD          ai51
MUL         ai52
MUL         ai53
ST          ao5
```

/



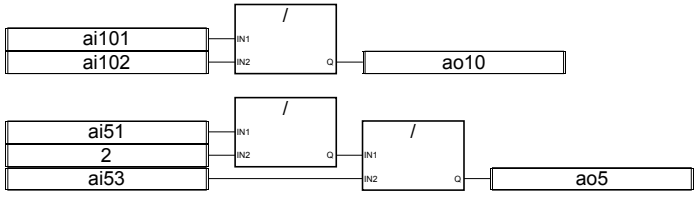
Arguments:

IN1	ENT-REEL	peut être entier ou réel (opérande)
IN2	ENT-REEL	valeur analogique non nulle (diviseur) (IN1 et IN2 doivent avoir le même format)
Q	ENT-REEL	division signée, entière ou réelle, de IN1 par IN2.

Description:

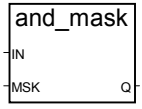
Division de deux variables analogiques (la première divisé par la seconde).

(* Programme FBD avec blocs "division" *)




```
(* Equivalence ST *)
ao10:= ai101 / ai102;
ao5:= (ai5 / 2) / ai53;

(* Equivalence IL *)
LD          ai101
DIV         ai102
ST          ao10
LD          ai51
DIV         2
DIV         ai53
ST          ao5
```

AND_MASK



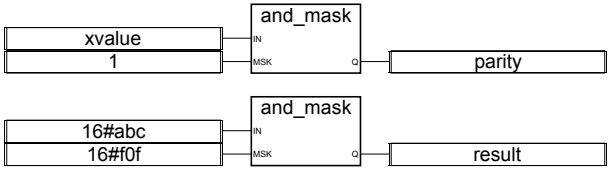
Arguments:

IN	ENT	doit avoir le format entier
MSK	ENT	doit avoir le format entier
Q	ENT	ET logique bit à bit entre IN et MSK

Description:

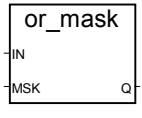
Masque ET bit à bit entre deux entiers analogiques.

(* Programme FBD avec bloc "Masque ET" *)




```
(* Equivalence ST: *)
parity:= AND_MASK (xvalue, 1); (* 1 si xvalue est impair *)
result:= AND_MASK (16#abc, 16#f0f); (* égal 16#a0c *)

(* Equivalence IL: *)
LD          xvalue
AND_MASK    1
ST          parity
LD          16#abc
AND_MASK    16#f0f
ST          result
```

OR_MASK



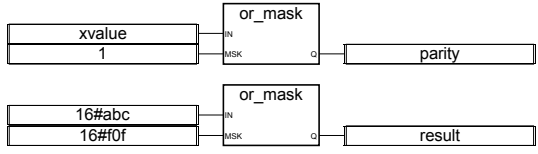
Arguments:

IN	ENT	doit avoir le format entier
MSK	ENT	doit avoir le format entier
Q	ENT	OU logique bit à bit entre IN et MSK

Description:

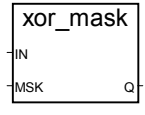
Masque OU bit à bit entre deux entiers analogiques.

(* Programme FBD avec bloc "Masque OU" *)




```
(* Equivalence ST: *)
is_odd:= OR_MASK (xvalue, 1); (* toujours impair *)
result:= OR_MASK (16#abc, 16#f0f); (* égal 16#fbf *)

(* Equivalence IL: *)
LD          xvalue
OR_MASK     1
ST          is_odd
LD          16#abc
OR_MASK     16#f0f
ST          result
```

XOR_MASK



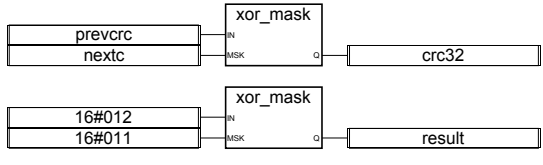
Arguments:

IN	ENT	doit avoir le format entier
MSK	ENT	doit avoir le format entier
Q	ENT	OU exclusif logique bit à bit entre IN et MSK

Description:

Masque OU exclusif bit à bit entre deux entiers analogiques.

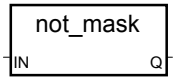
(* Programme FBD avec bloc "Masque OU Exclusif" *)




```
(* Equivalence ST: *)
crc32:= XOR_MASK (prevcrc, nextc);
result:= XOR_MASK (16#012, 16#011); (* égal 16#003 *)

(* Equivalence IL: *)
```

```
LD          prevcrc
XOR_MASK    nextc
ST          crc32
LD          16#012
XOR_MASK    16#011
ST          result
```

NOT_MASK



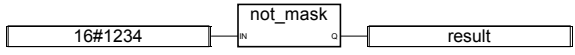
Arguments:

IN	ENT	doit avoir le format entier
Q	ENT	inversion logique bit à bit de IN (sur 32 bits)

Description:

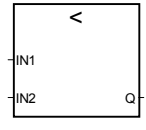
Masque inversion logique bit à bit d'un entier analogique.

(* Programme FBD avec bloc "NOT_MASK" *)




```
(* Equivalence ST: *)
result:= NOT_MASK (16#1234);
(* result = 16#FFFF_EDCB *)

(* Equivalence IL: *)
LD          16#1234
NOT_MASK
ST          result
```

<



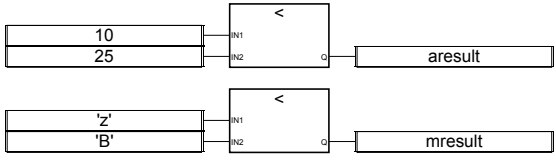
Arguments:

IN1	ENT-REEL-TMP-MSG	
IN2	ENT-REEL-TMP-MSG	les deux entrées doivent être du même type
Q	BOO	TRUE si IN1 < IN2

Description:

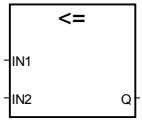
Teste si une valeur est plus petite (Less Than) qu'une autre (sur les analogiques, les temporisations ou les messages)

(* Programme FBD avec bloc "Plus petit que" *)




```
(* Equivalence ST: *)
aresult:= (10 < 25); (* aresult = TRUE *)
mresult:= ('z' < 'B'); (* mresult = FALSE *)

(* Equivalence IL: *)
LD          10
LT          25
ST          aresult
LD          'z'
LT          'B'
ST          mresult
```

<=



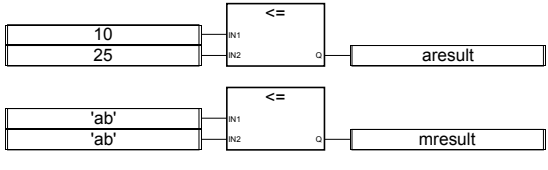
Arguments:

IN1	ENT-REEL-MSG	
IN2	ENT-REEL-MSG	les deux entrées doivent avoir le même type
Q	BOO	TRUE si IN1 <= IN2

Description:

Teste si une valeur est plus petite ou égale (Less or Equal) à une autre (sur les analogiques, ou les messages)

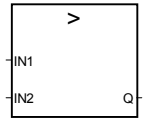
(* Programme FBD avec bloc "Plus petit ou égal" *)




```
(* Equivalence ST: *)
```

```
aresult:= (10 <= 25); (* aresult = TRUE *)
mresult:= ('ab' <= 'ab'); (* mresult = TRUE *)

(* Equivalence IL: *)
LD      10
LE      25
ST      aresult
LD      'ab'
LE      'ab'
ST      mresult
```

>



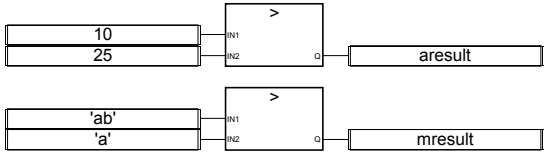
Arguments:

IN1	ENT-REEL-TMP-MSG	
IN2	ENT-REEL-TMP-MSG	les deux entrées doivent avoir le même type
Q	BOO	TRUE si IN1 > IN2

Description:

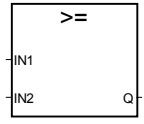
Teste si une valeur est plus grande (Greater Than) qu'une autre (sur les analogiques, les temporisations ou les messages)

(* Programme FBD avec bloc "Plus grand que" *)




```
(* Equivalence ST: *)
aresult:= (10 > 25); (* aresult = FALSE *)
mresult:= ('ab' > 'a'); (* mresult = TRUE *)

(* Equivalence IL: *)
LD      10
GT      25
ST      aresult
LD      'ab'
GT      'a'
ST      mresult
```

>=



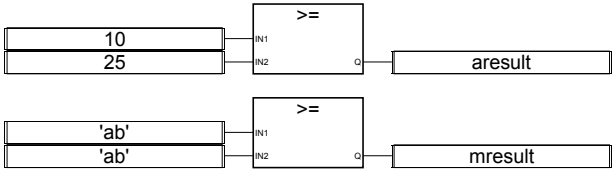
Arguments:

IN1	ENT-REEL-MSG	
IN2	ENT-REEL-MSG	les deux entrées doivent avoir le même type
Q	BOO	TRUE si IN1 >= IN2

Description:

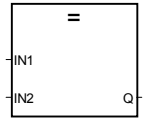
Teste si une valeur est plus grande ou égale (Greater or Equal) à une autre (sur les analogiques, ou les messages)

(* Programme FBD avec bloc "Plus grand ou égal" *)




```
(* Equivalence IL: *)
aresult:= (10 >= 25); (* aresult = FALSE *)
mresult:= ('ab' >= 'ab'); (* mresult = TRUE *)

(* Equivalence IL: *)
LD      10
GE      25
ST      aresult
LD      'ab'
GE      'ab'
ST      mresult
```

=

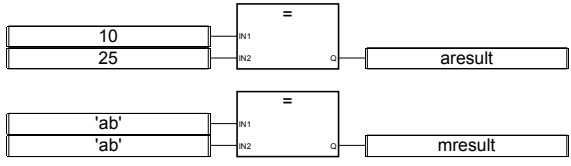


Arguments:

IN1	ENT-REEL-MSG	
IN2	ENT-REEL-MSG	les deux entrées doivent avoir le même type
Q	BOO	TRUE si IN1 = IN2

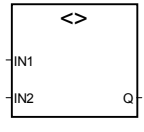
Description:
Teste si une valeur est égale (EQual) à une autre (sur les analogiques, ou les messages)

(* Programme FBD avec bloc "Est égal à" *)




```
(* Equivalence ST: *)
aresult:= (10 = 25); (* aresult = FALSE *)
mresult:= ('ab' = 'ab'); (* mresult = TRUE *)

(* Equivalence IL: *)
LD          10
EQ          25
ST          aresult
LD          'ab'
EQ          'ab'
ST          mresult
```

<>

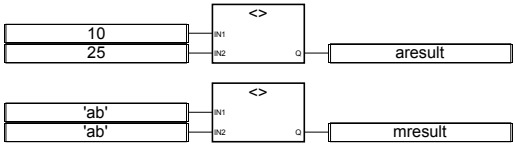


Arguments:

IN1	ENT-REEL-MSG	
IN2	ENT-REEL-MSG	les deux entrées doivent avoir le même type
Q	BOO	TRUE si IN1 <> IN2

Description:
Teste si une valeur est différente (Not Equal) d'une autre (sur les analogiques, ou les messages)

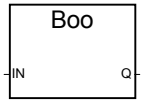
(* Programme FBD avec bloc "Est différent de" *)




```
(* Equivalence ST: *)
aresult:= (10 <> 25); (* aresult = TRUE *)
```

```
mresult:= ('ab' <> 'ab'); (* mresult = FALSE *)

(* Equivalence IL: *)
LD          10
NE          25
ST          aresult
LD          'ab'
NE          'ab'
ST          mresult
```

BOO



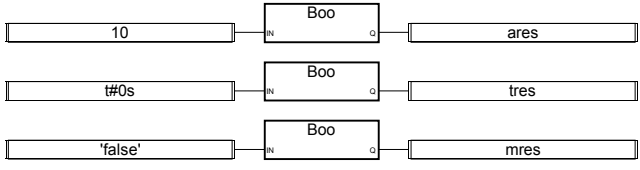
Arguments:

IN	Tout type	toute valeur non booléenne
Q	BOO	TRUE pour une valeur analogique non nulle FALSE pour une valeur analogique nulle TRUE pour un message 'TRUE' FALSE pour un message 'FALSE'

Description:

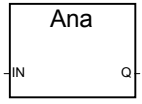
Convertit une variable en une variable booléenne

(* Programme FBD avec conversion en booléen *)




```
(* Equivalence ST: *)
ares:= BOO (10);                (* ares = TRUE *)
tres:= BOO (t#0s);              (* tres = FALSE *)
mres:= BOO ('false');           (* mres = FALSE *)

(* Equivalence IL: *)
LD          10
BOO
ST          ares
LD          t#0s
BOO
ST          tres
LD          'false'
BOO
```

ANA

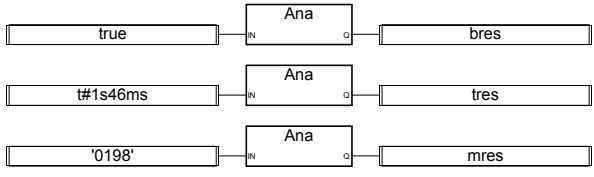


Arguments:

IN	Tout type	toute valeur non analogique entière
Q	ENT	0 pour FALSE / 1 pour TRUE nombre de milli-secondes pour une tempo partie entière d'un analogique réel valeur décimale représentée par un message

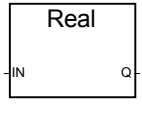
Description:
Convertit une variable en une variable analogique entière

(* Programme FBD avec conversion en entier *)




```
(* Equivalence ST: *)
bres:= ANA (true);                  (* bres = 1 *)
tres:= ANA (t#1s46ms);              (* tres = 1046 *)
mres:= ANA ('0198');                (* mres = 198 *)

(* Equivalence IL: *)
LD          true
ANA
ST          bres
LD          t#1s46ms
ANA
ST          tres
LD          '0198'
ANA
ST          mres
```

REAL



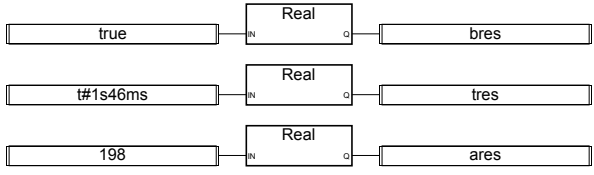
Arguments:

IN	BOO-ENT-

	TMP	toute valeur non analogique réelle (pas de message)
Q	REEL	0.0 pour FALSE / 1.0 pour TRUE nombre de milli-secondes pour une tempo nombre équivalent à un analogique entier

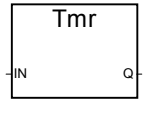
Description:
Convertit une variable en une variable analogique réelle

(* Programme FBD avec bloc conversion en réel *)




```
(* Equivalence ST: *)
bres:= REAL (true);                (* bres = 1.0 *)
tres:= REAL (t#1s46ms);            (* tres = 1046.0 *)
ares:= REAL (198);                 (* ares = 198.0 *)

(* Equivalence IL: *)
LD          true
REAL
ST          bres
LD          t#1s46ms
REAL
ST          tres
LD          198
REAL
ST          ares
```

TMR

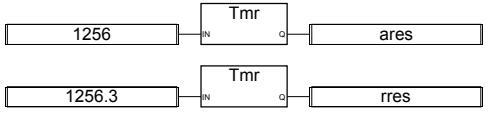


Arguments:

IN	ENT-REEL	toute valeur non temporelle IN (ou la partie entière de IN, si elle est réelle) représente le nombre de milli-secondes
Q	TMP	temps représenté par IN

Description:
Convertit une variable analogique en temporisation

(* Programme FBD avec conversion en temporisation *)




```
(* Equivalence ST: *)
ares:= TMR (1256);                (* ares:= t#1s256ms *)
rres:= TMR (1256.3);              (*rres:= t#1s256ms *)

(* Equivalence IL: *)
LD          1256
TMR
ST          ares
LD          1256.3
TMR
ST          rres
```

MSG

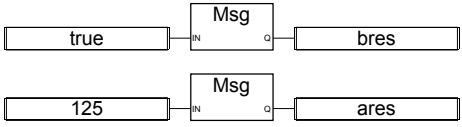
Arguments:

IN	BOO-ENT-REEL	toute valeur non message
Q	MSG	'false' ou 'true' si IN est un booléen représentation décimale pour si IN est un entier

Description:

Convertit une variable en variable message

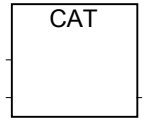
(* Programme FBD avec conversion en message *)




```
(* Equivalence ST: *)
bres:= MSG (true); (* bres = 'TRUE' *)
ares:= MSG (125); (* ares = '125' *)

(* Equivalence IL: *)
LD          true
MSG
ST          bres
LD          125
```

MSG
ST ares

CAT



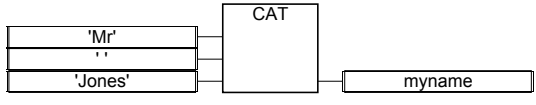
Remarque: Vous pouvez étendre le nombre d'entrées de cet opérateur à plus de deux.

Arguments:

(entrées)	MSG	(la somme des longueurs des messages d'entrée ne doit pas excéder la capacité du message de sortie)
sortie	MSG	concaténation des messages en entrée

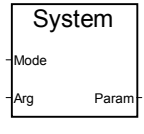
Description:
Concatène plusieurs message en un seul

(* Programme FBD avec bloc "Concaténation" *)




```
(* Equivalence ST: utilisation de l'opérateur + *)
myname:= ('Mr' + ' ') + 'Jones';
(* signifie: myname:= 'Mr Jones' *)

(* Equivalence IL: *)
LD      'Mr'
ADD     ' '
ADD     'Jones'
ST      myname
```

SYSTEM



Arguments:

Mode	ENT	identifie le paramètre système et le mode d'accès réalisé
Arg	ENT-TMP	nouvelle valeur dans le cas d'un accès en écriture
Param	ENT	valeur du paramètre accédé

Description:
Accès aux paramètres système

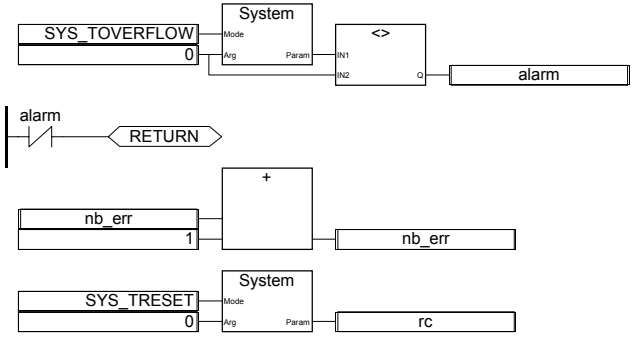
Voici la liste des ordres disponibles (mots clés prédéfinis) pour la fonction SYSTEM:

ordre	signification
SYS_TALLOWED	lecture du temps de cycle programmé
SYS_TCURRENT	lecture du temps de cycle courant
SYS_TMAXIMUM	lecture du temps de cycle maximum
SYS_TOVERFLOW	lecture des dépassements du temps de cycle
SYS_TRESET	remise à zéro des compteurs de temps
SYS_TWRITE	écriture du temps de cycle
SYS_ERR_TEST	test des erreurs d'exécution
SYS_ERR_READ	lecture de la dernière erreur d'exécution

Voici la liste des arguments requis pour les ordres prédéfinis de la fonction SYSTEM:

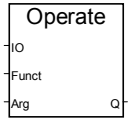
ordre	argument	valeur retournée
SYS_TALLOWED	0	temps de cycle autorisé
SYS_TCURRENT	0	temps de cycle courant
SYS_TMAXIMUM	0	temps de cycle max détecté
SYS_TOVERFLOW	0	nombre de dépassements
SYS_TRESET	0	0
SYS_TWRITE	nouveau temps de cycle autorisé	valeur écrite
SYS_ERR_TEST	0	0 si aucune erreur détectée
SYS_ERR_READ	0	code de la dernière erreur

(* Programme FBD avec bloc "System" *)




```
(* ST Equivalence: *)
alarm:= (SYSTEM (SYS_TOVERFLOW, 0) <> 0);
If (alarm) Then
        nb_err:= nb_err + 1;
        rc:= SYSTEM (SYS_TRESET, 0);
End_If;
```

OPERATE



Arguments:

IO	Tout type	variable d'entrée ou de sortie
Funct	ENT	action à effectuer
Arg	ENT	argument pour l'action d'E/S effectuée
Q	ENT	compte rendu d'exécution

Description:
Accès à une voie d'E/S

La signification des arguments de OPERATE dépend de l'implémentation de l'interface de l'E/S traitée. Référez-vous au manuel de votre automate ou à la fiche technique de la carte d'E/S correspondante pour plus d'informations sur les possibilités offertes par la fonction OPERATE.

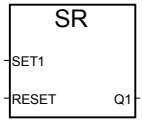
B.9.2 Blocs fonctionnels standards

Voici les blocs fonctionnels standards supportés par le système ISaGRAF. Ces blocs fonctionnels sont prédéfinis et n'ont pas à être déclarés dans la librairie.

- Booléen	SR	Bistable - forçage à TRUE prioritaire
	RS	Bistable - forçage à FALSE prioritaire
	R_Trig	Détection de front montant
	F_Trig	Détection de front descendant
	SEMA	Sémaphore
- Comptage	CTU	Compteur
	CTD	Décompteur
	CTUD	Compteur / décompteur
- Temporisations	TON	Temporisateur à enclenchement
	TOF	Temporisateur à déclenchement
	TP	Temporisateur d'impulsions
- Entiers	CMP	Comparateur
	StackInt	Pile de valeurs analogiques entières
- Réels	AVERAGE	Moyenne courante sur N échantillons
	HYSTER	Hystérésis sur la différence de 2 réels
	LIM_ALRM	Alarme de limite avec d'hystérésis
	INTEGRAL	Intégration en fonction du temps
	DERIVATE	Dérivée par rapport au temps
- Signaux	BLINK	Clignotant
	SIG_GEN	Base de temps

Note: Quand de nouveaux blocs fonctionnels en "C" sont développés, ils deviennent disponibles pour le langage FBD.

SR



Arguments:

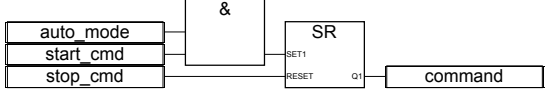
SET1	BOO	si TRUE, force Q1 à TRUE (prioritaire)
RESET	BOO	si TRUE, force Q1 à FALSE
Q1	BOO	état de la bascule

Description:

bistable à mise à 1 prioritaire: Voici sa table de vérité::

Set1	Reset	Q1	Q1 résultant
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

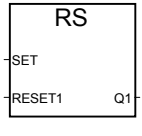
(* programme FBD avec bloc "SR" *)




```
(* Equivalence ST: SR1 est une instance du bloc SR *)
SR1((auto_mode & start_cmd), stop_cmd);
command:= SR1.Q1;
```

```
(* Equivalence IL: *)
LD          auto_mode
AND         start_cmd
ST          SR1.set1
LD          stop_cmd
ST          SR1.reset
CAL         SR1
LD          SR1.Q1
ST          command
```

RS



Arguments:

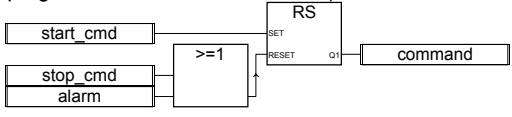
SET	BOO	si TRUE, force Q1 à TRUE
RESET1	BOO	si TRUE, force Q1 à FALSE (prioritaire)
Q1	BOO	état de la bascule

Description:

Bistable à remise à 0 prioritaire: Voici sa table de vérité:

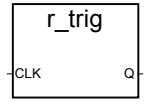
Set	Reset1	Q1	Q1 résultant
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

(* programme FBD avec bloc "RS" *)




```
(* Equivalence: RS1 est une instance du bloc RS *)
RS1(start_cmd, (stop_cmd OR alarm));
command:= RS1.Q1;

(* Equivalence IL: *)
LD      start_cmd
ST      RS1.set
LD      stop_cmd
OR      alarm
ST      RS1.reset1
CAL     RS1
LD      RS1.Q1
ST      command
```

R_TRIG



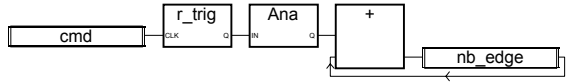
Arguments:

CLK	BOO	toute variable booléenne
Q	BOO	TRUE quand CLK passe de FALSE à TRUE FALSE dans tous les autres cas

Description:

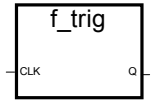
Détection du front montant d'une variable booléenne

(* Programme FBD avec bloc fonctionnel "R_TRIG" *)




```
(* Equivalence ST: R_TRIG1 est une instance du bloc R_TRIG *)
R_TRIG1(cmd);
nb_edge:= ANA(R_TRIG1.Q) + nb_edge;

(* Equivalence IL: *)
LD          cmd
ST          R_TRIG1.clk
CAL         R_TRIG1
LD          R_TRIG1.Q
ANA
ADD         nb_edge
ST          nb_edge
```

F_TRIG



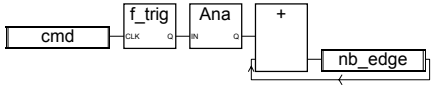
Arguments:

CLK	BOO	toute variable booléenne
Q	BOO	TRUE quand CLK passe de TRUE à FALSE FALSE dans tous les autres cas

Description:

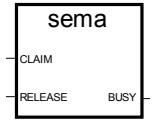
Détection le front descendant d'une variable booléenne

(* Programme FBD avec bloc fonctionnel "F_TRIG" *)




```
(* Equivalence ST: F_TRIG1 est une instance du bloc F_TRIG *)
F_TRIG1(cmd);
nb_edge:= ANA(F_TRIG1.Q) + nb_edge;

(* Equivalence IL: *)
LD          cmd
ST          F_TRIG1.clk
CAL         F_TRIG1
LD          F_TRIG1.Q
ANA
ADD         nb_edge
ST          nb_edge
```

SEMA



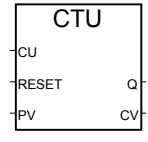
Arguments:

CLAIM	BOO	commande "test and set"
RELEASE	BOO	commande de libération de la ressource (libère le sémaphore)
BUSY	BOO	état du sémaphore (TRUE = occupé)

Description:

```
(* "x" est une variable booléenne initialisée à FALSE *)
busy:= x;
If claim Then
        x:= True;
Else
        If release Then
                busy:= False;
                x:= False;
        End_if;
End_if;
```

CTU



Arguments:

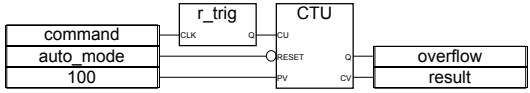
CU	BOO	entrée de comptage (compte si CU = TRUE)
RESET	BOO	remise à zéro (prioritaire)
PV	ENT	valeur maximum programmée
Q	BOO	signal "plein": TRUE quand CV = PV (overflow)
CV	ENT	résultat courant du comptage

Attention: Le bloc CTU ne détecte pas les fronts montants ou descendants de l'entrée de comptage (CU). Il doit être associé à un bloc fonctionnel "R_TRIG" ou "F_TRIG" pour réaliser un compteur d'impulsions.

Description:

Compte (en entier) de 0 à une valeur donnée par pas de 1

(* Programme FBD avec bloc fonctionnel "CTU" *)

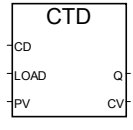


(* Equivalence ST: R_TRIG1 est une instance du bloc R_TRIG et CTU1 est une instance du bloc CTU *)
CTU1(R_TRIG1(command),NOT(auto_mode),100);
overflow:= CTU1.Q;
result:= CTU1.CV;

(* Equivalence IL: *)

```
LD      command
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ST      CTU1.cu
LDN     auto_mode
ST      CTU1.reset
LD      100
ST      CTU1.pv
CAL     CTU1
LD      CTU1.Q
ST      overflow
LD      CTU1.cv
ST      result
```

CTD



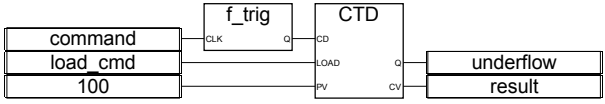
Arguments:

CD	BOO	entrée de comptage (décompte si CD = TRUE)
LOAD	BOO	commande de chargement (prioritaire) (CV = PV si LOAD = TRUE)
PV	ENT	valeur initiale programmée
Q	BOO	signal "vide": TRUE quand CV = 0 (underflow)
CV	ENT	résultat courant du comptage

Attention: Le bloc CTD ne détecte pas les fronts montants ou descendants de l'entrée de comptage (CD). Il doit être associé à un bloc fonctionnel "R_TRIG" ou "F_TRIG" pour réaliser un compteur d'impulsions.

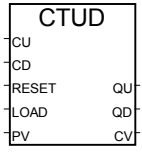
Description:
Compte (décompte) (en entier) d'une valeur donnée jusqu'à 0 par pas de 1

(* Programme FBD avec bloc fonctionnel "CTD" *)




```
(* Equivalence ST: F_TRIG1 est une instance du bloc F_TRIG bloc et CTD1 est une instance
du bloc CTD *)
CTD1(F_TRIG1(command),load_cmd,100);
underflow:= CTD1.Q;
result:= CTD1.CV;

(* IL Equivalence: *)
LD      command
ST      F_TRIG1.clk
CAL     F_TRIG1
LD      F_TRIG1.Q
ST      CTD1.cd
LD      load_cmd
ST      CTD1.load
LD      100
ST      CTD1.pv
CAL     CTD1
LD      CTD1.Q
ST      underflow
LD      CTD1.cv
ST      result
```

CTUD



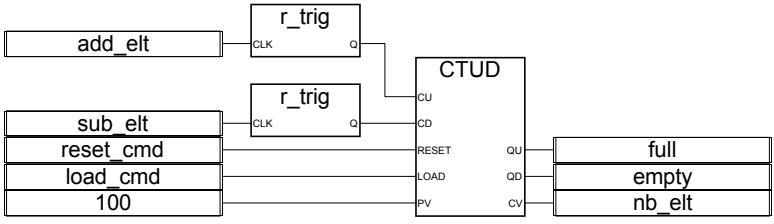
Arguments:

CU	BOO	comptage (incrémentation quand CU = TRUE)
CD	BOO	décomptage (décrémentation quand CD = TRUE)
RESET	BOO	commande de remise à zéro (prioritaire) (CV = 0 si RESET = TRUE)
LOAD	BOO	commande de chargement (CV = PV si LOAD = TRUE)
PV	ENT	valeur maximum programmée
QU	BOO	signal "plein": TRUE quand CV = PV (overflow
QD	BOO	signal "vide": TRUE quand CV = 0 (underflow)
CV	ENT	résultat courant du comptage

Attention: Le bloc CTUD ne détecte pas les fronts montants ou descendants des entrées de comptage (CU et CD). Il doit être associé à un bloc fonctionnel "R_TRIG" ou "F_TRIG" pour réaliser un compteur d'impulsions.

Description:
Compte (en entier) de 0 à une valeur donnée par pas de 1
ou d'une valeur donnée jusqu'à 0 par pas de 1

(* Programme FBD avec bloc fonctionnel "CTUD" *)

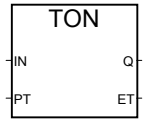


(* Equivalence ST: R_TRIG1 et R_TRIG2 sont deux instances du bloc R_TRIG et CTUD1 est une instance du bloc CTUD *)
CTUD1(R_TRIG1(add_elt), R_TRIG2(sub_elt), reset_cmd, load_cmd,100);
full:= CTUD1.QU;
empty:= CTUD1.QD;
nb_elt:= CTUD1.CV;

(* Equivalence IL: *)

```
LD      add_elt
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ST      CTUD1.cu
LD      sub_elt
ST      R_TRIG2.clk
CAL     R_TRIG2
LD      R_TRIG2.Q
ST      CTUD1.cd
LD      reset_cmd
ST      CTUD1.reset
LD      load_cmd
ST      CTUD1.load
LD      100
ST      CTUD1.pv
CAL     CTUD1
LD      CTUD1.QU
ST      full
LD      CTUD1.QD
ST      empty
LD      CTUD1.CV
ST      nb_elt
```

TON



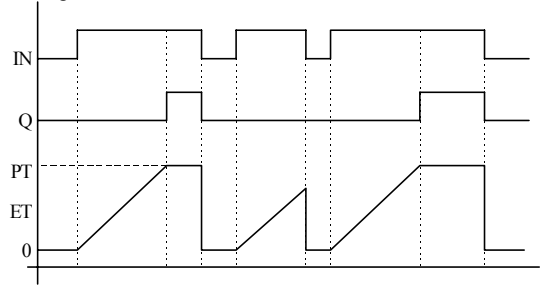
Arguments: {XE "TON"} {XE "Temporisateur à enclenchement"}

IN	BOO	si front montant, démarre l'incrément d'une temporisation interne si front descendant, arrête et remet à 0 la temporisation interne
PT	TMP	temps maximum programmé
Q	BOO	si TRUE, le temps programmé est écoulé
ET	TMP	temps écoulé depuis le démarrage

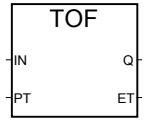
Description:

Incrémente une temporisation interne jusqu'à une valeur donnée (temporisateur à enclenchement)

Chronogramme:



TOF



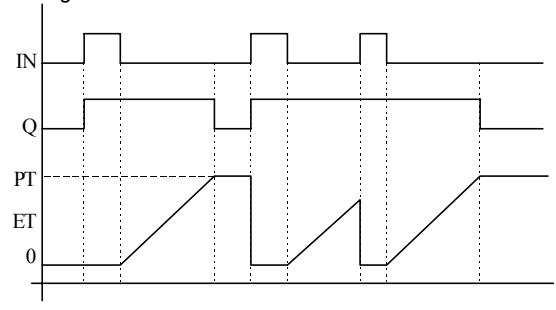
Arguments:

IN	BOO	si front descendant, démarre l'incrément d'une temporisation si front descendant, arrête et remet à 0 la temporisation interne
PT	TMP	temps maximum programmé
Q	BOO	si TRUE: le temps programmé n'est pas encore écoulé
ET	TMP	temps écoulé depuis le démarrage

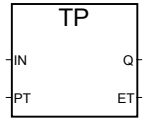
Description:

Incrémente une temporisation interne jusqu'à une valeur donnée (temporisateur à déclenchement)

Chronogramme:



TP



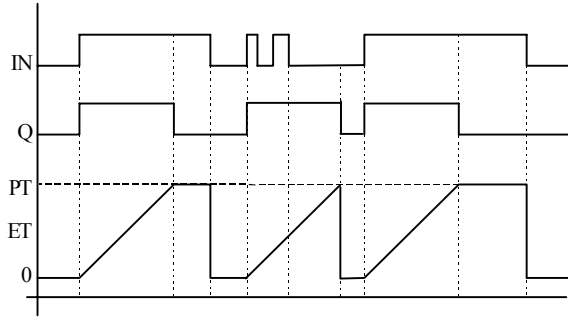
Arguments:

IN	BOO	si front montant, démarre l'incrément d'une temporisation interne (si elle n'est pas déjà active) si FALSE et seulement si le temps est écoulé, remet à 0 la temporisation interne Tout changement sur IN pendant le comptage, est sans effet.
PT	TMP	temps maximum programmé
Q	BOO	si TRUE: la temporisation est en cours (comptage)
ET	TMP	temps écoulé depuis le démarrage

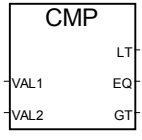
Description:

Incrémente une temporisation interne jusqu'à une valeur donnée (temporisateur à impulsion)

Chronogramme:



CMP



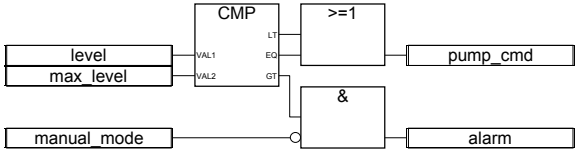
Arguments:

VAL1	ENT	toute valeur analogique entière signée
VAL2	ENT	toute valeur analogique entière signée
LT	BOO	TRUE si val1 est plus petit val2
EQ	BOO	TRUE si val1 est égal à val2
GT	BOO	TRUE si val1 si val1 est plus grand que val2

Description:

Compare deux valeurs: signale si elles sont égales, ou si la première est plus petite ou plus grande que la seconde.

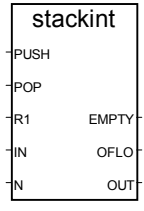
(* Programme FBD avec bloc fonctionnel "CMP" *)




```
(* Equivalence ST: CMP1 est une instance du bloc CMP *)
CMP1(level, max_level);
pump_cmd:= CMP1.LT OR CMP1.EQ;
alarm:= CMP1.GT AND NOT(manual_mode);

(* Equivalence IL: *)
LD          level
ST          CMP1.val1
LD          max_level
```

```
ST      CMP1.val2
CAL     CMP1
LD      CMP1.LT
OR      CMP1.EQ
ST      pump_cmd
LD      CMP1.GT
ANDN    manual_mode
ST      alarm
```

STACKINT



Arguments:

PUSH	BOO	commande "empiler" (sur front montant) ajoute la valeur IN au somment de la pile
POP	BOO	commande "dépiler" (sur front montant) détruit dans la pile la dernière valeur empilée (sommet de la pile)
R1	BOO	vide la pile
IN	INT	valeur à empiler
N	INT	taille de la pile définie par l'application
EMPTY	BOO	TRUE si la pile est vide
OFLO	BOO	TRUE si la pile est pleine (overflow)
OUT	INT	valeur au sommet de la pile

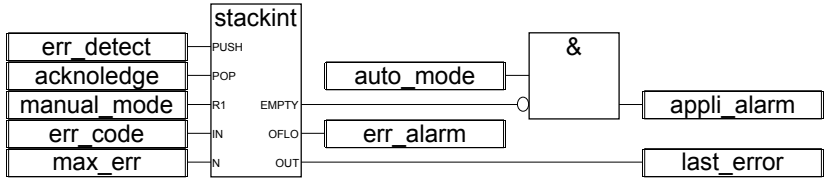
Description:

Gère une pile de valeurs entières.

Le bloc fonctionnel "STACKINT" inclut une détection de front montant pour les entrées PUSH et POP (commandes "empiler" et "dépiler"). La taille maximum absolue de la pile est de **128** éléments. La taille définie par l'application ne peut pas être inférieure à 1 ou supérieure à 128.

Remarque: la valeur de OFLO est géré seulement après au moins une remise à 0 (R1a été mis à TRUE au moins une fois et remis à FALSE).

(* Programme FBD avec bloc fonctionnel "STACKINT": gestion d'erreurs *)

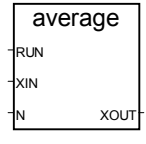


(* Equivalence ST: STACKINT1 est une instance du bloc STACKINT *)

```
STACKINT1(err_detect, acknoledge, manual_mode, err_code, max_err);
appli_alarm:= auto_mode AND NOT(STACKINT1.EMPTY);
err_alarm:= STACKINT1.OFLO;
last_error:= STACKINT1.OUT;
```

```
(* Equivalence IL: *)
LD      err_detect
ST      STACKINT1.push
LD      acknoledge
ST      STACKINT1.pop
LD      manual_mode
ST      STACKINT1.r1
LD      err_code
ST      STACKINT1.IN
LD      max_err
ST      STACKINT1.N
CAL     STACKINT1
LD      auto_mode
ANDN    STACKINT1.empty
ST      appli_alarm
LD      STACKINT1.OFLO
ST      err_alarm
LD      STACKINT1.OUT
ST      last_error
```

AVERAGE



Arguments:

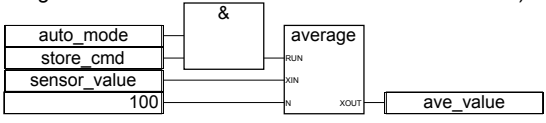
RUN	BOO	TRUE = marche / FALSE = ré-initialisation
XIN	REAL	toute valeur analogique réelle
N	INT	nombre d'échantillons (défini par l'application)
XOUT	REAL	moyenne courante de l'entrée XIN

Description:

Stocke une valeur à chaque cycle et calcule la valeur **moyenne** des valeurs déjà stockées. Seules les N dernières valeurs sont stockées.

Le nombre d'échantillons ne peut pas excéder **128**.
Si l'entrée "**RUN**" vaut **FALSE** (ré-initialisation), la valeur de sortie est égale à la valeur d'entrée.
Quand N valeurs ont été stockées, la première valeur qui a été stockée est écrasée par la dernière.

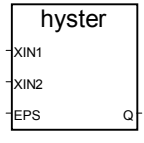
(* Programme FBD avec bloc fonctionnel "AVERAGE": *)




```
(* Equivalence ST: AVERAGE1 est une instance du bloc AVERAGE *)
AVERAGE1((auto_mode & store_cmd), sensor_value, 100);
ave_value:= AVERAGE1.XOUT;
```

```
(* Equivalence IL: *)
LD      auto_mode
AND     store_cmd
ST      AVERAGE1.run
LD      sensor_value
ST      AVERAGE1.xin
LD      100
ST      AVERAGE1.N
CAL     AVERAGE1
LD      AVERAGE1.XOUT
ST      ave_value
```

HYSTER

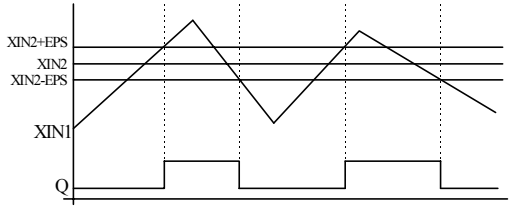


Arguments:

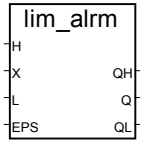
XIN1	REEL	toute valeur analogique réelle
XIN2	REEL	pour tester si XIN1 est au-delà de XIN2+EPS
EPS	REEL	REEL valeur de l'hystérésis (doit être positif)
Q	BOO	TRUE si XIN1 est au-delà de XIN2+EPS et n'est pas encore en dessous de XIN2-EPS

Description:
Hysteresis sur limite haute sur une valeur analogique réelle.

Exemple de chronogramme:



LIM_ALRM



Arguments:

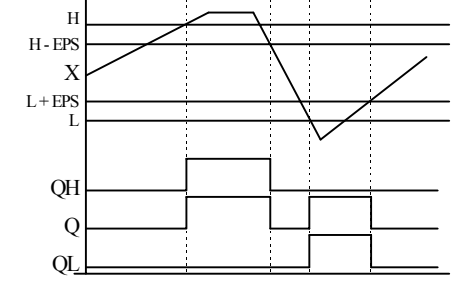
H	REEL	valeur de la limite haute
X	REEL	entrée: toute valeur analogique réelle
L	REEL	valeur de la limite basse
EPS	REEL	valeur de l'**hysteresis** (doit être positif)
QH	BOO	alarme haute: TRUE si X au dessus de la limite haute H
Q	BOO	alarme: TRUE si X hors limites
QL	BOO	alarme basse: TRUE si X en dessous de la limite basse L

Description:

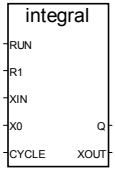
Hysteresis sur limites haute et basse sur une valeur analogique réelle.

Un hystérésis est appliqué aux limites haute et basse. Le delta d'hystérésis utilisé pour la limite haute ou la limite basse est la moitié de la valeur du paramètre EPS.

Exemple de chronogramme:



INTEGRAL



Arguments:

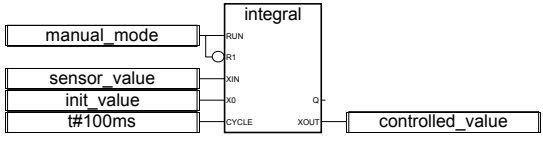
RUN	BOO	mode: TRUE = intégration / FALSE = attente
R1	BOO	commande de remise à zéro
XIN	REEL	entrée: toute valeur analogique réelle
X0	REAL	valeur initiale
CYCLE	TMP	période d'échantillonnage
Q	BOO	Not R1
XOUT	REEL	valeur intégrée

Description:

Intégration d'une valeur analogique réelle.

Si la valeur du paramètre "**CYCLE**" est inférieure au temps de cycle de l'application ISaGRAF, la période d'échantillonnage est égale au temps de cycle de l'application.

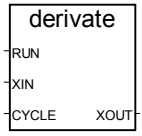
(* Programme FBD avec bloc "INTEGRAL": *)




```
(* Equivalence ST: INTEGRAL1 est une instance du bloc INTEGRAL *)
INTEGRAL1(manual_mode, NOT(manual_mode), sensor_value, init_value, t#100ms);
controlled_value:= INTEGRAL1.XOUT;
```

```
(* Equivalence IL: *)
LD      manual_mode
ST      INTEGRAL1.run
STN     INTEGRAL1.R1
LD      sensor_value
ST      INTEGRAL1.XIN
LD      init_value
ST      INTEGRAL1.X0
LD      t#100ms
ST      INTEGRAL1.CYCLE
CAL     INTEGRAL1
LD      INTEGRAL1.XOUT
ST      controlled_value
```

DERIVATE



Arguments:

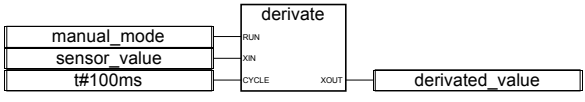
RUN	BOO	mode: TRUE = normal / FALSE = remise à zéro
XIN	REEL	entrée: toute valeur analogique réelle
CYCLE	TMP	période d'échantillonnage
XOUT	REEL	sortie: dérivée du signal

Description:

Différentiation d'une valeur analogique réelle.

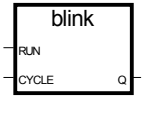
Si la valeur du paramètre "**CYCLE**" est inférieure au temps de cycle de l'application ISaGRAF, la période d'échantillonnage est égale au temps de cycle de l'application.

(* Programme FBD avec bloc "DERIVATE": *)




```
(* Equivalence ST: DERIVATE1 est une instance du bloc DERIVATE *)
DERIVATE1(manual_mode, sensor_value, t#100ms);
derivated_value:= DERIVATE1.XOUT;

(* Equivalence IL: *)
LD      manual_mode
ST      DERIVATE1.run
LD      sensor_value
ST      DERIVATE1.XIN
LD      t#100ms
ST      DERIVATE1.CYCLE
CAL     DERIVATE1
LD      DERIVATE1.XOUT
ST      derivated_value
```

BLINK



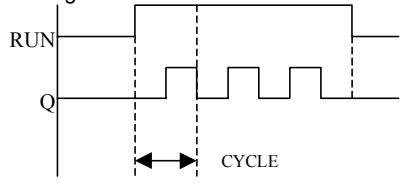
Arguments:

RUN	BOO	mode: TRUE = clignotant / FALSE = remise à false de la sortie
CYCLE	TMR	période du clignotant

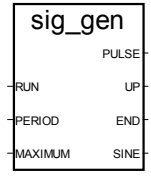
Q	BOO	sortie: signal clignotant

Description:
Génère un signal clignotant.

Chronogramme:



SIG_GEN



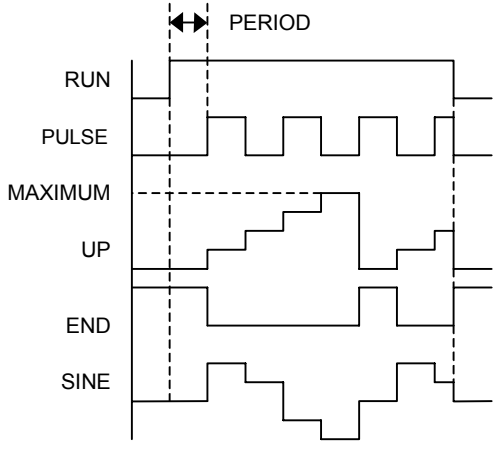
Arguments:

RUN	BOO	mode: TRUE = marche / FALSE = remise à 0
PERIOD	TMP	période d'un échantillon
MAXIMUM	ENT	valeur maximum de comptage
PULSE	BOO	inversé à chaque échantillon
UP	ENT	compteur incrémenté à chaque échantillon
END	BOO	TRUE en fin de comptage
SINE	REEL	sinus (période = temps de comptage)

Description:
Génère plusieurs signaux: clignotant sur un booléen, comptage d'un entier, et signal sinusoïdal réel.

Quand le comptage atteint sa valeur maximum, il redémarre à 0. Donc END reste à TRUE seulement pendant un échantillon (PERIOD).

Chronogramme:



B.9.3 Fonctions standards

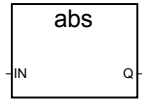
Voici les fonctions standards supportées par le système ISaGRAF. Ces fonctions sont prédéfinies et n'ont pas à être déclarées dans la librairie.

- Fonctions mathématiques.....	ABS	Valeur absolue
	EXPT, POW	Exponentiation, élévation à une puissance
	LOG	Logarithme
	SQRT	Racine carrée
	TRUNC	Partie entière
- Fonctions trigonométriques.....	ACOS,ASIN	Arc cosinus, Arc sinus
	ATAN	Arc tangente
	COS, SIN	Cosinus, Sinus
	TAN	Tangente
- Manipulation de registres.....	ROL, ROR	Rotation à gauche, Rotation à droite
	SHL, SHR	Décalage à gauche, Décalage à droite
- Manipulation de données.....	MIN, MAX	Minimum, Maximum
	LIMIT	Limite
	MOD	Modulo
	MUX4, MUX8	Multiplexeur (4 ou 8 entrées)
	SEL	Sélecteur binaire
	ODD	Imparité
	RAND	Nombre aléatoire
- Conversion de données	ASCII	Caractère → Code Ascii
	CHAR	Code Ascii → Caractère
- Chaînes de caractères.....	MLEN	Longueur courante
	DELETE	Suppression
	INSERT	Insertion de caractères
	FIND	Recherche
	REPLACE	Remplacement de caractères
	LEFT, MID	Extraction à gauche, au milieu,
	RIGHT	à droite
	DAY_TIME	Date et heure courantes
- Gestion de tableaux.....	ARCREATE	Création de tableau

	ARREAD	Lecture d'éléments
	ARWRITE	Ecriture d'éléments
- Gestion de fichiers binaires.....	F_ROPEN	Ouvre un fichier binaire en mode lecture
	F_WOPEN	Ouvre un fichier binaire en mode écriture
	F_CLOSE	Ferme un fichier binaire
	F_EOF	Teste la fin d'un fichier binaire
	FA_READ	Lit une valeur analogique dans un fichier binaire
	FA_WRITE	Ecrit une valeur analogique dans un fichier binaire
	FM_READ	Lit un message (chaîne de caractères) dans un fichier binaire
	FM_WRITE	Ecrit un message (chaîne de caractères) dans un fichier binaire

Note: quand une nouvelle procédure "C" est créée, elle devient disponible sous forme d'une boîte fonction pour le langage FBD.

ABS



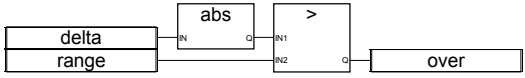
Arguments:

IN	REEL	toute valeur réelle signée
Q	REEL	**valeur absolue** (toujours positive)

Description:

Donne la valeur absolue (positive) d'une valeur analogique réelle.

(* Programme FBD avec bloc "ABS" *)

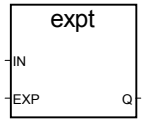


(* Equivalence ST: *)
over:= (ABS (delta) > range);

(* Equivalence IL: *)

```
LD      delta
ABS
GT      range
ST      over
```

EXPT



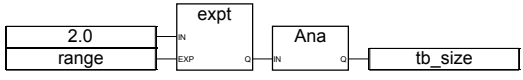
Arguments:

IN	REEL	toute valeur réelle signée
EXP	ENT	exposant entier
Q	REEL	(IN ^{EXP}) (**Exponentiation**)

Description:

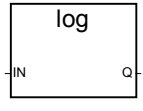
Donne le résultat analogique réel de l'opération: (base ^{exponent}) 'base' étant le premier argument et 'exponent' le deuxième.

(* Programme FBD avec bloc "EXT" *)




```
(* Equivalence ST: *)
tb_size:= ANA (EXPT (2.0, range) );

(* Equivalence IL: *)
LD          2.0
EXPT        range
ANA
ST          tb_size
```

LOG



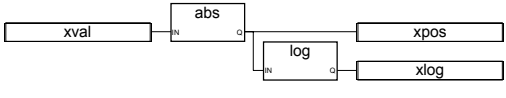
Arguments:

IN	REEL	doit être plus grand que zéro
Q	REEL	**logarithme** (base 10) de la valeur d'entrée

Description:

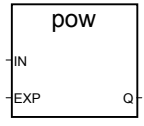
Calcule le logarithme (base 10) d'une valeur réelle.

(* Programme FBD avec bloc "LOG" *)



```
(* Equivalence ST: *)
xpos:= ABS (xval);
xlog:= LOG (xpos);

(* Equivalence IL: *)
LD          xval
ABS
ST          xpos
LOG
ST          xlog
```

POW



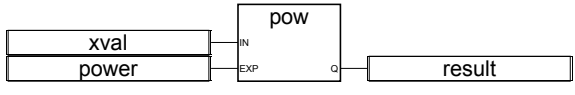
Arguments:

IN	REEL	nombre analogique réel à élever
EXP	REEL	puissance (exposant)
Q	REEL	(IN ^{EXP}) (élévation à une **puissance**) 1.0 si IN est différent de 0.0 et EXP est égal à 0.0 0.0 si IN est égal à 0.0 et EXP est négatif 0.0 si IN et EXP sont égaux à 0.0 0.0 si IN est négatif et Y ne correspond pas à un entier

Description:

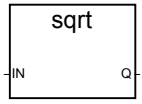
Donne le résultat analogique réel de: (base ^{exponent}) 'base' étant le premier argument et 'exponent' le deuxième. 'exponent' est une valeur réelle.

(* Programme FBD avec bloc "POW" *)




```
(* Equivalence ST: *)
result:= POW (xval, power);

(* Equivalence IL: *)
LD          xval
POW         power
ST          result
```

SQRT

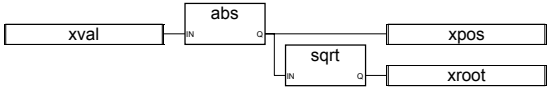


Arguments:

IN	REEL	doit être plus grand ou égal à zéro
Q	REEL	**racine carrée** de la valeur d'entrée

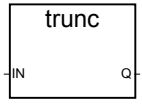
Description:
Calcule la racine carrée d'une valeur réelle.

(* Programme FBD avec bloc "SQRT" *)




```
(* Equivalence ST: *)
xpos:= ABS (xval);
xroot:= SQRT (xpos);

(* Equivalence IL: *)
LD          xval
ABS
ST          xpos
SQRT
ST          xrout
```

TRUNC

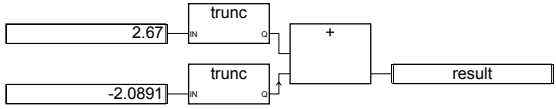


Arguments:

IN	REEL	toute valeur réelle signée
Q	REEL	si IN>0, plus grand entier inférieur ou égal à IN si IN<0, plus petit entier supérieur ou égal à IN

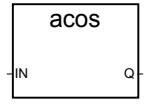
Description:
Tronque une valeur réelle de façon à obtenir sa **partie entière**.

(* Programme FBD avec bloc "TRUNC" *)




```
(* Equivalence ST: *)
result:= TRUNC (+2.67) + TRUNC (-2.0891);
(* signifie: result:= 2.0 + (-2.0):= 0.0; *)

(* Equivalence IL: *)
LD          2.67
TRUNC
ST          temporary   (* Résultat temporaire du premier TRUNC *)
LD          -2.0891
TRUNC
ADD         temporary
ST          result
```

ACOS



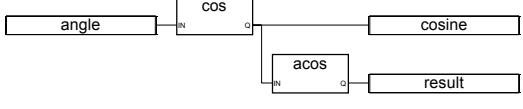
Arguments:

IN	REEL	doit être dans l'intervalle [-1.0 .. +1.0]
Q	REEL	**arc-cosinus** de la valeur d'entrée dans l'intervalle [0.0 .. PI] = 0.0 pour une entrée invalide

Description:

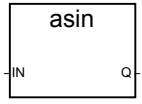
Calcule l'Arc cosinus d'une valeur réelle.

(* Programme FBD avec blocs "COS" et "ACOS" *)




```
(* Equivalence ST: *)
cosine:= COS (angle);
result:= ACOS (cosine); (* result est égal à angle *)

(* Equivalence IL: *)
LD          angle
COS
ST          cosine
ACOS
ST          result
```

ASIN

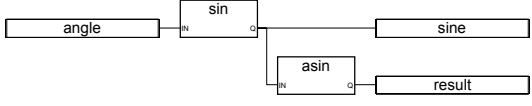


Arguments: **{**XE "ASIN"**}**

IN	REEL	doit être dans l'intervalle [-1.0 .. +1.0]
Q	REEL	**arc-sinus{**XE "Arc sinus"**}** de la valeur d'entrée dans l'intervalle [-PI/2 .. +PI/2] = 0.0 pour une entrée invalide

Description:
Calcule l'Arc sinus d'une valeur réelle.

(* Programme FBD avec blocs "SIN" et "ASIN" *)



(* Equivalence ST: *)
sine:= SIN (angle);
result:= ASIN (sine); (* result est égal à angle *)

(* Equivalence IL: *)
LD angle
SIN
ST sine
ASIN
ST result

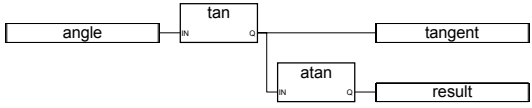
ATAN

Arguments:

IN	REEL	toute valeur analogique réelle
Q	REEL	**arc-tangente** de la valeur d'entrée dans l'intervalle [-PI/2 .. +PI/2] = 0.0 pour une entrée invalide

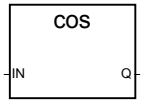
Description:
Calcule l'Arc tangente d'une valeur réelle.

(* Programme FBD avec blocs "TAN" et "ATAN" *)




```
(* Equivalence ST: *)
tangent:= TAN (angle);
result:= ATAN (tangent); (* result est égal à angle*)

(* Equivalence IL: *)
LD          angle
TAN
ST          tangent
ATAN
ST          result
```

COS



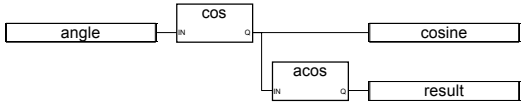
Arguments:

IN	REEL	toute valeur analogique réelle
Q	REEL	**cosinus** de la valeur d'entrée dans l'intervalle [-1.0 .. +1.0]

Description:

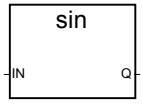
Calcule le Cosinus d'une valeur réelle.

(* Programme FBD avec blocs "COS" et "ACOS" *)




```
(* Equivalence ST: *)
cosine:= COS (angle);
result:= ACOS (cosine); (* result est égal à angle *)

(* Equivalence IL: *)
LD          angle
COS
ST          cosine
ACOS
ST          result
```

SIN

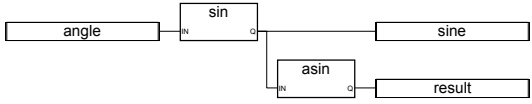


Arguments:

IN	REEL	toute valeur analogique réelle
Q	REEL	**sinus** de la valeur d'entrée dans l'intervalle [-1.0 .. +1.0]

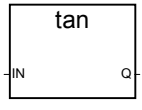
Description:
Calcule le Sinus d'une valeur réelle.

(* Programme FBD avec blocs "SIN" et "ASIN" *)




```
(* Equivalence ST: *)
sine:= SIN (angle);
result:= ASIN (sine); (* result est égal à angle *)

(* Equivalence IL: *)
LD      angle
SIN
ST      sine
ASIN
ST      result
```

TAN

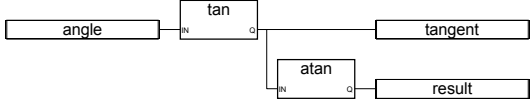


Arguments:

IN	REEL	ne peut pas être égal à PI/2 modulo PI
Q	REEL	**tangente** de la valeur d'entrée = 1E+38 pour une entrée invalide

Description:
Calcule la Tangente d'une valeur réelle.

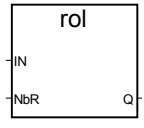
(* Programme FBD avec blocs "TAN" et "ATAN" *)



(* Equivalence ST: *)

```
tangent:= TAN (angle);
result:= ATAN (tangent); (* result est égal à angle*)

(* Equivalence IL: *)
LD              angle
TAN
ST              tangent
ATAN
ST              result
```

ROL

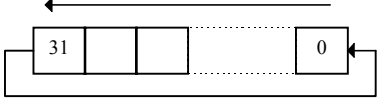


Arguments:

IN	ENT	toute valeur analogique ENTIERE
NbR	ENT	nombre de rotations de 1 bit (dans l'intervalle [1..31])
Q	ENT	valeur décalée à gauche sans effet si NbR <= 0

Description:

Rotation à gauche des bits d'un entier. La rotation est effectuée sur 32 bits:



(* Programme FBD avec bloc "ROL" *)


```
(* Equivalence ST: *)
result:= ROL (register, 1);
(* register = 2#0100_1101_0011_0101*)
(* result   = 2#1001_1010_0110_1010*)

(* Equivalence IL: *)
LD              register
ROL             1
ST              result
```

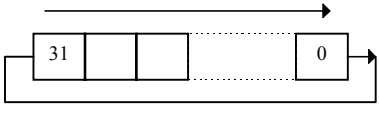
ROR

Arguments:

IN	ENT	toute valeur analogique ENTIERE
NbR	ENT	nombre de rotations de 1 bit (dans l'intervalle [1..31])
Q	ENT	valeur décalée à droite sans effet si NbR <= 0

Description:

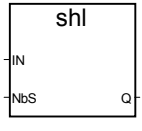
Rotation à droite des bits d'un entier. La rotation est effectuée sur 32 bits:



(* Programme FBD avec bloc "ROR" *)


```
(* Equivalence ST: *)
result:= ROR (register, 1);
(* register = 2#0100_1101_0011_0101 *)
(* result   = 2#1010_0110_1001_1010 *)

(* Equivalence IL: *)
LD          register
ROR         1
ST          result
```

SHL

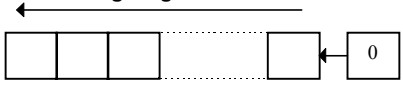


Arguments:

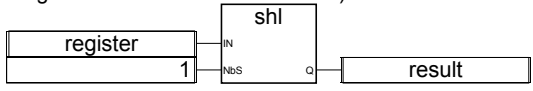
IN	ENT	toute valeur analogique ENTIERE
NbS	ENT	nombre de décalages de 1 bit (dans l'intervalle [1..31])
Q	ENT	valeur décalée à gauche sans effet si NbS <= 0 la valeur 0 remplace le bit de poids faible

Description:

Décalage à gauche des bits d'un entier. Le décalage est effectué sur 32 bits:

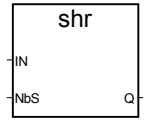


(* Programme FBD avec bloc "SHL" *)




```
(* Equivalence ST: *)
result:= SHL (register,1);
(* register = 2#0100_1101_0011_0101 *)
(* result   = 2#1001_1010_0110_1010 *)

(* Equivalence IL: *)
LD          register
SHL         1
ST          result
```

SHR

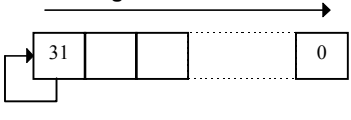


Arguments:

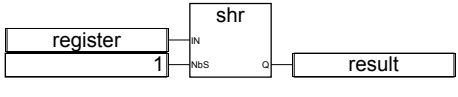
IN	ENT	toute valeur analogique ENTIERE
NbS	ENT	nombre de décalages de 1 bit (dans l'intervalle [1..31])
Q	ENT	valeur décalée à droite sans effet si NbS <= 0 le bit de poids fort est recopié à chaque décalage

Description:

Décalage à droite des bits d'un entier. Le décalage est effectué sur 32 bits:



(* Programme FBD avec bloc "SHR" *)




```
(* Equivalence ST: *)
result:= SHR (register,1);
```

```
(* register = 2#1100_1101_0011_0101 *)
(* result   = 2#1110_0110_1001_1010 *)

(* Equivalence IL: *)
LD          register
SHR         1
ST          result
```

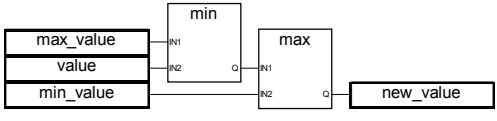
MIN

Arguments:

IN1	INT	toute valeur entière signée
IN2	INT	(ne peut pas être REEL)
Q	INT	**minimum** des deux entrées

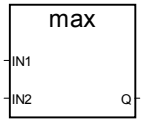
Description:
Donne la valeur minimum entre deux entiers.

(* Programme FBD avec blocs "MIN" et "MAX" *)




```
(* Equivalence ST: *)
new_value:= MAX (MIN (max_value, value), min_value);
(* Limite la valeur à l'intervalle [min_value..max_value] *)

(* Equivalence IL: *)
LD          max_value
MIN         value
MAX         min_value
ST          new_value
```

MAX

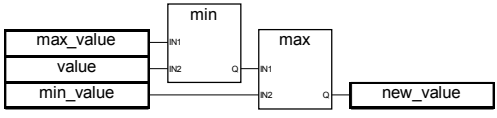


Arguments:

IN1	ENT	toute valeur entière signée
IN2	ENT	(ne peut pas être REEL)
Q	ENT	**maximum** des deux entrées

Description:
Donne la valeur maximum entre deux entiers.

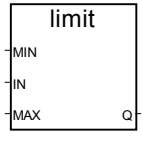
(* Programme FBD avec blocs "MIN" et "MAX" *)



(* Equivalence ST: *)
new_value:= MAX (MIN (max_value, value), min_value);
(* Limite la valeur à l'intervalle [min_value..max_value] *)

(* Equivalence IL: *)
LD max_value
MIN value
MAX min_value
ST new_value

LIMIT

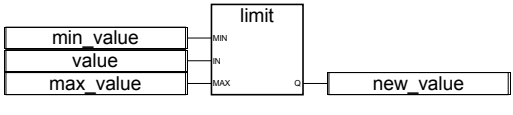


Arguments:

MIN	ENT	valeur minimum autorisée
IN	ENT	toute valeur analogique entière signée
MAX	ENT	valeur maximum autorisée
Q	ENT	valeur limitée à la plage autorisée

Description:
Limite une valeur entière à un intervalle donné. Soit elle conserve sa valeur si elle est entre le minimum et le maximum, soit elle est forcée à la valeur maximum si elle est au dessus, soit elle est forcée au minimum si elle est en dessous.

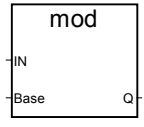
(* Programme FBD avec bloc "LIMIT" *)



(* Equivalence ST: *)

```
new_value:= LIMIT (min_value, value, max_value);
(* Limite la valeur à l'intervalle [min_value..max_value] *)

(* Equivalence IL: *)
LD          min_value
LIMIT       value, max_value
ST          new_value
```

MOD



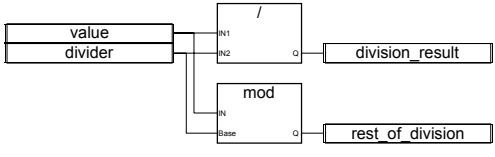
Arguments:

IN	ENT	toute valeur ENTIERE signée
Base	ENT	doit être plus grand que zéro
Q	ENT	résultat du **modulo** (input MOD base) rend -1 si Base <= 0

Description:

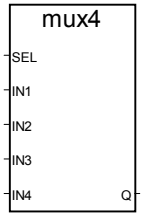
Calcule le modulo d'une valeur entière.

(* Programme FBD avec bloc "MOD" *)




```
(* Equivalence ST: *)
division_result:= (value / divider); (*division entière *)
rest_of_division:= MOD (value, divider); (* reste de la division *)

(* Equivalence IL: *)
LD          value
DIV         divider
ST          division_result
LD          value
MOD         divider
ST          rest_of_division
```

MUX4



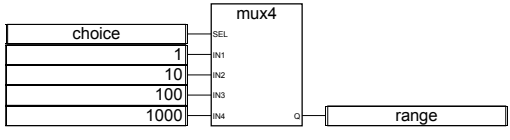
Arguments:

SEL	ENT	valeur entière de sélection dans l'intervalle [0..3]
IN1..IN4	ENT	valeurs entières signées
Q	ENT	= IN1 si SEL = 0 = IN2 si SEL = 1 = IN3 si SEL = 2 = IN4 si SEL = 3 = 0 pour toutes les autres valeurs de SEL

Description:

Multiplexeur 4 entrées: sélectionne une valeur parmi 4 valeurs entières.

(* Programme FBD avec bloc "MUX4" *)

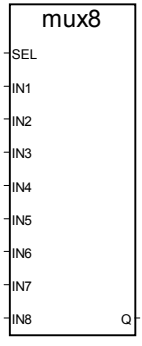


(* Equivalence ST: *)
range:= MUX4 (choice, 1, 10, 100, 1000);
(* sélectionne une valeur parmi 4 échelles prédéfinies, par exemple, si choice = 1, range sera égal à 10 *)

(* Equivalence IL: *)

```
LD      choice
MUX4    1,10,100,1000
ST      range
```

MUX8



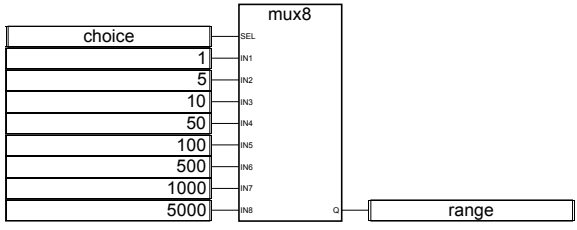
Arguments:

SEL	ENT	valeur entière de sélection dans l'intervalle [0..7]
IN1..IN8	ENT	valeurs entières signées
Q	ENT	= IN1 si SEL = 0 = IN2 si SEL = 1 ... = IN8 si SEL = 7 = 0 pour toutes les autres valeurs de SEL

Description:

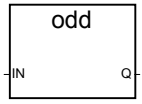
Multiplexeur 8 entrées: sélectionne une valeur parmi 8 valeurs entières.

(* Programme FBD avec bloc "MUX8" *)




```
(* Equivalence ST: *)
range:= MUX8 (choice, 1, 5, 10, 50, 100, 500, 1000, 5000);
(* sélectionne une valeur parmi 8 échelles prédéfinies, par exemple, si choice = 3, range sera
égal à 50 *)

(* Equivalence IL: *)
LD          choice
MUX8        1,5,10,50,100,500,1000,5000
ST          range
```

ODD



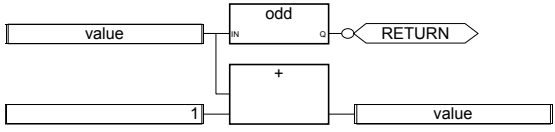
Arguments:

IN	ENT	toute valeur entière signée
Q	BOO	TRUE si la valeur est impaire FALSE si la valeur est paire

Description:

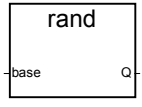
Teste la **parité** d'un entier: le résultat est impair ou pair.

(* Programme FBD avec bloc "ODD" *)




```
(* Equivalence ST: *)
If Not (ODD (value)) Then Return; End_if;
value:= value + 1;
(* rend la valeur toujours paire *)

(* Equivalence IL: *)
LD          value
ODD
RETNC
LD          value
ADD         1
ST          value
```

RAND



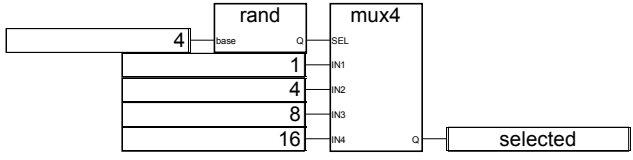
Arguments:

base	ENT	définit l'intervalle de valeurs autorisées
Q	ENT	valeur aléatoire dans l'intervalle [0..base-1]

Description:

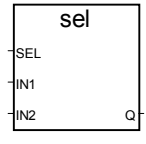
Retourne un **nombre aléatoire** (valeur entière) dans un intervalle donné.

(* Programme FBD avec bloc "RAND" *)




```
(* Equivalence ST: *)
selected:= MUX4 ( RAND (4), 1, 4, 8, 16 );
(*
sélection aléatoire d'une valeur parmi 4 valeurs prédéfinies.
La valeur issue de l'appel à RAND est dans l'intervalle [0..3],
Donc 'selected' issue de MUX4, prendra aléatoirement la valeur
1 si 0 est issue de RAND,
ou 4 si 1 est issue de RAND,
ou 8 si 2 est issue de RAND,
ou 16 si 3 est issue de RAND,
*)

(* Equivalence IL: *)
LD          4
RAND
MUX4        1,4,8,16
ST          selected
```

SEL



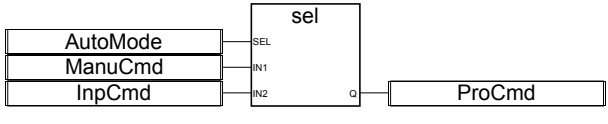
Arguments:

SEL	BOO	indique la valeur choisie
IN1, IN2	ENT	valeurs analogiques entières
Q	ENT	= IN1 si SEL = FALSE = IN2 si SEL = TRUE

Description:

Sélecteur binaire: sélectionne une valeur parmi deux valeurs entières.

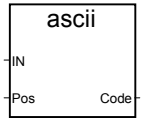
```
(* Programme FBD avec bloc "SEL" *)
```




```
(* Equivalence ST: *)
```

```
ProCmd:= SEL (AutoMode, ManuCmd, InpCmd);
(* sélection de commande pour le procédé *)

(* Equivalence IL: *)
LD          AutoMode
SEL         ManuCmd,InpCmd
ST          ProCmd
```

ASCII



Arguments:

IN	MSG	chaîne de caractères non vide
Pos	ENT	position du caractère sélectionné dans l'intervalle [1 .. long] (long est la longueur du message IN)
Code	ENT	code du caractère sélectionné (dans l'intervalle[0 .. 255]) retourne 0 si Pos n'est pas dans la chaîne

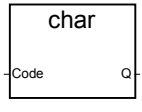
Description:

Donne le code ASCII d'un caractère dans une chaîne de caractère (message).

```
(* Programme FBD avec bloc "ASCII" *)
```


```
(* Equivalence ST: *)
FirstChr:= ASCII (message, 1);
(* FirstChr est le code Ascii du premier caractère de la chaîne *)

(* Equivalence IL: *)
LD          message
ASCII       1
ST          FirstChr
```

CHAR



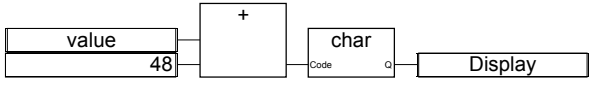
Arguments:

Code	ENT	code ascii dans l'intervalle [0 .. 255]

Q	MSG	chaîne de un caractère le caractère a le code ascii spécifié dans Code (le code ascii est pris modulo 256)

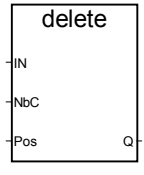
Description:
Donne un message de un caractère à partir de son code ASCII.

(* Programme FBD avec bloc "CHAR" *)




```
(* Equivalence ST: *)
Display:= CHAR ( value + 48 );
(* value est dans l'intervalle [0..9] *)
(* 48 est le code ASCII de '0' *)
(* le résultat est une chaîne de un caractère de '0' à '9' *)

(* Equivalence IL: *)
LD          value
ADD         48
CHAR
ST          Display
```

DELETE

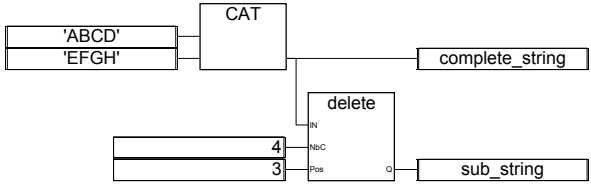


Arguments:

IN	MSG	chaîne de caractères non vide
NbC	ENT	nombre de caractères à supprimer
Pos	ENT	position du premier caractère à supprimer (le premier caractère de la chaîne a la position 1)
Q	MSG	chaîne modifiée chaîne vide si position < 1 chaîne initiale si Pos > longueur de la chaîne IN chaîne initiale si NbC <= 0

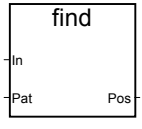
Description:
Détruit une partie(**sous-chaîne**) d'une chaîne de caractères.

(* Programme FBD avec bloc "DELETE" *)




```
(* Equivalence ST: *)
complete_string:= 'ABCD' + 'EFGH'; (* complete_string = 'ABCDEFGH' *)
sub_string:= DELETE (complete_string, 4, 3); (* sub_string = 'ABGH' *)

(* Equivalence IL: *)
LD          'ABCD'
ADD         'EFGH'
ST          complete_string
DELETE      4,3
ST          sub_string
```

FIND



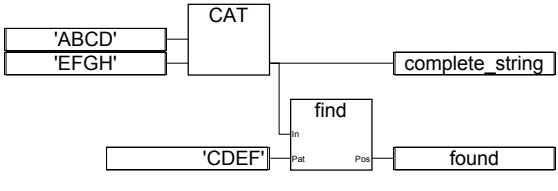
Arguments:

In	MSG	chaîne de caractères
Pat	MSG	toute chaîne non vide (Pattern)
Pos	ENT	= 0 si sous-chaîne Pat non trouvée = position du premier caractère de la première occurrence de la sous-chaîne Pat (la première position valide est 1) Cette fonction **différencie les majuscules et minuscules**

Description:

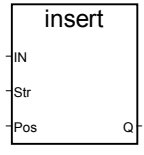
Recherche une sous-chaîne dans une chaîne message. Donne la position dans la chaîne de la sous-chaîne.

(* Programme FBD avec bloc "FIND" *)



```
(* Equivalence ST: *)
complete_string:= 'ABCD' + 'EFGH'; (* complete_string = 'ABCDEFGH' *)
found:= FIND (complete_string, 'CDEF'); (* found = 3 *)

(* Equivalence IL: *)
LD          'ABCD'
ADD         'EFGH'
ST          complete_string
FIND        'CDEF'
ST          found
```

INSERT



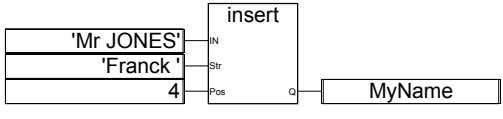
Arguments:

IN	MSG	chaîne initiale
Str	MSG	chaîne de caractères à insérer
Pos	ENT	position d'insertion l'insertion se fait avant la position donnée (la première position valide est 1)
Q	MSG	chaîne modifiée chaîne vide si Pos <= 0 concaténation des deux chaînes si Pos est plus grand que la longueur de la chaîne IN

Description:

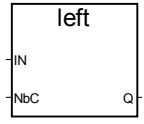
Insertion d'une sous-chaîne dans une chaîne message à une position donnée.

(* Programme FBD avec bloc "INSERT" *)




```
(* Equivalence ST: *)
MyName:= INSERT ('Mr JONES', 'Frank ', 4);
(* MyName = 'Mr Frank JONES' *)

(* Equivalence IL: *)
LD          'Mr JONES'
INSERT      'Frank ',4
ST          MyName
```

LEFT



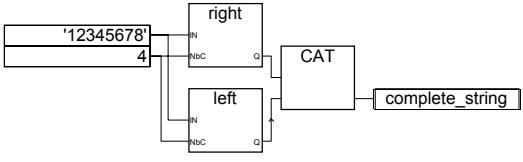
Arguments:

IN	MSG	chaîne de caractères non vide
NbC	INT	Nombre de caractères à extraire ne peut pas être plus grand que la longueur de la chaîne IN
Q	MSG	partie gauche de la chaîne IN (sa longueur = NbC) chaîne vide si NbC <= 0 chaîne IN initiale si NbC >= longueur de la chaîne IN

Description:

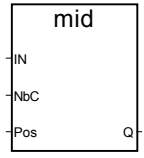
Extraction de la partie gauche d'une chaîne message. Le nombre de caractères à extraire est donné en paramètre.

(* Programme FBD avec blocs "LEFT" et "RIGHT" *)




```
(* Equivalence ST: *)
complete_string:= RIGHT ('12345678', 4) + LEFT ('12345678', 4);
(* complete_string = '56781234'
la valeur issue de l'appel à RIGHT = '5678'
la valeur issue de l'appel à LEFT = '1234'
*)

(* Equivalence IL: on fait d'abord l'appel à LEFT *)
LD          '12345678'
LEFT        4
ST          sub_string  (* résultat intermédiaire *)
LD          '12345678'
RIGHT       4
ADD         sub_string
ST          complete_string
```

MID



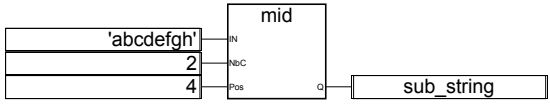
Arguments:

IN	MSG	chaîne de caractères non vide
NbC	ENT	nombre de caractères à extraire ne peut pas être plus grand que la longueur de la chaîne IN
Pos	ENT	position de la sous-chaîne le premier caractère de la sous-chaîne sera celui pointé par Pos (la première position valide est 1)
Q	MSG	partie du milieu de la chaîne (sa longueur = NbC) chaîne vide si les paramètres sont invalides

Description:

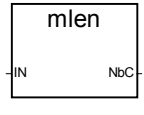
Extraction d'une sous-chaîne au milieu d'une chaîne message. Le nombre de caractères à extraire et la position du premier caractère sont donnés en paramètre.

(* Programme FBD avec bloc "MID" *)




```
(* Equivalence ST: *)
sub_string:= MID ('abcdefgh', 2, 4);
(* sub_string = 'de' *)

(* Equivalence IL: *)
LD      'abcdefgh'
MID     2,4
ST      sub_string
```

MLEN

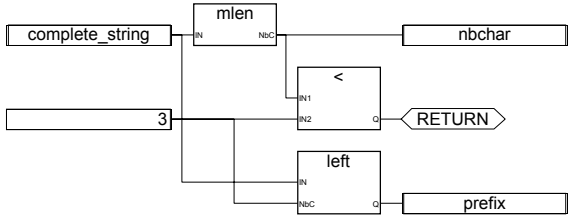


Arguments:

IN	MSG	chaîne de caractères
NbC	INT	nombre de caractères dans la chaîne IN

Description:
Calcule la **longueur d'une chaîne** message.

(* Programme FBD avec bloc "MLEN" *)

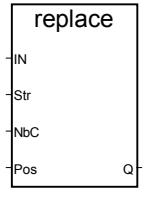


(* Equivalence ST: *)
nbchar:= MLEN (complete_string);
If (nbchar < 3) Then Return; End_if;
prefix:= LEFT (complete_string, 3);
(* ce programme extrait les 3 caractères à gauche de la chaîne et met le résultat dans la variable message 'prefix'.
Rien n'est fait si la longueur de la chaîne est inférieure à 3 *)

(* Equivalence IL: *)

```
LD      complete_string
MLEN
ST      nbchar
LT      3
RETC
LD      complete_string
LEFT    3
ST      prefix
```

REPLACE



Arguments:

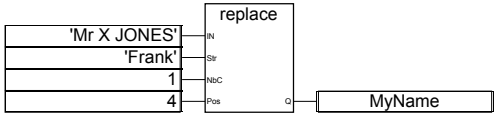
IN	MSG	chaîne de caractères
Str	MSG	chaîne à insérer (pour remplacer NbC caractères)
NbC	ENT	nombre de caractères à détruire
Pos	ENT	position du premier caractère modifié (la première position valide est 1)
Q	MSG	chaîne modifiée: - NbC caractères sont détruits à partir de Pos

- ensuite la sous-chaîne Str est insérée à cette position
rend une chaîne vide si Pos <= 0
rend la concaténation des chaînes(IN+Str) si Pos est plus grand que la longueur de la chaîne IN
rend la chaîne initiale IN si NbC <= 0

Description:

Remplacement d'une partie d'une chaîne message par une nouvelle sous-chaîne.

(* Programme FBD avec bloc "REPLACE" *)

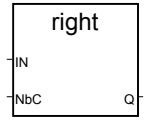


(* Equivalence ST: *)
MyName:= REPLACE ('Mr X JONES, 'Frank', 1, 4);
(* MyName = 'Mr Frank JONES' *)

(* Equivalence IL: *)

LD	'Mr X JONES'
REPLACE	'Frank',1,4
ST	MyName

RIGHT



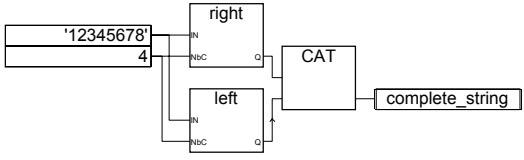
Arguments:

IN	MSG	chaîne de caractères non vide
NbC	ENT	Nombre de caractères à extraire ne peut pas être plus grand que la longueur de la chaîne IN
Q	MSG	partie droite de la chaîne IN (sa longueur = NbC) chaîne vide si NbC <= 0 chaîne IN initiale si NbC >= longueur de la chaîne IN

Description:

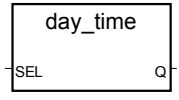
Extraction de la partie droite d'une chaîne message. Le nombre de caractères à extraire est donné en paramètre.

(* Programme FBD avec blocs "LEFT" et "RIGHT" *)




```
(* Equivalence ST: *)
complete_string:= RIGHT ('12345678', 4) + LEFT ('12345678', 4);
(* complete_string = '56781234'
la valeur issue de l'appel à RIGHT = '5678'
la valeur issue de l'appel à LEFT = '1234'
*)

(* Equivalence IL: on fait d'abord l'appel à LEFT *)
LD          '12345678'
LEFT        4
ST          sub_string  (* résultat intermédiaire *)
LD          '12345678'
RIGHT       4
ADD         sub_string
ST          complete_string
```

DAY_TIME



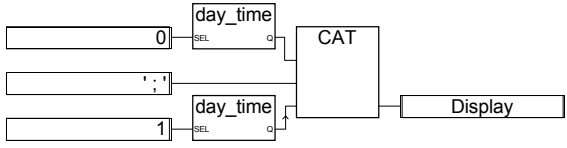
Arguments:

SEL	ENT	sélection de l'information 0= date courante 1= heure courante 2= jour de la semaine
Q	MSG	date/heure sous forme de chaîne de caractère 'AAAA/MM/JJ' si SEL = 0 'HH:MM:SS' si SEL = 1 nom du jour SEL = 2 (ex: 'Monday') (les noms de jour sont toujours en anglais)

Description:

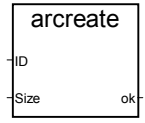
Donne la date ou l'heure ou le jour courant sous forme de chaîne message.

(* Programme FBD avec bloc "DAY_TIME" *)




```
(* Equivalence ST: *)
Display:= Day_Time (0) + ' ; ' + Day_Time (1);
(* Le format texte de Display est: 'YYYY/MM/DD ; HH:MM:SS' *)

(* Equivalence IL: on fait d'abord l'appel à day_time(1) *)
LD              1
DAY_TIME
ST              hour_str     (* résultat intermédiaire *)
LD              0
DAY_TIME
ADD             ' ; '
ADD             hour_str
ST              Display
```

ARCREATE



Arguments:

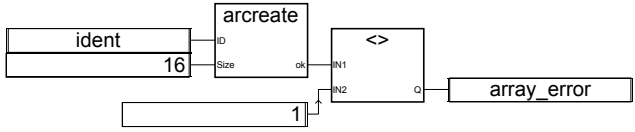
ID	ENT	identificateur du tableau (doit être dans l'intervalle [0..15])
Size	ENT	nombre d'éléments du tableau à créer
ok	ENT	compte rendu d'exécution: **1** = si le tableau a été crée avec succès **2** = identificateur invalide ou tableau déjà créé **3** = taille invalide **4** = pas assez de mémoire

Description:

Création d'un tableau d'entiers.

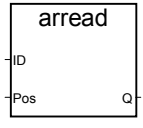
Attention: Il ne peut pas y avoir plus de **16** tableaux dans une application. Les tableaux contiennent des valeurs **analogiques entières**. La mémoire étant allouée dynamiquement, cette procédure peut endommager le système et provoquer des erreurs fatales si la taille demandée est trop proche de la taille de la mémoire système disponible.

(* Programme FBD avec gestion de tableaux d'entiers *)




```
(* Equivalence ST: *)
array_error:= (ARCREATE (ident, 16) <> 1));

(* Equivalence IL: *)
LD              ident
ARCREATE        16
NE              1
ST              array_error
```

ARREAD



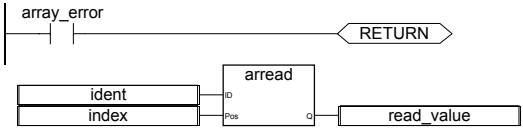
Arguments:

ID	ENT	identificateur du tableau (doit être dans l'intervalle [0..15])
Pos	ENT	position de l'élément dans le tableau dans l'intervalle [0 .. size-1]
value	ENT	valeur de l'élément lu 0 si les arguments sont invalides

Description:

Lit un élément dans un tableau d'entiers.

(* Programme FBD avec gestion de tableaux *)




```
(* Equivalence ST: *)
If (array_error) Then Return; End_if;
read_value:= ARREAD (ident, index);
(* array_error vient de l'appel à ARCREATE *)

(* Equivalence IL: *)
LD              array_error
RETC
```

```
LD          ident
ARREAD      index
ST          read_value
```

ARWRITE

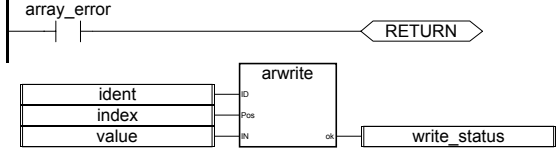
Arguments:

ID	ENT	identificateur du tableau (doit être dans l'intervalle [0..15])
Pos	ENT	position de l'élément dans le tableau dans l'intervalle [0 .. size-1]
IN	ENT	nouvelle valeur pour l'élément
ok	ENT	compte rendu d'exécution **1** = l'écriture a réussi **2** = identificateur de tableau invalide **3** = indice (Pos) invalide

Description:

Stocke (**écrit**) une valeur dans un **tableau** d'entiers.

(* Programme FBD avec gestion de tableaux *)




```
(* Equivalence ST: *)
If (array_error) Then Return; End_if;
write_status:= ARWRITE (Ident, Index, value);
(* array_error vient de l'appel à ARCREATE *)

(* Equivalence IL: *)
LD          array_error
RETC
LD          ident
ARWRITE     index,value
ST          write_status
```

F_ROPEN

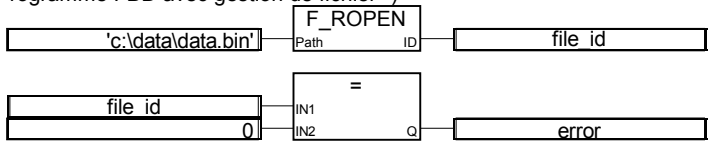
Arguments:

Path	MSG	nom de fichier peut inclure le chemin d'accès au fichier avec les symboles \ ou / pour spécifier un répertoire. Pour améliorer la portabilité de l'application, / et \ sont équivalents.
ID	ENT	identificateur de fichier 0 en cas d'erreur: le fichier n'existe pas.

Description:

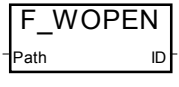
Ouvre un fichier binaire en mode **lecture**. Doit être utilisé avec FX_READ et F_CLOSE. Cette fonction n'est pas incluse dans le simulateur ISaGRAF.

(* Programme FBD avec gestion de fichier *)




```
(* Equivalence ST: *)
file_id:= F_ROPEN('c:\data \data.bin');
error:= (file_id=0);

(* Equivalence IL: *)
LD          'c:\data\data.bin'
F_ROPEN
ST          file_id
EQ          0
ST          error
```

F_WOPEN



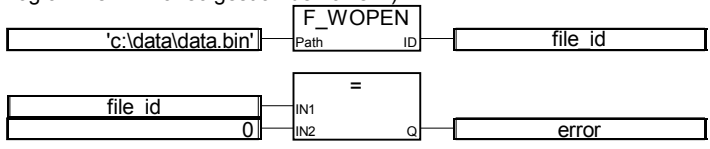
Arguments:

Path	MSG	nom de fichier peut inclure le chemin d'accès au fichier avec les symboles \ ou / pour spécifier un répertoire. Pour améliorer la portabilité de l'application, / et \ sont équivalents.
ID	ENT	identificateur de fichier 0 en cas d'erreur. Si le fichier existe déjà il est écrasé.

Description:

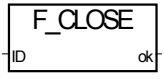
Ouvre un fichier binaire en mode **écriture**. Doit être utilisé avec FX_WRITE et F_CLOSE.
Cette fonction n'est pas incluse dans le simulateur ISaGRAF.

(* Programme FBD avec gestion de fichier *)




```
(* Equivalence ST: *)
file_id:= F_WOPEN('c:\data \data.bin');
error:= (file_id=0);

(* Equivalence IL: *)
LD          'c:\data\data.bin'
F_WOPEN
ST          file_id
EQ          0
ST          error
```

F_CLOSE

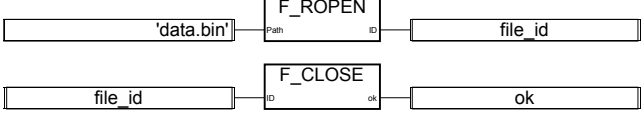


Arguments:

ID	ENT	identificateur de fichier: retourné par F_ROPEN ou F_WOPEN.
ok	BOO	statut TRUE si la fermeture de fichier s'est bien passée FALSE en cas d'erreur

Description:
Ferme un fichier binaire ouvert avec la fonction F_ROPEN ou F_WOPEN.
Cette fonction n'est pas incluse dans le simulateur ISaGRAF.

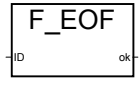
(* Programme FBD avec gestion de fichier *)




```
(* Equivalence ST: *)
file_id:= F_ROPEN('data.bin');
ok:= F_CLOSE(file_id);
```

```
(* Equivalence IL: *)
LD              'data.bin'
F_ROPEN
ST              file_id
F_CLOSE                 (* file_id est déjà dans le résultat courant IL *)
ST              ok
```

F_EOF



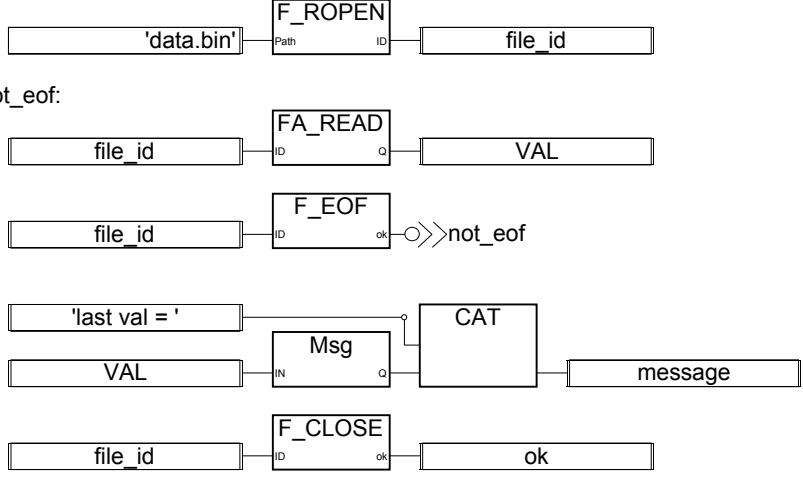
Arguments:

ID	ENT	identificateur de fichier: retourné par F_ROPEN ou F_WOPEN.
ok	BOO	indicateur de fin de fichier TRUE si la fin du fichier a été atteinte au moment du dernier appel aux procédures de lecture ou d'écriture. Avec FM_READ, le dernier message lu dans un fichier peut ne pas être correcte, si le dernier caractère n'est pas un caractère de fin de chaîne.

Description:

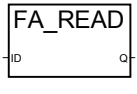
Teste si la **fin de fichier** est atteinte.
Cette fonction n'est pas incluse dans le simulateur ISaGRAF.
Notez que EOF en anglais signifie End Of File.

(* Programme FBD avec gestion de fichier *)



```
(* Equivalence ST: *)
file_id:= F_ROPEN('data.bin');
WHILE not(F_EOF(file_id))
        VAL:= FA_READ(file_id);
END_WHILE;
MESSAGE:= 'last val = ' + msg(VAL);
ok:= F_CLOSE(file_id);
(* Equivalence IL: *)
                LD          'data.bin'
                F_ROPEN
                ST          file_id
                LD          file_id
                F_EOF
                JMPC        END_OF_FILE
NOT_EOF:        LD          file_id
                FA_READ
                ST          VAL
                LD          file_id
                F_EOF
                JMPNC       NOT_EOF  (* si pas fin de fichier:eof, continuer la lecture *)
END_OF_FILE:LD              VAL
                MSG
                ST          val_msg     (* conversion de VAL en message *)
                LD          'last val = '
                ADD         val_msg
                ST          MESSAGE
                LD          file_id
                F_CLOSE
                ST          ok
```

FA_READ



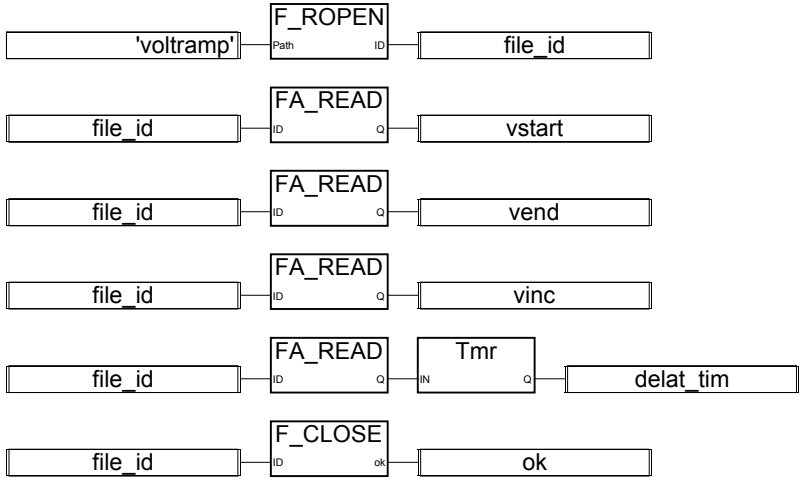
Arguments:

ID	ENT	identificateur de fichier: retourné par F_ROPEN
Q	ENT	valeur analogique entière lue dans le fichier

Description:

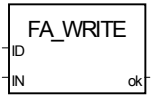
Lit des valeurs analogiques dans un fichier binaire. Doit être utilisé avec F_ROPEN et F_CLOSE.
Cette procédure accède de façon séquentielle au fichier, à partir de la dernière position.
Le premier appel après F_ROPEN lit les 4 premiers octets dans le fichier,
chaque appel décale le pointeur de lecture.
Pour tester si la fin du fichier est atteinte, utilisez F_EOF.
Cette fonction n'est pas incluse dans le simulateur ISaGRAF.

(* Programme FBD avec gestion de fichier *)




```
(* Equivalence ST: *)
file_id:= F_ROPEN('voltramp.bin');
vstart:= FA_READ(file_id);
vend:= FA_READ(file_id);
vinc:= FA_READ(file_id);
delta_tim:= tmr(FA_READ(file_id));
ok:= F_CLOSE(file_id);
(* Equivalence IL: *)
                LD          'voltramp.bin'
                F_ROPEN
                ST          file_id
                FA_READ                 (* lit vstart *)
                ST          vstart
                LD          file_id
                FA_READ                 (* lit vend *)
                ST          vend
                LD          file_id
                FA_READ                 (* lit vinc *)
                ST          vinc
                LD          file_id
                FA_READ                 (* lit delta_tim *)
                TMR                     (* conversion en temporisation *)
                ST          delta_tim
                LD          file_id
                F_CLOSE
                ST          ok
```

FA_WRITE



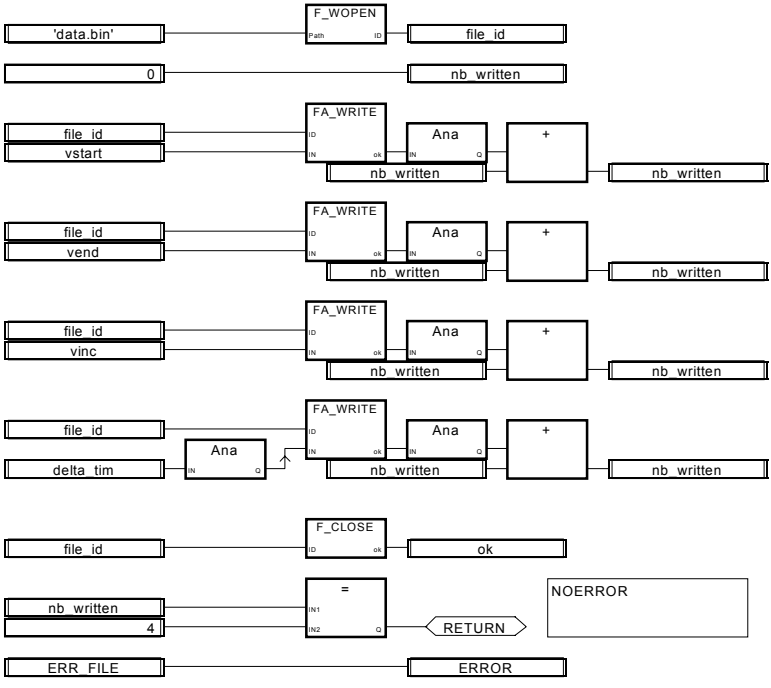
Arguments:

ID	ENT	identificateur de fichier: retourné par F_WOPEN
IN	ENT	valeur analogique entière à écrire dans le fichier
OK	BOO	compte rendu d'exécution: TRUE si ok

Description:

Ecrit des valeurs analogiques dans un fichier binaire.
Cette procédure accède de façon séquentielle au fichier, à partir de la dernière position.
Le premier appel après F_WOPEN écrit les 4 premiers octets dans le fichier,
chaque appel décale le pointeur d'écriture.
Cette fonction n'est pas incluse dans le simulateur ISaGRAF.

(* Programme FBD avec gestion de fichier *)



(* Equivalence ST: *)
file_id:= F_WOPEN('voltramp.bin');
nb_written := 0;
nb_written:= nb_written + ana(FA_WRITE(file_id,vstart));
nb_written:= nb_written + ana(FA_WRITE(file_id,vend));

```
nb_written:= nb_written + ana(FA_WRITE(file_id,vinc));
nb_written:= nb_written + ana(FA_WRITE(file_id,ana(delta_tim)));
ok:= F_CLOSE(file_id);
IF ( nb_written <> 4) THEN
        ERROR:= ERR_FILE;
END_IF;

(* Equivalence IL: *)
        LD      'voltramp.bin'
        F_ROPEN
        ST      file_id
        LD      0
        ST      nb_written
        LD      file_id     (* écrit vstart *)
        FA_WRITE vstart
        ANA
        ADD     nb_written
        ST      nb_written
        LD      file_id     (* écrit vend *)
        FA_WRITE vend
        ANA
        ADD     nb_written
        ST      nb_written
        LD      file_id     (* écrit vinc *)
        FA_WRITE vinc
        ANA
        ADD     nb_written
        LD                  (* écrit delta_tim *)
        ANA                 (* conversion en entier *)
        ST      ana_delta_tim
        LD      file_id
        FA_WRITE ana_delta_tim
        ANA
        ADD     nb_written
        ST      nb_written
        F_CLOSE
        ST      ok
        LD      nb_written
        EQ      4
        RETC                (* return si égal 4 *)
        LD      ERR_FILE    (* sinon erreur *)
        ST      ERROR
```

FM_READ

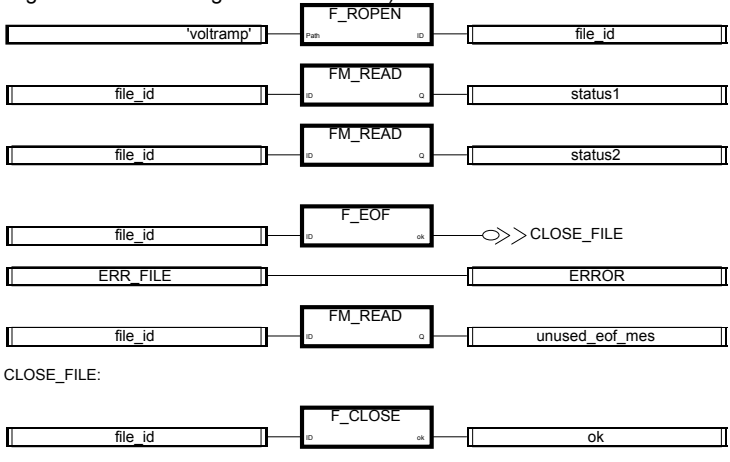
Arguments:

ID	ENT	identificateur de fichier: retourné par F_ROPEN
Q	MSG	valeur du message lu dans le fichier

Description:

Lit des variables messages dans un fichier binaire.
Doit être utilisé avec F_ROPEN et F_CLOSE.
Cette procédure accède de façon séquentielle au fichier, à partir de la dernière position.
Le premier appel après F_ROPEN lit la première chaîne dans le fichier,
chaque appel décale le pointeur de lecture.
Une chaîne finit par nul (0), le caractère fin de ligne ('\n') ou retour ('\r');
Pour tester si la fin du fichier est atteinte, utilisez F_EOF.
Cette fonction n'est pas incluse dans le simulateur ISaGRAF.

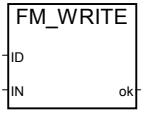
(* Programme FBD avec gestion de fichier *)




```
(* Equivalence ST: *)
file_id:= F_ROPEN('voltramp.bin');
status1:= FM_READ(file_id);
status2:= FM_READ(file_id);
IF (F_EOF(file_id)) THEN
        ERROR:= ERR_FILE;
        unused_eof_mes:= FM_READ(file_id);
END_IF;
ok:= F_CLOSE(file_id);
(* Equivalence IL: *)
                LD      'voltramp.bin'
                F_ROPEN
                ST      file_id
                FM_READ             (* lit status1 *)
                ST      status1
                LD      file_id
                FM_READ             (* lit status2 *)
                ST      status2
                LD      file_id
                F_EOF
                JMPNC   CLOSE_FILE (* si fin de fichier, saut non effectué *)
```

```
            LD      ERR_FILE
            ST      ERROR
            LD      file_id
            FM_READ             (* lecture unused_eof_mes *)
            ST      unused_eof_mes
CLOSE_FILE  LD      file_id
            F_CLOSE
            ST      ok
```

FM_WRITE



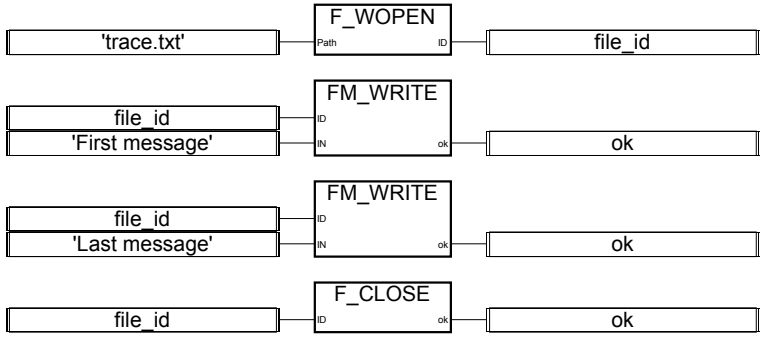
Arguments:

ID	ENT	identificateur de fichier: retourné par F_WOPEN
IN	MSG	valeur du message à écrire dans le fichier
ok	BOO	compte rendu d'exécution TRUE en cas de succès

Description:

Ecrit des variables message dans un fichier binaire.
Doit être utilisé avec F_WOPEN et F_CLOSE.
Un message est écrit dans le fichier comme une chaîne terminée par le caractère nul.
Cette procédure accède de façon séquentielle au fichier, à partir de la dernière position.
Le premier appel après F_WOPEN écrit la première chaîne dans le fichier,
Cette fonction n'est pas incluse dans le simulateur ISaGRAF.

(* Programme FBD avec gestion de fichier *)



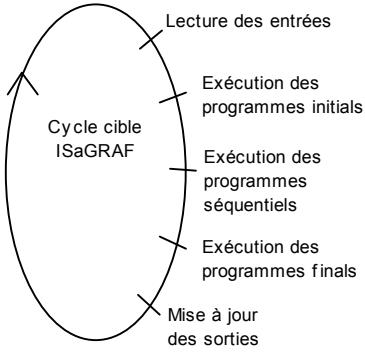

```
(* Equivalence ST: *)
file_id:= F_WOPEN('trace.txt');
ok:= FM_WRITE(file_id,'First message');
```

```
ok:= FM_WRITE(file_id,'Last message');
ok:= F_CLOSE(file_id);

(* Equivalence IL: *)
                LD          'trace.txt'
                F_WOPEN
                ST          file_id
                FM_WRITE'First message'      (* écrit le premier message *)
                ST          ok
                LD          file_id
                FM_WRITE'Last message'       (* écrit le deuxième message *)
                ST          ok
                LD          file_id
                F_CLOSE
                ST          ok
```

C. Manuel d'utilisation de la cible

C.1 Introduction

La cible ISaGRAF est un logiciel temps réel qui exécute une application ISaGRAF sur votre ordinateur industriel ou votre carte selon le schéma suivant:



Un cycle de la cible se compose de la lecture des entrées physiques du procédé à exécuter, du traitement des données de l'application selon les programmes de l'atelier ISaGRAF[1] et de la mise à jour des sorties physiques.

- La première partie de cette section décrit la mise en route des systèmes cibles spécifiques (DOS, OS-9, VxWorks und NT). Pour chacun de ces systèmes, cette partie explique d'abord comment exécuter la cible ISaGRAF. Ensuite des informations sont données sur des fonctionalités spécifiques, comme la mise en marche de la cible au moment de la mise sous tension, la gestion des erreurs, le comportement général...

- La deuxième partie décrit la méthode d'implémentation des fonctions, blocs fonctionnels et fonctions de conversions utilisateur en "C" afin d'améliorer les performances de la cible ISaGRAF.

- La troisième partie contient des informations sur Modbus et l'implémentation d'ISaGRAF. Elle décrit les formats standards des différents codes de fonction.

- La quatrième partie décrit quelques outils pour la gestion des pannes de courant et de la remise en marche de la cible.

[1] Ce manuel assume que le lecteur est familier avec l'utilisation de l'atelierISaGRAF.

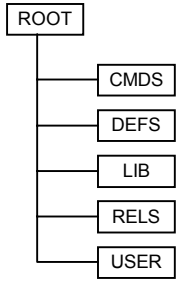
C.2 Installation

L'Installation demande 1 MOctet de mémoire sur votre disque dur.

L'installation est réalisée avec le programme install.bat qui se trouve sur la disquette d'installation. Ce programme installe tous les fichiers nécessaires pour une plateforme spécifique sur votre disque dur.

Exemple: a:\install a: c:*path*
installe les fichiers à partir du lecteur de disquettes a: sur c: dans le répertoire *path*.

L'architecture suivante est utilisée:



Le répertoire ROOT contient quelques outils et les fichiers Readme
Le répertoire CMDS contient les fichiers exécutables
Le répertoire DEFS contient les fichiers des définitions
Le répertoire LIB contient les librairies
Le répertoire RELS contient les fichiers relogeables (d'objets)
Le répertoire USER contient les procédures "C" pour les fonctions, blocs fonctionnels et fonctions de conversions en "C" (fichiers de source et d'entête)

Maintenant, vous pouvez mettre en route la plateforme installée.

C.3 Mise en route de la cible ISaGRAF DOS

C.3.1 Exécution d'ISaGRAF: ISA.EXE

Dans l'implémentation MS-DOS, la cible est exécutée en un programme unique: ISA.EXE
Pour la mettre en marche il suffit d'exécuter la commande d'aide isa -? dans le répertoire CMDS.

Dans un tel système les opérations peuvent être critiques. Par exemple il est recommandé de ne pas surcharger la liaison de communication pour garantir une bonne performance.
Le programme cible n'empêche pas l'exécution des routines contrôlées par interruption.

Liaison de communication et configuration: option -t

La cible ISaGRAF utilise une liaison série pour la communication avec le debugger. Le nom de la liaison est défini à l'aide de l'option -t. Puisque l'interface de communication a été conçue pour être compatible avec tous types de machines, les ports COM1, COM2 ou COM3 peuvent être utilisés, suivant la version BIOS.

Sans valeur par défaut: Si cette option n'est pas utilisée, la communication avec la cible n'est pas possible. Dans ce cas l'erreur n° 7 s'affichera.

Avec la cible ISaGRAF DOS, la communication utilisant une liaison Ethernet n'est pas disponible. Renseignez-vous auprès de votre fournisseur pour une implémentation spéciale.

Avant la mise en marche d' ISaGRAF, il est nécessaire de régler les paramètres de communication afin que l'utilisateur puisse utiliser les paramètres nécessaires en toute liberté. Si vous utilisez le debugger de l'atelier, assurez-vous que les paramètres de communication de l'atelier (voir Guide d'utilisateur: Gestion des programmes) correspondent à ceux de la cible.

Exemple:
MODE COM1:9600,N,8,1
Règle les paramètres de communication avec les valeurs suivantes:
- Vitesse 9600 baud
- sans parité
- 8 bits de données
- 1 bit de stop

Notez que la valeur de 19200 bauds, donnée par défaut dans l'atelier, n'est pas autorisée pour certaines versions BIOS.
CJ fournit l'utilitaire ISAMOD.EXE pour le règlement des paramètres de l'atelier:
ISAMOD COM1
est équivalent à MODE COM1:19200,N,8,1

Numéro d'esclave: option -s

Cette option définit le numéro d'esclave de la cible. Il est compris entre 1 et 255, à l'exception du numéro 13 ($0D). Le numéro d'esclave est utilisé par le protocole de la liaison de

communication et sert principalement à distinguer différents esclaves lorsque plusieurs cibles sont connectées ensemble. Si vous utilisez le debugger de l'atelier, assurez-vous que le paramètre de communication de l'atelier (voir Guide d'utilisateur: Gestion des programmes) correspond à celui de la cible.

Valeur par défaut: Le numéro d'esclave par défaut est 1 (comme pour l'atelier)

Exemples:

isamod COM1 Configure COM1 à 19200 baud, sans parité, 8 bits de données, 1 bit de stop.

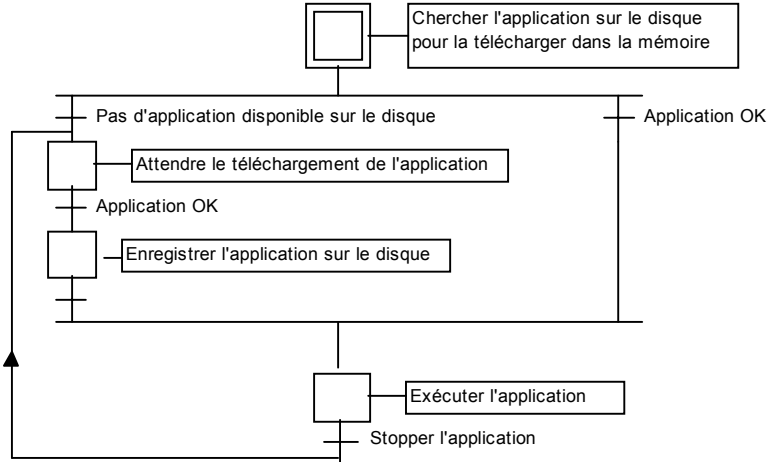
isa -t=COM1 Lance la cible ISaGRAF avec le numéro d'esclave par défaut (1) et avec COM1 comme port de communication.

isa -s=3 -t=COM1 Lance la cible ISaGRAF avec le numéro d'esclave 3 et avec COM1 comme port de communication.

C.3.2 Fonctionalités spécifiques

Lancement de la cible ISaGRAF

Au lancement de la cible l'algorithme suivant est exécuté.



- **Définitions**

Le code de l'application est la base de données binaire générée et téléchargée par l'atelier et ensuite exécutée par la cible. Il peut être complété par la table des symboles.
La table des symboles d'une application est une base de données ASCII générée et téléchargée par l'atelier. Cette table réalise la liaison entre les objets symboliques et les objets internes de la cible. Elle n'est pas indispensable dans la cible, sauf pour la gestion des symboles spécifiques de l'utilisateur. Pour plus d'informations sur la table des symboles voir le Guide d'Utilisateur: Techniques de programmation avancées.

- **Sauvegarde de l'application**

Lorsqu'une nouvelle application est téléchargée dans la cible, le code de l'application est enregistré dans le répertoire courant de la cible sous le nom:

ISAx1 Fichier de sauvegarde du code d'application (x est le numéro d'esclave)

Par ailleurs, si la table des symboles de l'application a été téléchargée auparavant, elle est aussi enregistrée dans le répertoire courant de la cible sous le nom:

ISAx6 Fichier de sauvegarde des symboles de l'application ISaGRAF (x est le numéro d'esclave)

Au lancement de la cible ISaGRAF, ces fichiers sont recherchés dans le répertoire courant et téléchargés dans la mémoire.
Si le fichier des symboles n'est pas disponible, la cible commence l'exécution du code d'application sans les symboles.
Si aucun code d'application n'est diponible, la cible attend le téléchargement d'une application.

Afin de lancer la cible avec une application spécifique au moment de la mise sous tension, sans utiliser le debugger, on peut copier ces fichiers à partir du disque dur (si l'atelier se trouve sur le même PC) ou à l'aide d'une disquette directement dans le répertoire courant de la cible. Si votre cible ne possède pas de disque, vous pouvez utiliser un disque virtuel.

Lorsque l'atelier ISaGRAF est installé dans le répertoire standard \ISAWIN, le fichier du code d'application du projet MYPROJ s'appelle:

\ISAWIN\APL\MYPROJ\appli.x8m

et le fichier des symboles de l'application du projet MYPROJ s'appelle:

\ISAWIN\APL\MYPROJ\appli.tst

Exemple:
A partir du répertoire, dans lequel isa.exe est installé, lorsque la commade suivante est entrée:

copy \ISAWIN\APL\MYPROJ\appli.x8m isa11

Alors isa.exe recherchera l'application 'myproj' et l'exécutera.

Il est possible de gouper ces commandes dans un fichier Batch, par exemple, et de les lancer ensuite à partir du menu d'outils de l'atelier (voir Guide d'Utilisateur: Gestion des programmes).

Gestion des erreurs et messages de sortie

Le logiciel de la cible ISaGRAF permet la gestion de la détection d'erreurs. En annexe vous trouverez la liste des messages d'erreurs avec leur description.

La détection d'erreurs est traitée de la manière suivante:

- Une erreur se compose d'une erreur et d'un numéro d'argument qui sont envoyés à la routine d'erreurs d' ISaGRAF.
- Si la détection d'erreurs a été validée dans les paramètres d'exécution de l'atelier, l'erreur sera traitée. Sinon les informations seront perdues et la détection d'erreurs s'arrètera là.

Si les erreurs sont traitées:

- Le numéro d'erreur (valeur décimale) et l'argument (valeur hexadécimale) s'afficheront dans la sortie par défaut stdout.

- Le numéro d'erreur et l'argument seront envoyés dans un buffer de type FIFO, pour pouvoir être récupérés plus tard. La taille du buffer est définie dans les options "paramètres d'exécution" de l'atelier. Lorsque le buffer est plein, à chaque arrivée d'une nouvelle erreur la plus ancienne erreur sera perdue.
- Les erreurs peuvent être appelées soit par le debugger soit par l'application courante en utilisant l'appel SYSTEM (voir Guide d'utilisateur).

Si le debugger a détecté une erreur, un message avec la description de l'erreur s'affiche dans la fenêtre d'erreur. Selon le contexte de l'application (en exécution ou non) le debugger peut afficher le nom de l'objet (variable ou programme) dans lequel se trouve l'erreur, ou bien l'erreur d'argument (valeur décimale) entre parenthèses [x] ce qui peut avoir une signification différente pour chaque erreur.

Un message de bienvenue et les valeurs d'erreurs sont affichés dans la sortie stdout, lorsque la cible démarre et si des erreurs ont été détectées. Si cette affichage n'est pas souhaitée sur le canal de sortie standard, une commande de redirection peut être utilisée, comme:

isa -t=COM1 -s=1 **>NUL**

Horloge système

Puisque la cible ISaGRAF a été conçue pour être exécutée sur tous systèmes, la référence temporelle utilisée pour la synchronisation du cycle, ainsi que pour la mise à jour des variables de temporisation, est le tic standard d'environ 55 millisecondes.

Ainsi, il est impossible d'obtenir une précision plus grande que 55 ms pour les variables de temporisations. Pour la même raison un temps de cycle de moins ou égal à 55 ms et non égal à zéro génere un dépassement de temps de cycle (erreur 62) et des cycles non déclenchés.

Il est déconseillé de modifier le tic système pour éviter que les applications résidentes, ou les fonctions et blocs fonctionnels en "C" intégrés dans l'application, soient perturbés par l'exécution d' ISaGRAF.

Demandez une implémentation spécifique à votre fournisseur si votre application a besoin de plus de précision.

Quitter ISaGRAF

En testant une application sous des conditions non-industrielles sur un PC, l'utilisateur peut arrêter ISaGRAF en tapant une combinaison complexe de touches, évitant des arrêts non voulus. Cette séquence de touches est la suivante:

shift + ctrl + alt

Naturellement, si l'application industrielle ne doit pas être arrêtée par une touche, on devrait désactiver ces combinaisons.

Un effet secondaire dangereux de ces sorties rapides est la suivante: l'interface des cartes E/S n'est pas fermée. Pour cette raison la manière correcte d'arrêter la cible ISaGRAF est la suivante:
- arrêter l'application à partir du debugger (les cartes E/S sont fermées)
- stopper la cible ISaGRAF à partir du clavier.

Taille de l'application

Puisque la cible ISaGRAF MS-DOS a été conçue pour Intel Real Mode, la taille maximale d'une structure de données est 64K. Le code d'application téléchargé par l'atelier ne doit donc pas dépasser cette limite. Dans des cas très rares il est possible que même la structure interne allouée par ISaGRAF dépasse cette limite et que votre application échoue aprés téléchargement. La mémoire totale disponible est limitée à 640K de mémoire conventionelle.

Veuillez-vous adresser à votre fournisseur, si votre application a besoin de plus de capacité mémoire.

C.4 Mise en route de la cible ISaGRAF OS9

Tout d'abord il est nécessaire de transférer certains fichiers (au moins les fichiers exécutables du répertoire CMDS) à l'aide d'un outil de transfert de données sur votre cible OS-9.
Pour la mise en marche il suffit de lancer les commandes d'aide dans le répertoire CMDS de votre système OS-9:

```
isa -?
isaker -?
isatst -?
Isanet -?
```

C.4.1 Exécution d'ISaGRAF mono-tâche: isa

Il est possible d'exécuter la cible ISaGRAF en une seule tâche, mais les opérations dans une telle configuration peuvent être critiques. Par exemple, il est recommandé de ne pas surcharger la liaison de communication pour garantir une bonne performance. Le système OS-9 multi-tâches permet d'exécuter plusieurs cibles ISaGRAF mono-tâches sur une seule unité centrale à condition qu'elles possèdent différents numéros d'esclave et ports de communication.
L'implémentation mono-tâche a été conçue principalement pour certaines plateformes matérielles, comme les cartes ou PCs MS-DOS, ou pour créer un premier prototype afin de le porter sur une nouvelle plateforme. Pour cette raison, on devrait préférer la solution de l'implémentation multi-tâches de la cible ISaGRAF.
La cible ISaGRAF mono-tâche n'empêche pas l'exécution des traitements d'arrière-plan ou des routines contrôlées par interruption.

Liaison de communication et configuration: option -t

La cible ISaGRAF mono-tâche utilise une liaison série pour la communication avec le debugger. Le nom du descripteur est défini à l'aide de l'option -t.

Sans valeur par défaut: Si cette option n'est pas utilisée, la communication avec la cible n'est pas possible. Dans ce cas l'erreur n° 7 s'affichera peut-être.

Avec l'implémentation mono-tâche, la communication utilisant une liaison Ethernet n'est pas disponible.

Un dispositif de liaison série est ouvert en mode de transfert de données binaires (sans caractères de contrôle, sans XON/XOFF). Avant le lancement d' ISaGRAF, il est nécessaire de régler les paramètres de communication afin que l'utilisateur puisse utiliser les paramètres nécessaires en toute liberté. Si vous utilisez le debugger de l'atelier, assurez-vous que les paramètres de communication de l'atelier (voir Guide d'utilisateur: Gestion des programmes) correspondent à ceux de la cible.
Exemple:
xmode /t0 baud=19200
Règle la vitesse de communication à 19200 Baud sur le terminal /t0.

Numéro d'esclave: option -s

Cette option definit le numéro d'esclave de la cible. Il est compris entre 1 et 255, à l'exception du numéro 13 ($0D). Le numéro d'esclave est utilisé par le protocole de la liaison de communication et sert principalement à distinguer différents esclaves lorsque plusieurs cibles sont exécutées. Si vous utilisez le debugger de l'atelier, assurez-vous que le paramètre de communication de l'atelier (voir Guide d'utilisateur: Gestion des programmes) correspond à une cible existante.

Valeur par défaut: Le numéro d'esclave par défaut est 1 (comme dans l'atelier)

Exemples:

isa -t=/t0 Lance une cible ISaGRAF mono-tâche avec le numéro d'esclave par défaut (1) et avec /t0 comme port de communication.

isa -s=3 -t=/t1 Lance une cible ISaGRAF mono-tâche avec le numéro d'esclave 3 et avec /t0 comme port de communication.

isa -t=/t0 &
isa -s=3 -t=/t1 Lance deux cibles ISaGRAF mono-tâches. Une avec le numéro d'esclave par défaut (1) et avec /t0 comme port de communication. L'autre avec le numéro d'esclave 3 et avec /t0 comme port de communication.

C.4.2 Exécution d'ISaGRAF multi-tâches: isaker, isatst, isanet

Afin d'améliorer le temps de réponse du noyau de la cible ISaGRAF et de la liaison de communication, la cible est divisée en deux tâches séparant le travail de communication (tâche de communication istst ou isanet) et l'exécution de l'application (tâche noyau isaker).
Une telle architecture est beaucoup plus flexible. Elle permet à l'utilisateur d'exécuter plusieurs tâches de communication liées à une seule tâche noyau ou d'exécuter jusqu'à 4 noyaux avec la même tâche de communication. Ceci facilite certaines intégrations, comme une liaison de visualisation de procédé et la liaison debugger sur la même application, ou une liaison simple à 4 applications différentes par le même port physique.

Les tâches noyau et communication sont indépendantes et peuvent être lancées séparément. Il est seulement nécessaire de lancer la (les) tâche(s) noyau d'abord, afin qu'elle puisse initialiser son environnement système et que la (les) tâche(s) de communication puisse se connecter.

La cible ISaGRAF multi-tâches n'empêche pas l'exécution des traitements d'arrière-plan ou des routines contrôlées par interruption.

C.4.2.1 Exécution de la tâche noyau: isaker

Numéro d'esclave: option -s

Cette option definit le numéro d'esclave de la cible. Il est compris entre 1 et 255, à l'exception du numéro 13 ($0D). Le numéro d'esclave est utilisé par le protocole de la liaison de communication et sert principalement à distinguer différents esclaves lorsque plusieurs cibles sont exécutées.

Valeur par défaut: Le numéro d'esclave par défaut est 1 (comme dans l'atelier)

C.4.2.2 Exécutuion de la tâche de communication série: isatst

Liaison de communication et configuration: option -t

La tâche de communication isatst utilise une liaison série pour la communication avec le debugger. Le nom du descripteur est défini à l'aide de l'option -t.

Sans valeur par défaut: Si cette option n'est pas utilisée, la communication avec la cible n'est pas possible. Dans ce cas l'erreur n° 7 s'affichera.

Avec l'implémentation de la tâche isatst, la communication utilisant une liaison Ethernet n'est pas disponible.

Le dispositif de liaison série est ouvert en mode de transfert de données binaire (sans caractères de contrôle, sans XON/XOFF). Avant la mise en marche d' ISaGRAF, il est nécessaire de régler d'autres paramètres de communication afin que l'utilisateur puisse utiliser les paramètres nécessaires en toute liberté. Si vous utilisez le debugger de l'atelier, assurez-vous que les paramètres de communication de l'atelier (voir Guide d'utilisateur: Gestion des programmes) correspondent à ceux de la cible.
Exemple:
xmode /t0 baud=19200
Règle la vitesse de communication à 19200 Baud sur le dispositif /t0.

Numéro d'esclave: option -s

Cette option definit le(s) numéro(s) d'esclave de la cible. Il est compris entre 1 et 255, à l'exception du numéro 13 ($0D). Cette option peut être répétée jusqu'à 4 fois pour établir la liaison avec jusqu'à 4 noyaux différents. Le numéro d'esclave est utilisé par le protocole de la liaison de communication et sert principalement à distinguer entre différents esclaves lorsque plusieurs cibles sont exécutées. Si vous utilisez le debugger de l'atelier, assurez-vous que le paramètre de communication de l'atelier (voir Guide d'utilisateur: Gestion des programmes) correspond à une cible existante (tâches noyau et de communication).

Valeur par défaut: Le numéro d'esclave par défaut est 1 (comme dans l'atelier)

Numéro logique des tâches de communication: option -c

Cette option spécifie le numéro logique de la tâche de communication, qui est utilisé pour la gestion simultanée de plusieurs tâches de communication. Il est compris entre 1 et 255 et doit être différent pour chaque tâche de communication.

Valeur par défaut: La dernière option -s spécifiée est utilisée. La valeur par défaut assure la compatibilité avec des versions d'ISaGRAF antérieure (3.0).

C.4.2.3 Exécution de la tâche de communication Ethernet: isanet

Liaison de communication et configuration: option -t

La tâche de communication cible isanet utilise la liaison standard Ethernet pour la communication avec le debugger. Le numéro de port est specifié avec l'option -t Option.

Sans valeur par défaut: Si cette option n'est pas utilisée, la communication avec la cible n'est pas possible. Dans ce cas l'erreur n° 7 s'affichera.

Si vous utilisez le debugger de l'atelier assurez-vous que les paramètres de communication de l'atelier (voir Guide d'utilisateur: Gestion des programmes) correspondent à ceux de la cible.

Pour ISaGRAF, la cible OS-9 est le serveur. Le debugger est le client, qui se connecte au numéro de port spécifié.
Avant de commencer votre première séance de debugging sur Ethernet, assurez-vous que votre dispositif Ethernet OS-9 soit correctement configuré. Vous pouvez par exemple envoyer un "ping" au système OS-9.

Numéro d'esclave: option -s

Cette option définit le(s) numéro(s) d'esclave de la cible auxquels la tâche de communication est liée. Ils sont compris entre 1 et 255, à l'exception du numéro 13 ($0D). Cette option peut être répétée pour établir la liaison avec jusqu'à 4 esclaves noyaux différents. Le numéro d'esclave est utilisé par le protocole de la liaison de communication et sert principalement à distinguer différents esclaves lorsque plusieurs cibles sont exécutées. Si vous utilisez le debugger de l'atelier, assurez-vous que le paramètre de communication de l'atelier (voir Guide d'utilisateur: Gestion des programmes) correspond à une cible existante (tâches noyau et de communication).

Valeur par défaut: Le numéro d'esclave par défaut est 1 (comme dans l'atelier)

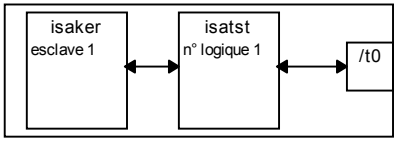
Numéro logique des tâches de communication: option -c

Cette option spécifie le numéro logique de la tâche de communication, qui est utilisé pour la gestion simultanée de plusieurs tâches de communication. Il est compris entre 1 et 255 et doit être différent pour chaque tâche de communication.

Valeur par défaut: La dernière option -s spécifiée est utilisée. La valeur par défaut assure la compatibilité avec des versions d'ISaGRAF antérieure (3.0).

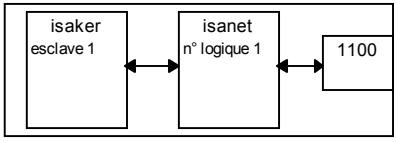
C.4.2.4 Exemples:

isaker &
isatst -t=/t0



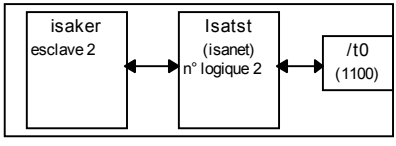
Lance:
Une tâche noyau ISaGRAF avec le numéro d'esclave par défaut (1).
Une tâche de communication série ISaGRAF sur le port /t0, liée au numéro d'esclave par défaut (1) et avec le numéro logique par défaut (dernier numéro d'esclave spécifié = défaut = 1).

isaker &
isanet -t=1100



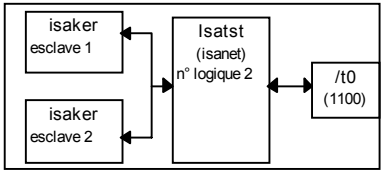
Lance:
Une tâche noyau ISaGRAF avec le numéro d'esclave par défaut (1).
Une tâche de communication Ethernet ISaGRAF sur le port numéro 1100, liée au numéro d'esclave par défaut (1) et avec le numéro logique par défaut (dernier numéro d'esclave spécifié = défaut = 1).

isaker -s=2 &
isatst -t=/t0 -s=2 (respectivement isanet -t=1100 -s=2)



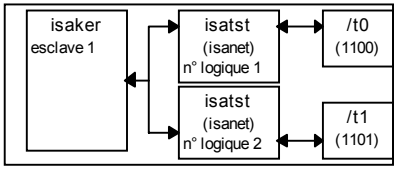
Lance:
Une tâche noyau ISaGRAF avec le numéro d'esclave 2.
Une tâche de communication série (Ethernet) ISaGRAF sur le port /t0 (port numéro 1100) liée au numéro d'esclave 2 et avec le numéro logique par défaut (dernier numéro d'esclave spécifié = 2).

Isaker -s=1 &
isaker -s=2 &
isatst -t=/t0 -s=1 -s=2 (respectivement isanet -t=1100 -s=1 -s=2)



Lance:
Une tâche noyau ISaGRAF avec le numéro d'esclave 1.
Une tâche noyau ISaGRAF avec le numéro d'esclave 2.
Une tâche de communication série (Ethernet) ISaGRAF sur le port /t0 (port numéro 1100) liée au numéros d'esclave 1 et 2 et avec le numéro logique par défaut (dernier numéro d'esclave spécifié = 2).

Isaker -s=1 &
isatst -t=/t0 -s=1 -c=1 & (respectivement isanet -t=1100 -s=1 -c=1 &)
isatst -t=/t1 -s=1 -c=2 (respectivement isanet -t=1101 -s=1 -c=2)



Lance:
Une tâche noyau ISaGRAF avec le numéro d'esclave 1.
Une tâche de communication série (Ethernet) ISaGRAF sur le port /t0 (port numéro 1100) liée au numéro d'esclave 1 et avec le numéro logique 1.
Une tâche de communication série (Ethernet) ISaGRAF sur le port /t0 (port numéro 1100) liée au numéro d'esclave 1 et avec le numéro logique 2.
Note:
Les tâches de communication séries et Ethernet peuvent être mélangées.

C.4.3 Fonctionalités spécifiques

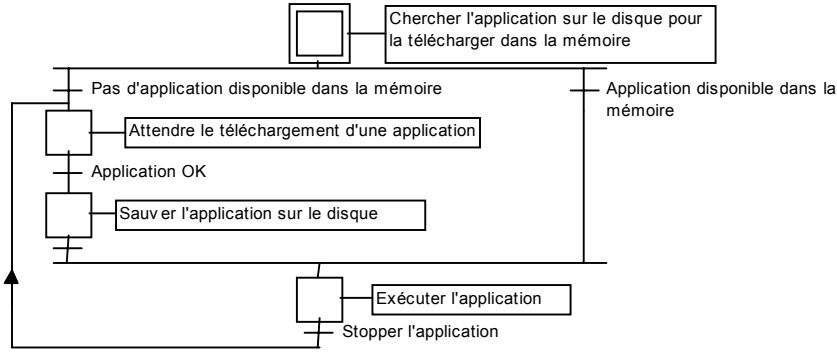
Liaison de communication

Le Gestionnaire sériel de caractères OS-9 est extèmement flexible. La plupart des dipositifs physiques bidirectionnels supportés par Microware peuvent être utilisés:

Exemple:
La liaison serielle peut être un chemin de réseau menant à un port physique qui se trouve sur une autre unité centrale.
Alors l'option -t peut par exemple être utilisée comme suite: -t=/nr/MASTER/t0
Ici, la liaison de communication est déportée sur une unité centrale appelée MASTER dans un réseau ramnet. Le port physique utilisé est /t0.

Lancement de la cible ISaGRAF

Au lancement de la cible l'algorithme suivant est exécuté.



- **Définitions**

Le code de l'application est la base de données binaire générée et téléchargée par l'atelier et ensuite exécutée par la cible. Il peut être complété par la table des symboles.
La table des symboles d'une application est une base de données ASCII générée et téléchargée par l'atelier. Cette table réalise la liaison entre les objets symboliques et les objets internes de la cible. Elle n'est pas indispensable dans la cible, sauf pour la gestion des symboles spécifiques de l'utilisateur. Pour plus d'informations sur la table des symboles voir le Guide d'Utilisateur: Techniques de programmation avancées.

- **Objets ISaGRAF OS-9 et l'application multiple**

Tous les noms d'objets publics ISaGRAF commencent avec '**ISAxn**', où **x** est le numéro d'esclave noyau et **n** est un numéro d'espace ayant une signification spécifique, à l'exception de **ISAy3**, où **y** est le numéro logique d'une tâche de communication dans une implémentation multi-tâche.
Sur une même unité centrale, des applications différentes (tâches noyau et de communication) peuvent être exécutées en même temps, à condition de posséder des numéros d'esclave différents et des numéros logiques de tâche de communication différents, respectivement. Lorsque des applications différentes sont exécutées, l'utilisateur doit néanmoins faire attention à l'accès partagé de certains objets de l'application, tel que les cartes E/S. Il se peut que différentes applications (noyaux) utilisent des cartes physiques distinctes, sauf si un serveur E/S ou sémaphore est implémenté à travers le driver E/S.

Noms d'objets OS-9:
Fichiers disque:

ISAx1	Fichier de sauvegarde du code d'application ISaGRAF
ISAx6	Fichier de sauvegarde des symboles de l'application ISaGRAF

Modules mémoire:

ISAx0	Données système du noyau ISaGRAF
ISAx1	Code d'application ISaGRAF
ISAx2	Base de données temps réel du noyau ISaGRAF
ISAy3	Buffer d'échange des données de communication ISaGRAF
ISAx4	Modifications en-ligne 1 du code d'application ISaGRAF
ISAx5	Modifications en-ligne 2 du code d'application ISaGRAF
ISAx6	Symboles d'application ISaGRAF

Pour cette raison l'utilisateur doit faire attention à ne pas utiliser les mêmes noms d'objets.

- **Sauvegarde de l'application**

Quand une nouvelle application est téléchargée par le debugger de l'atelier dans la cible, le code d'application est sauvegardé dans le répertoire courant sous le nom:
ISAx1 Fichier de sauvegarde du code d'application ISaGRAF (x est le numéro d'esclave)

Si la table des symboles de l'application est téléchargée auparavant, elle est aussi sauvegardée dans le répertoire courant de la cible sous le nom:
ISAx6 Fichier de sauvegarde des symboles de l'application ISaGRAF (x est le numéro d'esclave)

Lorsque la cible ISaGRAF est lancée, ces fichiers de code et de symboles sont recherchés dans le répertoire courant et téléchargés sous les mêmes noms comme modules de données dans la mémoire.

Ensuite, si le fichier des symboles n'est pas disponible, la cible commence l'exécution du code d'application sans les symboles.
Si aucun code d'application n'est disponible en mémoire, la cible attend le téléchargement d'une application.

Afin de démarrer la cible avec une application spécifique au moment de la mise sous tension, sans utiliser le debugger, il y a deux possibilités:

- Copier les fichiers à partir du PC Host, sur lequel l'atelier est installé, à l'aide d'un outil de transfert de données directement sur le disque répertoire courant de la cible. Vous pouvez utiliser le menu d'outils de l'atelier (voir Guide d'utilisateur: Gestion des programmes) pour simplifier ces manipulations.

- Stocker le code d'application (et la table des symboles, si nécessaire) à partir de fichiers sur le PC Host, sur lequel l'atelier est installé, dans une mémoire non-volatile (comme PROM ou EPROM) en utilisant vos propres outils.

  Si vous souhaitez un accès ou une gestion des points d'arrêt plus rapide, par exemple, vous pouvez ensuite, au moment de la mise sous tension du système, télécharger le code d'application (et éventuellement la table des symboles de l'application) comme module memoire **ISAx1** (et **ISAx6**) du PROM au RAM en utilisant vos propres outils.

  ATTENTION:
  La gestion des points d'arrêts du debugger ISaGRAF ne peut être exécutée correctement, si le module de code d'application n'est pas accessible à l'écriture. Ceci ne pose pas de problème, car normalement votre application a été testée complètement auparavant.

Sur le PC Host, si l'atelier ISaGRAF est installé dans le répertoire standard \ISAWIN:
le fichier de code d'application du projet MYPROJ est:
\ISAWIN\APL\MYPROJ\appli.x6m (correspond à isax1 sur la cible).
le fichier de symboles de l'application du projet MYPROJ est:
\ISAWIN\APL\MYPROJ\appli.tst (correspond à isax6 sur la cible).

Gestion des erreurs et messages de sortie

Le logiciel de la cible ISaGRAF permet la gestion de la détection d'erreurs. En annexe vous trouverez la liste des messages d'erreurs avec leur description.

La détection d'erreurs est traitée de la manière suivante:
- Une erreur se compose d'une erreur et d'un numéro d'argument qui sont envoyés à la routine d'erreurs d' ISaGRAF.
- Si la détection d'erreurs a été validée dans les paramètres d'exécution de l'atelier, l'erreur sera traitée. Sinon les informations seront perdues et la détection d'erreurs s'arrêtera là.

Si les erreurs sont traitées:
- Le numéro d'erreur (valeur décimale) et l'argument (valeur hexadécimale) s'afficheront dans la sortie de défaut stdout.
- Le numéro d'erreur et l'argument seront envoyés dans un buffer de type FIFO, pour pouvoir être récupérés plus tard. La taille du buffer est définie dans les options "peremètres d'exécution" de l'atelier. Lorsque le buffer est plein, à chaque arrivé d'une nouvelle erreur l'erreur la plus ancienne sera perdue.
- Les erreurs peuvent être appelées soit par le debugger soit par l'application courante en utilisant l'appel SYSTEM (voir Guide d'utilisateur).

Si le debugger a détecté une erreur, un message avec la description de l'erreur s'affiche dans la fenêtre d'erreurs. Selon le contexte de l'application (en exécution ou non) le debugger peut afficher le nom de l'objet (variable ou programme) dans lequel se trouve l'erreur, ou bien l'erreur d'argument (valeur décimale) entre parenthèses [x] ce qui peut avoir une signification différente pour chaque erreur.

Un message de bienvenue et les valeurs d'erreur sont affichés dans la sortie stdout, lorsque la cible démarre et si des erreurs ont été détectées. Si cet affichage n'est pas souhaité sur le canal de sortie standard, une commande de redirection peut être utilisée, comme:

prog_name [options] **>>>/nil**

Durée de cycle, comportement et priorités des tâches

- A la fin d'un cycle ISaGRAF, juste avant le début d'un nouveau cycle, l'algorithme suivant est exécuté:
  Si un temps de cycle est spécifié (à partir de l'atelier: voir Guide d'utilisateur: Gestion des programmes) l'unité centrale est abandonnée pour la période de temps restante (temps de cycle spécifié - temps de cycle de l'application courante). Si la période de temps est négative, un dépassement est généré et l'unité centrale est abandonnée pour un tic afin de forcer le scheduling.

  Si le temps de cycle n'est pas spécifié ou si le temps restant est inférieur ou égal à 1 tic ou égal à zéro, l'unité centrale est abandonnée pour 1 tic pour forcer le scheduling.

  La précision temporelle de la cible correspond à un tic standard du système OS-9.

  En général, la spécification du temps de cycle sert à déclencher les cycles ou à mettre l'unité centrale à la disposition des autres tâches exécutées sur le système OS-9.

- Tant qu'il n'y a pas de données entrant par la liaison de communication la tâche de communication se trouve en état de "sommeil". Si nécessaire, cette tâche obtient des informations sur l'application courante à l'aide d'un protocole question/réponse avec la tâche noyau. La tâche de communication envoie une question au noyau. A la fin du cycle (pour obtenir une image synchrone de l'application), le noyau donne la réponse à la tâche de communication.

Les tâches ISaGRAF ne modifient pas les priorités qui leur ont été données. L'utilisateur peut librement ajuster ces priorités selon le comportement des tâches ISaGRAF décrites ci-dessus et suivant les besoins globaux de l'application.
Afin de s'assurer qu'ISaGRAF n'est pas occupé par une tâche de priorité inférieure, on peut par exemple modifier les paramètres de gestion des tâches OS-9, tels que **MIN_AGE** et **MAX_AGE**.

Mode terminal

Le protocole de communication série de la cible reconnait une séquence de 3 caractères retours charriot ($0D) et ensuite lance une tâche OS-9 shell, si disponible, sur le dispositif de liaison série.
Ceci permet d'obtenir une invite OS-9 shell sur tout terminal en utilisant la liaison série de la cible ISaGRAF.

Exemple:
A partir du PC host:
- Fermer le debugger ISaGRAF.
- Commencer une séance terminal Windows (groupe accessoires) avec les paramètres de communication corrects.
- Appuyer 3 fois sur la touche retour charriot.

Vous êtes maintenant connecté à un shell OS-9:
- Entrer **logout** pour quitter le mode terminal.

ATTENTION:On doit toujours quitter une séance en mode terminal d'une manière propre, en utilisant logout et rien d'autre, sinon la prochaine connection avec l'atelier sera sans succès.

C.5 Mise en route de la cible VxWorks

Pour l'exécution de la (des) cible(s) ISaGRAF on doit d'abord exécuter quelques commandes sur le système VxWorks afin de définir l'environnement de la configuration et enfin lancer la (les) cible(s) ISaGRAF. Toutes ces commandes peuvent être lancées à partir d'un fichier Script. Elle seront décrites dans les chapitres suivants.

C.5.1 Le gestionnaire des ressources systèmes: isassr.o

Ce module est nécessaire pour toutes les configurations de la cible ISaGRAF et doit être téléchargé en premier. Il permet la gestion des ressources systèmes de plusieurs cibles exécutées en même temps.

C.5.2 Fonctionnalités communes de isa.o, isakerse.o et isakeret.o

Pour l'exécution d'ISaGRAF il est possible de télécharger un des modules suivants .

isa.o: permet le démarrage de cibles ISaGRAF mono-tâches (liaison de communication série uniquement).

Isakerse.o: permet le démarrage de cibles ISaGRAF multi-tâches (liaison de communication série uniquement).

Isakeret.o: permet le démarrage de cibles ISaGRAF multi-tâches (liaison de communication série et/ou Ethernet).

Ces modules sont décrits en détails dans les chapitres suivants.

Configuration de la liaison de communication série

La cible ISaGRAF utilise une liaison série pour la communication avec le debugger. Lorsque cette liaison est ouverte à partir de la cible ISaGRAF, la configuration n'est pas automatique mais peut être définie librement par l'utilisateur. Ceci nécessite un mode de transfert de données binaires (RAW Mode). Pour ceci la sous-routine *ISAMOD ()* est disponible:

```
uchar ISAMOD
  (
  char *desc,        /* nom de l'unité série */
  uint32 baudrate    /* débit (baud)        */
  )
```

Description:
Configure l'unité de liaison série spécifiée pour un transfert de données binaire avec le débit spécifié.

Valeur de retour:
0 si réussi, BAD_RET en cas d'erreurs.

Si vous utilisez le debugger de l'atelier, assurez-vous que les paramètres de communication de l'atelier (voir Guide d'utilisateur: Gestion des programmes) correspondent à ceux de la cible.

Taux d'horloge système

La variable globale CLKRATE (uint32) doit être initialisée en fonction de l'horloge du système VxWorks. Vous pouvez utiliser:

```
CLKRATE = sysClkRateGet ()
```

La valeur par défaut de CLKRATE est 60Hz.

C.5.3 Exécution d'ISaGRAF mono-tâche: isa.o

Il est possible d'exécuter la cible ISaGRAF en une seule tâche, mais les opérations dans une telle configuration peuvent être critiques. Par exemple, il est recommandé de ne pas surcharger la liaison de communication pour garantir une bonne performance. Le système VxWorks multi-tâche permet d'exécuter plusieurs cibles ISaGRAF mono-tâches sur une seule unité centrale à condition qu'elles possèdent différents numéros d'esclave et ports de communication.
L'implémentation mono-tâche a été conçue principalement pour certaines plateformes matérielles, comme les cartes ou PCs MS-DOS, ou pour créer un premier prototype afin de le porter sur une nouvelle plateforme. Pour cette raison, on devrait préférer la solution de l'implémentation multi-tâche de la cible ISaGRAF.
La cible ISaGRAF mono-tâche n'empêche pas l'exécution des traitements d'arrière-plan ou des routines contrôlées par interruption.

Enregistrement des esclaves

Une cible ISaGRAF est caractérisée par son numéro d'esclave. Il est compris entre 1 et 255, à l'exception du numéro 13 ($0D). Le numéro d'esclave est utilisé par le protocole de la liaison de communication et sert à distinguer entre différents esclaves lorsque plusieurs cibles sont exécutées en même temps. Ainsi, avant de lancer la ou les tâche(s) de la cible ISaGRAF, elle(s) doit être enregistrée(s). Pour cette raison la sous-routine *isa_register_slave()* est disponible.

```
uchar isa_register_slave
  (
  uchar slave    /* numéro d'esclave */
  )
```

Description:
Enregistre un nouvel esclave dans le système de gestion des cibles multiples.

Valeur de retour:
0 si réussi, BAD_RET en cas d'erreurs.

Unité de stockage pour la sauvegarde du fichier d'application

La variable globale TSK_FUNIT (char *) peut être initialisée par une chaîne de caractères contenant le chemin de l'unité de stockage pour la sauvegarde du fichier d'application. La cible ISaGRAF utilise simplement les routines de gestion de fichiers standards fopen, fread, fwrite et fclose pour la sauvegarde du fichier d'application.
La valeur par défaut est une chaîne vide (""), ce qui signifie qu'il n'y a pas d'unité de stockage.

Exemple:
TSK_FUNIT = "*host name:*/C:/ISaGRAF/target/apl/"

Spécifie ISaGRAF\target\apl\, sur la racine de l'unité C:, sur le PC *host_name*, comme répertoire de sauvegarde du fichier d'application. N'oubliez-pas le dernier slash, sinon la sauvegarde sera effectuée sur ISaGRAF\target\ avec des noms de fichiers préfixés par apl.

Si nécessaire, cette variable peut être réglée à différentes unités pour chaque cible avant chaque lancement.
Vous trouverez des informations supplémentaires sur la sauvegarde des fichiers d'application dans "Fonctionnalités spécifiques", au chapitre Sauvegarde de l'application.

Contrôle de fin de cycle

La variable TSK_NBTCKSCHED (uint 32) peut être réglée à une valeur spécifiant un délai de tic et utilisée à la fin du cycle par la cible ISaGRAF.
La valeur par défaut est 0 (même priorité de tâche).
Si nécessaire, cette variable peut être réglée à une valeur différente pour chaque cible avant chaque lancement.
Vous trouverez des informations supplémentaires dans "Fonctionnalités spécifiques", dans le chapitre "Durée de cycle, comportement et priorités des tâches".

Lancement de la cible ISaGRAF

Une fois l'environnement de la configuration défini, la ou les cibles ISaGRAFsont lancées: isa_main.

```
uchar isa_main
  (
  uchar slave,      /* numéro d'esclave       */
  char *com         /* nom de l'unité série */
  )
```

Description:
Lance une tâche de la cible ISaGRAF.

Valeur de retour:
Retourne une valeur différente à zéro en cas d'erreur.

Pour le numéro d'esclave voir chapitre "Enregistrement des esclaves".
Il est possible de lancer plusieurs cibles à condition que leurs numéros d'esclave et leurs ports de communication soient différents.
Si vous utilisez le debugger de l'atelier, assurez-vous que le paramètre de communication de l'atelier (voir Guide d'utilisateur: Gestion des programmes) correspond à une cible existante.

Exemple

Cet exemple montre comment lancer une cible mono-tâche ISaGRAF avec le numéro d'esclave 1 et avec le dispositif /tyCo/1 pour la liaison série.
Le répertoire hôte courant est celui dans lequel est installé la cible.

Télécharger le module isassr.o
ld < RELS/isassr.o

Télécharger le module isa.o
ld < CMDS/isa.o

Configuration de la communication série
ISAMOD ("/tyCo/1", 19200)

Taux d'horloge du système
CLKRATE = sysClkRateGet ()

Enregistrement de l'esclave
isa_register_slave (1)

Unité de stockage de fichier (peut être sauté à cause de la valeur par défaut)
TSK_FUNIT = ""

Contrôle de fin de cycle (peut être sauté à cause de la valeur par défaut)
TSK_NBTCKSCHED = 0

Lancement de la cible ISaGRAF
sp (isa_main, 1, "/tyCo/1")

C.5.4 Exécution d'ISaGRAF multi-tâches: isakerse.o und isakeret.o

Afin d'améliorer le temps de réponse du noyau ISaGRAF et de la liaison de communication, la cible est divisée en deux tâches séparant le travail de communication (tâche de communication) et l'exécution de l'application (tâche du noyau).
Une telle architecture est beaucoup plus flexible. Elle permet à l'utilisateur d'exécuter plusieurs tâches de communication liées à une seule tâche du noyau ou d'exécuter jusqu'à 4 noyaux avec la même tâche de communication. Ceci facilite certaines intégrations, comme une liaison de visualisation de procédé et la liaison debugger sur la même application ou une liaison simple à 4 applications différentes par le même port physique.

Les tâches noyau et communication sont indépendantes et peuvent être lancées séparément. Il est seulement nécessaire de lancer la (les) tâche(s) noyau d'abord, afin qu'elle puisse initialiser son environnement système et que la (les) tâche(s) de communication puisse se connecter.

La cible ISaGRAF multi-tâche n'empêche pas l'exécution des traitements d'arrière-plan ou des routines contrôlées par interruption.

Deux modules sont disponibles suivant les capacités de communication du matériel:
- Noyau et liaison série: isakerse.o
Ce module permet de lancer la (les) tâch(s) noyau et la (les) tâche(s) de communication série.
- Noyau et liaison série et/ou Ethernet: isakeret.o
Ce module permet de lancer la (les) tâche(s) noyau et la (les) tâche(s) de communication série et/ou Ethernet.

Le lancement d'ISaGRAF est le même pour les deux modules (isakerse.o und isakeret.o) sauf que pour isakeret.o il faut spécifier, soit le nom d'un port série, soit le numéro de port Ethernet, comme paramètre pour le dispositif de communication lorsque la tâche de communication ISaGRAF tst_main_ex est lancée (voir ci-dessous).

Pour ISaGRAF, la cible VxWorks est le serveur. Le debugger est le client qui se connecte au numéro de port spécifié.

Eregistrement d'un ou de plusieurs noyaux

Un noyau ISaGRAF est caractérisé par son numéro d'esclave. Il est compris entre 1 et 255, à l'exception du numéro 13 ($0D). Le numéro d'esclave est utilisé par le protocole de la liaison de communication et par la (les) tâche(s) de communication liée(s) au noyau. Il sert à distinguer différents esclaves lorsque plusieurs cibles sont exécutées en même temps. Ainsi, avant de lancer la ou les tâches noyau de la cible ISaGRAF, elles doivent être enregistrées. Pour cette raison la sous-routine *isa_register_slave()* est disponible.

```
uchar isa_register_slave
  (
  uchar slave    /* numéro d'esclave */
  )
```

Description:
Enregistre un nouvel esclave dans le système de gestion de cibles multiples.

Valeur de retour:
0 si réussi, BAD_RET en cas d'erreurs.

Enregistrement d'une ou de plusieurs tâches de communication

Une tâche de communication ISaGRAF se caractérise par son numéro logique, qui est utilisé pour la gestion de plusieurs tâches de communication en même temps. Ce numéro se situe entre 1 et 255 et doit être différent pour chaque tâche de communication. Avant de lancer une ou plusieurs tâches de communication ISaGRAF il faut les enregister. Pour ceci on utilisera la sous-routine *isa_register_com()*.

```
uchar isa_register_com
  (
  uchar com_id      /* identificateur de la tâche de communication */
  )
```

Description:
Enregistre une nouvelle tâche de communication dans le système de gestion de cibles multiples.

Valeur de retour:
0 si réussi, BAD_RET en cas d'erreurs.

Unité de stockage pour la sauvegarde du fichier d'application

La variable globale TSK_FUNIT (char *) peut être initialisée par une chaîne de caractères contenant le chemin de l'unité de stockage pour la sauvegarde du fichier d'application. La cible ISaGRAF utilise simplement les routines de gestion de fichiers standards fopen, fread, fwrite et fclose pour la sauvegarde du fichier d'application.
La valeur par défaut est une chaîne vide (""), ce qui signifie qui n'y a pas d'unité de stockage.

Exemple:

TSK_FUNIT = "*host name:*/C:/ISaGRAF/target/apl/"
Spécifie ISaGRAF\target\apl\, sur la racine de C:, sur le PC *host_name*, comme répertoire de sauvegarde du fichier d'application. N'oubliez-pas le dernier slash, sinon la sauvegarde sera effectuée sur ISaGRAF\target\ avec des noms de fichiers préfixés par apl.

Si nécessaire, cette variable peut être réglée à différentes unités pour chaque cible avant chaque lancement.
Vous trouverez des informations supplémentaires sur la sauvegarde des fichiers d'application dans "Fonctionnalités spécifiques", dans le chapitre "Sauvegarde de l'application".

Contrôle de fin de cycle

La variable TSK_NBTCKSCHED (uint 32) peut être réglée à une valeur spécifiant un délai de tic et utilisée à la fin du cycle par la cible ISaGRAF.
La valeur par défaut est 0 (même priorité de tâche).
Si nécessaire, cette variable peut être réglée à une valeur différente pour chaque cible avant chaque lancement.
Vous trouverez des informations supplémentaires dans "Fonctionnalités spécifiques", dans le chapitre "Durée de cycle, comportement et priorités des tâches".

Lancement du noyau ISaGRAF

Une fois que l'environnement de la configuration est réglé on lance le ou les noyaux ISaGRAF: isa_main.

```
uchar isa_main
  (
  uchar slave,      /* Numéro d'esclave       */
  char *com         /* NOT USED Chaîne vide est OK */
  )
```

Description:
Lance une tâche noyau ISaGRAF.

Valeur de retour:
En cas d'erreur la valeur de retour est différente de zéro.

Pour le numéro d'esclave voir chapitre "Enregistrement d'esclaves".
Il est possible de lancer plusieurs noyaux à condition qu'ils aient des numéros d'esclaves différents.

Lancement de la tâche de communication ISaGRAF

Une fois que l'environnement de la configuration est réglé on lance la ou les tâches de communication ISaGRAF: tst_main_ex.

```
uchar tst_main_ex
  (
  char *com,        /* nom du dispositif de communication */
  uchar *slave,     /* localisation d'un champs de 4 Octets spécifiant le ou les esclaves
  noyau pour la liaison */
  uchar com_id      /* identificateur de la tâche de communication */
  )
```

Description:

Lance une tâche de communication ISaGRAF.

Valeur de retour:
En cas d'erreur la valeur de retour est différente de zéro.

Le champ de 4 Octets spécifie le ou les esclaves noyaux auxquels la tâche de communication est liée. Si moins de 4 esclaves noyaux sont nécessaires, le champ doit être complété par des zéros. Une fois la tâche lancée, ce champ n'est plus nécessaire.
Le nom du dispositif de communication correspond au nom du dispositif sériel à utiliser pour la liaison de communication.
Il est possible de lancer plusieurs tâches de communications à condition que leurs identificateurs de tâche soient différents.
Si vous utilisez le debugger de l'atelier, assurez-vous que les paramètres de communication de l'atelier (voir Guide d'utilisateur: Gestion des programmes) correspondent à une cible existante (tâches noyau et de communication).

Exemple:

Cet exemple montre comment lancer:
Une tâche noyau ISaGRAF avec le numéro d'esclave 1.
Une tâche de communication ISaGRAF identifiée par le numéro 1, liée à l'esclave noyau 1 et avec le dispositif /tyCo/1 pour la liaison série.
Une tâche de communication ISaGRAF identifiée par le numéro 2, liée à l'esclave noyau 1 et avec le numéro de port 1100 pour la liaison de communication Ethernet.
Le répertoire hôte courant est celui dans lequel est installé la cible.

Télécharger le module isassr.o
ld < RELS/isassr.o

Télécharger le module isakeret.o (Vous pouvez aussi télécharger isakerse.o, si vous n'avez pas besoin d'une liaison de communication Ethernet)
ld < CMDS/isakeret.o

Configuration de la communication série
ISAMOD ("/tyCo/1", 19200)

Taux d'horloge du système
CLKRATE = sysClkRateGet ()

Enregistrement de l'esclave
isa_register_slave (1)

Enregistrement de la communication
isa_register_com (1)
isa_register_com (2)

Unité de stockage de fichier (peut être sauté à cause de la valeur par défaut)
TSK_FUNIT = ""

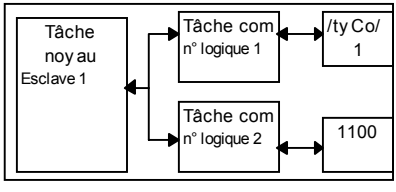
Contrôle de fin de cycle (peut être sauté à cause de la valeur par défaut)
TSK_NBTCKSCHED = 0

Lancement du noyau ISaGRAF
sp (isa_main, 1, "")

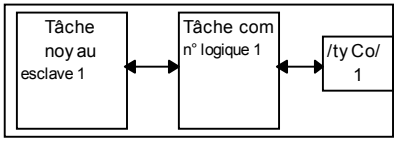
Tâche de communication, liaison des esclaves
SlavesLink = 0x01000000

Lancement des tâches de communication ISaGRAF
sp (tst_main_ex, "/tyCo/1", &SlavesLink, 1)
sp (tst_main_ex, "1100", &SlavesLink, 2)

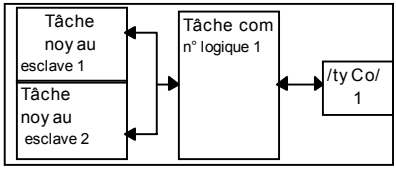
Ce lancement correspond au schéma suivant:



En plus, vous avez le choix entre les configuration de base suivantes:



La configuration la plus basique est une tâche noyau associée à une tâche de communication sur une liaison série (Ethernet).

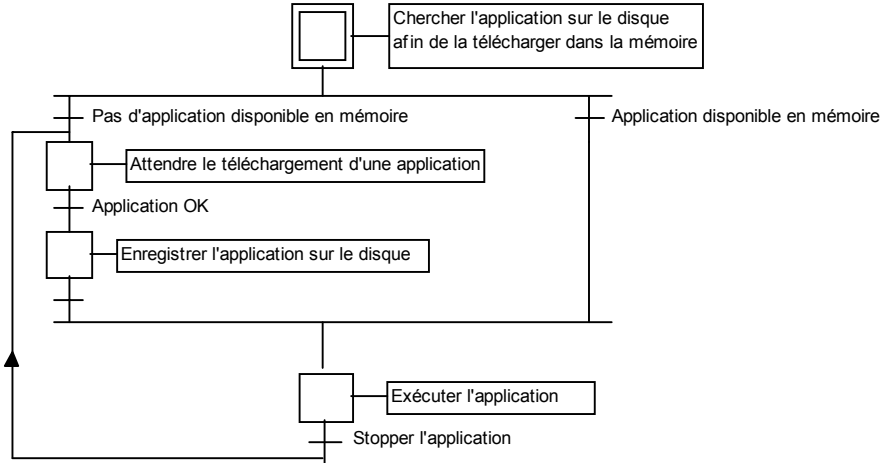


Une autre configuration comprend 2 noyaux associés à une tâche de communication sur une liaison série (Ethernet). Dans ce cas, SlavesLink = 0x01020000.

C.5.5 Fonctionnalités spécifiques

Lancement d'ISaGRAF

Après le lancement de la cible l'algorithme suivant est exécuté.



- **Définitions**

Le code de l'application est la base de données binaire générée et téléchargée par l'atelier et ensuite exécutée par la cible. Il peut être complétée par la table des symboles.
La table des symboles d'une application est une base de données ASCII générée et téléchargée par l'atelier. Cette table réalise la liaison entre les objets symboliques et les objets internes de la cible. Elle n'est pas indispensable dans la cible, sauf pour la gestion des symboles spécifiques de l'utilisateur. Pour plus d'informations sur la table des symboles voir le Guide d'Utilisateur: Techniques de programmation avancées.
Le chemin de l'unité de fichier disque est spécifié au lancement de la cible ISaGRAF en utilisant la variable globale TSK_FUNIT (valeur par défaut = "" signifie, qu'il n'y a pas d'unité de fichier disque).

- **Applications multiples ISaGRAF**

Sur une même unité centrale, des applications différentes (tâches noyau et de communication) peuvent être exécutées en même temps, à condition de posséder des numéros d'esclave différents et des numéros logiques de tâche de communication différents, respectivement. Lorsque des applications différentes sont exécutées, l'utilisateur doit néanmoins faire attention à l'accès partagé de certains objets de l'application, tel que les cartes E/S. Il se peut que différentes applications (noyaux) utilisent des cartes physiques distinctes, sauf si un serveur E/S ou sémaphore est implémenté à travers le driver E/S.

- **Sauvegarde de l'application**

Quand une nouvelle application est téléchargée par le debugger de l'atelier dans la cible le code d'application (la cible utilise les routines standards de gestion de fichiers fopen,...) est sauvegardé sous le nom suivant:

*path***ISAx1** Fichier de sauvegarde du code d'application ISaGRAF (x est le numéro d'esclave)

Si la table des symboles de l'application est téléchargée auparavant, elle est aussi sauvegardée dans le répertoire courant de la cible sous le nom:

*path***ISAx6** Fichier de sauvegarde des symboles de l'application ISaGRAF (x est le numéro d'esclave)

Le chemin *path* est spécifié au lancement de la cible ISaGRAF en utilisant la variable globale TSK_FUNIT. Une chaîne vide ("") signifie, qu'il n'y a pas d'unité de fichier disque (valeur par défaut).

Lorsque la cible ISaGRAF est lancée, ces fichiers de code et de symboles sont recherchés dans le répertoire courant et téléchargés dans la mémoire.

Ensuite, si le fichier des symboles n'est pas disponible en mémoire, la cible commence l'exécution du code de l'application sans les symboles.
Si aucun code d'application n'est diponible en mémoire, la cible attend le téléchargement d'une application.

Afin de démarrer la cible avec une application spécifique au moment de la mise sous tension, sans utiliser le debugger, il y a deux possibilités:

- Copier les fichiers à partir du PC Host, sur lequel l'atelier est installé, à l'aide d'un outil de transfert de données dans l'unité de stockage de la sauvegarde de l'application. Vous pouvez utiliser le menu d'outils de l'atelier (voir Guide d'utilisateur: Gestion des programmes) pour simplifier ces manipulations.

- Stocker le code de l'application (et la table des symboles, si nécessaire) à partir de fichiers sur le PC Host, sur lequel l'atelier est installé, dans une mémoire non-volatile (comme PROM ou EPROM) en utilisant vos propres outils.

  Si vous souhaitez un accès ou une gestion des points d'arrêt plus rapide, par exemple, vous pouvez ensuite, au moment de la mise sous tension du système, télécharger le code de l'application (et éventuellement la table des symboles de l'application) du PROM au RAM en utilisant vos propres outils.

  Ensuite au lancement d'ISaGRAF (juste avant le lancement des tâches) vous devez spécifier la (les) adresse(s), où est stocké le code de l'application (et si nécessaire, la table des symboles de l'application). Pour ceci vous devez initialiser la variable globale SSR comme suit:
  SSR[x][1].space = *Adresse du code d'application*
  et si nécessaire:
  SSR[x][6].space = *Adresse de la table des symboles de l'application*

  Pour faire ceci vous pouvez écrire une courte procédure. La Variable globale SSR est déclarée comme un type de structure str_ssr qui est défini dans le fichier tasy0ssr.h.

  ATTENTION:
  La gestion des points d'arrêts du debugger ISaGRAF ne peut être exécutée correctement, si le module de code de l'application n'est pas accessible à l'écriture. Ceci ne pose pas de problème, car normalement votre application a été testée complètement auparavant.

Sur le PC Host, si l'atelier ISaGRAF est installé dans le répertoire standard \ISAWIN:
le fichier de code d'application du projet MYPROJ est:
\ISAWIN\APL\MYPROJ\appli.x6m (correspond à isax1 sur la cible).
le fichier de symboles de l'application du projet MYPROJ est:

\ISAWIN\APL\MYPROJ\appli.tst (correspond à isax6 sur la cible).

Gestion des erreurs et messages de sortie

Le logiciel de la cible ISaGRAF permet la gestion de la détection d'erreurs. En annexe vous trouverez la liste des messages d'erreurs avec leur description.

La détection d'erreurs est traitée de la manière suivante:

- Une erreur se compose d'une erreur et d'un numéro d'argument qui sont envoyés à la routine d'erreurs d' ISaGRAF.
- Si la détection d'erreurs a été validée dans les paramètres d'exécution de l'atelier, l'erreur sera traitée. Sinon les informations seront perdues et la détection d'erreurs s'arrêtera là.

Si les erreurs sont traités:

- Le numéro d'erreur (valeur décimale) et l'argument (valeur hexadécimale) s'afficheront dans la sortie de défaut stdout.
- Le numéro d'erreur et l'argument seront envoyés dans un buffer de type FIFO, pour pouvoir être récupérés plus tard. La taille du buffer est définie dans les options "peremètres d'exécution" de l'atelier. Lorsque le buffer est plein, à chaque arrivé d'une nouvelle erreur l'erreur la plus ancienne sera perdue.
- Les erreurs peuvent être appelées soit par le debugger soit par l'application courante en utilisant l'appel SYSTEM (voir Guide d'utilisateur).

Si le debugger a détecté une erreur, un message avec la description de l'erreur s'affiche dans la fenêtre d'erreur. Selon le context de l'application (en exécution ou non) le debugger peut afficher le nom de l'objet (variable ou programme) dans lequel se trouve l'erreur, ou bien l'erreur d'argument (valeur décimale) entre parenthèses [x] ce qui peut avoir une signification différente pour chaque erreur.

Sur la cible, les valeurs des erreurs détectées sont affichées dans la sortie par défaut stdout. L'affichage peut être dirigée en utilisant des routines VxWorks, tel que:

ioGlobalStdSet()

ou *ioTaskStdSet()*

Dans le dernier cas afin que ni le noyau ni les tâches de communication peuvent générer des erreurs.

Durée de cycle, comportement et priorités des tâches

- A la fin d'un cycle ISaGRAF, juste avant le début d'un nouveau cycle, l'algorithme suivant est exécuté:

  Si un temps de cycle est spécifié (à partir de l'atelier: voir Guide d'utilisateur: Gestion des programmes) l'unité centrale est abandonnée pour la période de temps restante (temps de cycle spécifié - temps de cycle de l'application courante). Si la période de temps est négative, un dépassement est généré et l'unité centrale est abandonnée pour TSK_NBTCKSCHED (variable réglée au démarrage d'ISaGRAF) afin de forcer le scheduling.

  Si le temps de cycle n'est pas spécifié ou si le temps restant est inférieur ou égal à 1 tic ou égal à zéro, l'unité centrale est abandonnée pour TSK_NBTCKSCHED tic(s) pour forcer le scheduling.

  La précision de temps de la cible correspond à un tic standard du système VxWorks.

En général, la spécification du temps de cycle sert à déclencher les cycles ou à mettre l'unité centrale à la disposition des autres tâches exécutées sur le système VxWorks.

- Tant qu'il n'y a pas de données en entrée la tâche de communication se trouve en état de "sommeil". Si nécessaire, cette tâche obtient des informations sur l'application courante à l'aide d'un protocole question/réponse avec la tâche noyau. La tâche de communication envoit une requête au noyau. A la fin du cycle (pour obtenir une image synchrone de l'application), le noyau donne la réponse à la tâche de communication.

Les tâches ISaGRAF ne modifient pas les priorités qui leur ont été données. L'utilisateur peut librement ajuster ces priorités selon le comportement des tâches ISaGRAF décrites ci-dessus et suivant les besoins globales de l'application.

C.6 Mise en route de la cible ISaGRAF NT

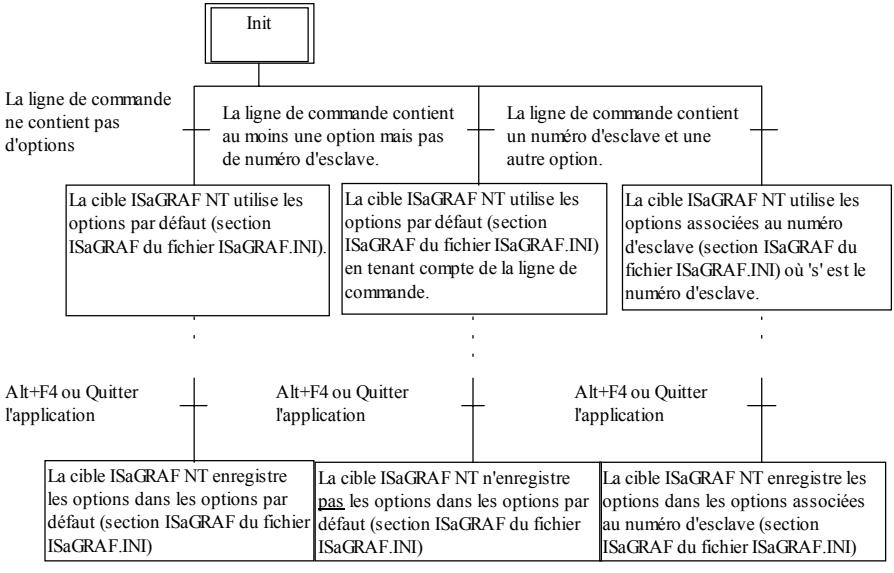
C.6.1 Exécution d'ISaGRAF

Dans l'implémentation NT la cible est exécutée en un seul programme: WISAKER.EXE, qui peut être lancé plusieurs fois. Ceci permet d'exécuter autant de cibles ISaGRAF NT que souhaitées, car chaque instance possède son propre numéro d'esclave.

Le programme cible n'empêche pas l'exécution de routines contrôlées par interruption.
Le logiciel WISAKER est conçu pour être exécuté sous Windows NT 3.51 ou plus.

C.6.2 Informations générales sur les options

Les options sont enregistrées et restituées selon le diagramme suivant:



Note: Le fichier ISAGRAF.INI est enregistré dans le répertoire courant.

▬ *Numéro d'esclave: option -s*

Cette option définit le numéro d'esclave de la cible. Il est compris entre 1 et 255, à l'exception du numéro 13 ($0D). Le numéro d'esclave est utilisé par le protocole de la liaison de communication et sert principalement à distinguer différents esclaves lorsque plusieurs cibles sont connectées au même atelier hôte ou quand plusieurs cibles sont exécutées sur le même PC. Si vous utilisez le debugger de l'atelier, assurez-vous que les esclaves de l'atelier (voir Guide d'utilisateur: Gestion des programmes) correspondent à ceux de la cible.

Valeur par défaut: Le numéro d'esclave par défaut est 1 ou celui contenu dans le fichier ISaGRAF.INI.

Exemple:
WISAKER.EXE -s=2

Interface utilisateur: Cette fenêtre s'affiche en sélectionnant la commande "Options/Slave number" dans la fenêtre principale de la cible ISaGRAF NT.

La valeur de cette option peut être modifiée en utilisant la souris ou les boutons flèches (Vers le haut et Vers le bas). Il faut relancez la cible ISaGRAF NT pour pouvoir utiliser la nouvelle valeur.

Liaison de communication et configuration: option -t

La cible ISaGRAF peut utiliser une liaison série ou une liaison Ethernet pour la communication avec le debugger. Le nom du port est défini à l'aide de l'option -t. Puisque l'interface de communication a été conçue pour être compatible avec tout type de machine, les ports COM1, COM2, COM3 ou COM4 peuvent être utilisés pour la communication série, et les numéros de port commençant par 1100 peuvent être utilisés pour la communication Ethernet.

Valeur par défaut: Le port de communication par défaut est 1100 pour Ethernet et COM1 pour la communication série (ou celui contenu dans le fichier ISaGRAF.INI).

NOTE: La liaison de communication par défaut est Ethernet.

Exemples:
WISAKER -t=COM2
WISAKER -t=1101

Configuration série:
Certaines options peuvent seulement être utilisées lorsque l'option -t=COMx est spécifiée.

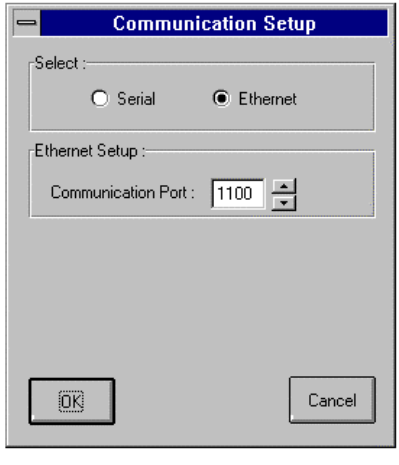
Voici les options utilisées pour la configuration de la liaison série:

Option	Valeurs	Signification
baud	600 1200 2400 4800 9600 **19200**	Vitesse baud
parity	**n** e o	Sans parité Pair Impair
data	7 o. **8**	Nombre de bits
stop	**1** o. 2	Nombre de bit de stop
flow	h **n**	Contrôle matériel Pas de contrôle

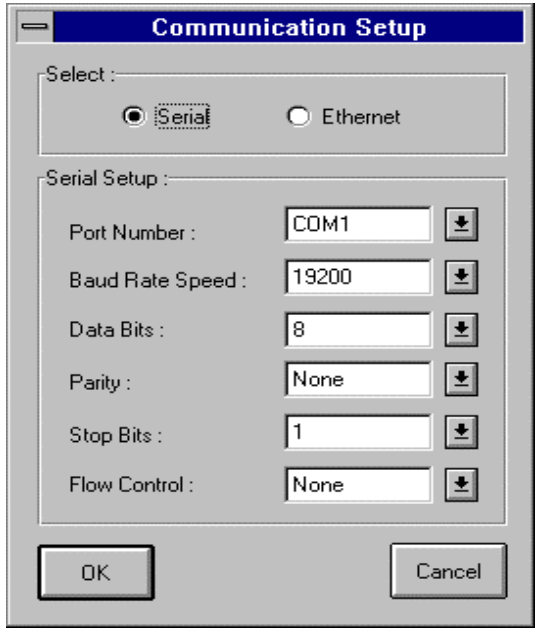
Les valeurs par défaut sont 19200, sans parité, 8 bits de données, 1 bit de stop, pas de contrôle

Exemple:
WISAKER -t=COM1 baud=1200 data=8 parity=n stop=2

Interface utilisateur: Cette fenêtre s'affiche en sélectionnant la commande "Options/Communication" dans la fenêtre principale de la cible ISaGRAF NT.



Il est possible de choisir entre la communication série et la communication Ethernet. La communication Ethernet permet de modifier le numéro de port. Ce numéro de port doit être le même que dans la spécification de la liaison PC-PLC de l'atelier.



En sélectionnant la communication série la configuration s'affiche. Cette configuration doit être la même que celle spécifiée pour la liaison PC-PLC de l'atelier.

Simulation graphique des cartes virtuelles: option -x

Si cette option est utilisée, les cartes déclarées virtuelles dans l'éditeur de connection E/S (voir partie A) sont simulées.
Les valeurs utilisées sont 0 ou 1. 0 signifie pas de simulation, 1 signifie simulation validée.

Valeur par défaut: La valeur par défaut est 0 ou celle contenue dans le fichier ISaGRAF.INI.

Exemple:
WISAKER -x=1 valide la simulation des cartes virtuelles.

Interface utilisateur: L'élément de menu sera validé ou pas suivant l'état de l'option. Les cartes simulées s'affichent dans un panneau graphique.

Priorité de la cible ISaGRAF NT: option -p

Puisque la cible est exécutée sous NT, il est très utile de spécifier un niveau de priorité. Il est par exemple possible d'exécuter une application ISaGRAF de temporisation critique dans une cible avec une priorité plus haute ou d'exécuter une ou plusieurs cibles ayant des priorités plus basses en arrière-plan.

Les valeurs possibles sont 0, 1, 2 ou 3. 0 est la plus haute priorité et 3 la plus basse.

Exemples:
WISAKER -p=0
WISAKER -p=1

Interface utilisateur: Cette fenêtre s'affiche en sélectionnant la commande "Options/Priorité" dans la fenêtre principale de la cible ISaGRAF NT.

La priorité la plus haute est temps réel et la plus basse est temps mort.
0: Priorité temps réel
1: Priorité haute
2: Priorité normale
3: Priorité temps mort

Exemples:

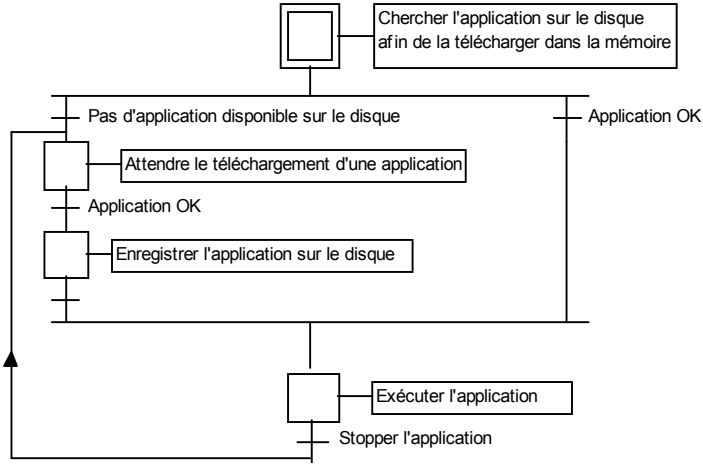
wisaker -t=COM1 Lance la cible ISaGRAF avec le numéro d'esclave par défaut (1) et avec COM1 comme port de communication.

wisaker -s=3 -t=COM1 Lance la cible ISaGRAF avec le numéro d'esclave 3 et avec COM1 comme port de communication.

C.6.3 Fonctionnalités spécifiques

Lancement d'ISaGRAF

Au lancement de la cible l'algorithme suivant est exécuté.



- **Définitions**

Le code d'application est la base de données binaire générée et téléchargée par l'atelier et ensuite exécutée par la cible. Il peut être complété par la table des symboles.
La table des symboles d'une application est une base de données ASCII générée et téléchargée par l'atelier. Cette table réalise la liaison entre les objets symboliques et les objets internes de la cible. Elle n'est pas indispensable dans la cible, sauf pour la gestion des symboles spécifiques de l'utilisateur, comme la fonction DDE ou la simulation des cartes E/S avec la fonction des noms symboliques. Pour plus d'informations sur la table des symboles voir le Guide d'Utilisateur: Techniques de programmation avancées.

- **Applications multiples ISaGRAF**

Sur une même unité centrale, des applications différentes peuvent être exécutées en même temps, à condition de posséder des numéros d'esclave différents et des numéros logiques de tâche de communication différents, respectivement. Lorsque des applications différentes sont exécutées, l'utilisateur doit néanmoins faire attention à l'accès partagé de certains objets de l'application, tel que les cartes E/S. Il se peut que différentes applications (noyaux) utilisent des cartes physiques distinctes, sauf si un serveur E/S ou sémaphore est implémenté à travers le driver E/S.

- **Sauvegarde de l'application**

Quand une nouvelle application est téléchargée par le debugger de l'atelier dans la cible le code d'application est sauvegardé dans le répertoire courant de la cible sous le nom:

ISAx1 Fichier de sauvegarde du code d'application ISaGRAF (x est le numéro d'esclave)

Si la table des symboles de l'application est téléchargée auparavant, elle est aussi sauvegardée dans le répertoire courant de la cible sous le nom:

ISAx6 Fichier de sauvegarde des symboles de l'application ISaGRAF (x est le numéro d'esclave)

Lorsque la cible ISaGRAF est lancée, ces fichiers de code et de symboles sont recherchés dans le répertoire courant et chargés dans la mémoire.
Ensuite, si le fichier des symboles n'est pas disponible, la cible commence l'exécution du code d'application sans les symboles.
Si aucun code d'application n'est diponible, la cible attend le téléchargement d'une application.

Afin de démarrer la cible avec une application spécifique au moment de la mise sous tension, sans utiliser la liaison debugger, on peut copier ces fichiers à partir du même disque, si l'atelier se trouve sur le même PC, ou bien à l'aide d'une disquette, directement sur le disque répertoire courant de la cible.

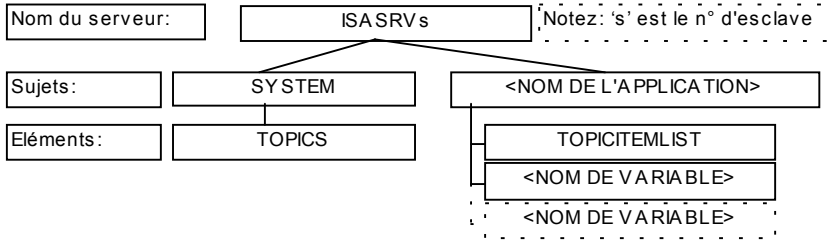
Si l'atelier ISaGRAF est installé dans le répertoire standard \ISAWIN:
le fichier de code d'application du projet MYPROJ est:
\ISAWIN\APL\MYPROJ\appli.x8m
le fichier de symboles de l'application du projet MYPROJ est:
\ISAWIN\APL\MYPROJ\appli.tst

Exemple:
A partir du répertoire, sur lequel est installé WISAKER.EXE, si la commade suivante est entrée:
copy \ISAWIN\APL\MYPROJ\appli.x8m isa11
Alors WISAKER.EXE cherchera l'application 'myproj' et l'exécutera.

Toutes ces commandes peuvent être groupées, par exemple dans un fichier Batch, et lancées à partir du menu d'outils de l'atelier (voir Guide d'utilisateur: Gestion des programmes)

Spécification DDE

La cible ISaGRAF NT inclut un serveur DDE (Dynamic Data Exchange). Tout logiciel client peut être connecté à la cible pour échanger des variables. Par exemple, MSEXCEL peut animer des graphiques avec des valeurs en provenance de la cible ISaGRAF via DDE.
La fonction DDE a besoin de la table des symboles de l'application sur la cible.
Les sujets DDE sont définis comme suit:



« ISASRVs » est le nom du serveur DDE, 's' est le numéro d'esclave.
« SYSTEM » est un sujet standard donnant accès à l'élément « TOPICS ».
« TOPICS » donne la liste des sujets définis: System, et le nom de l'application en exécution dans la cible ISaGRAF NT.
« NOM DE L'APPLICATION » est le nom de l'application.

« TOPICITEMLIST » est la liste des éléments disponibles sous le sujet courant, ceci donne la liste des variables accessibles par DDE.
« NOM DE VARIABLE » est le nom d'une variable.

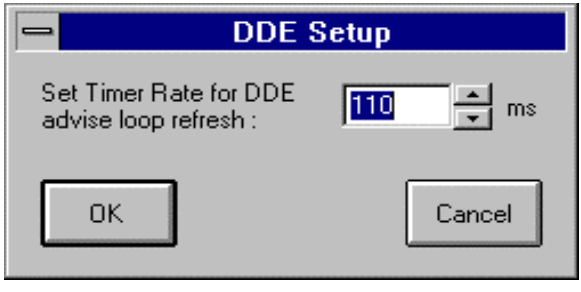
DDE Advise Loop Rate pour cibles ISaGRAF NT: option -d
En général, le client DDE appelle les variable à chaque fois qu'il a besoin d'elles. Si les variables d'une application sont très nombreuses, ceci peut prendre un temps considérable. Il existe un mode appelé Mode Advise (advise loop) dans lequel le serveur envoie uniquement les variables modifiées. Ainsi, les communications sont réduites et efficaces. Dans ce mode, les variables marquées comme "advised" sont regardées périodiquement par le serveur pour décider quelles variables doivent être envoyées. Cette période s'appelle DDE Advise Loop Rate.

Le DDE Advise Loop Rate (en ms) est spécifié avec l'option -d.

Valeur par défaut: La valeur par défaut est 1000 ms ou la valeur contenue dans le fichier ISaGRAF.INI.

Exemple:
WISAKER -d=100

Interface utilisateur: Cette fenêtre s'affiche en sélectionnant la commande "Options/DDE" dans la fenêtre principale de la cible ISaGRAF NT.



Gestion des erreurs et messages de sortie

Le logiciel de la cible ISaGRAF permet la gestion de la détection d'erreurs. En annexe vous trouverez la liste des messages d'erreurs avec leur description.

La détection d'erreurs est traitée de la manière suivante:
- Une erreur se compose d'une erreur et d'un numéro d'argument qui sont envoyés à la routine d'erreurs d' ISaGRAF.
- Si la détection d'erreurs a été validée dans les peremètres d'exécution de l'atelier, l'erreur sera traitée. Sinon les informations seront perdues et la détection d'erreurs s'arrêtera là.

Si les erreurs sont traitées:
- Le numéro d'erreur (valeur décimale) et l'argument (valeur hexadécimale) s'afficheront dans la sortie (fenêtre du WISAKER.EXE).
- Le numéro d'erreur et l'argument seront envoyés dans un buffer de type FIFO, pour pouvoir être récupérés plus tard. La taille du buffer est définie dans les options

"paramètres d'exécution" de l'atelier. Lorsque le buffer est plein, à chaque arrivée d'une nouvelle erreur l'erreur la plus ancienne sera perdue.
- Les erreurs peuvent être appellées soit par le debugger soit par l'application courante en utilisant l'appel SYSTEM (voir Guide d'utilisateur).

Si le debugger a détecté une erreur, un message avec la description de l'erreur s'affiche dans la fenêtre d'erreur. Selon le contexte de l'application (en exécution ou non) le debugger peut afficher le nom de l'objet (variable ou programme) dans lequel se trouve l'erreur, ou bien l'erreur d'argument (valeur décimale) entre parenthèses [x] ce qui peut avoir une signification différente pour chaque erreur.

Un message de bienvenue s'affiche sur la sortie lorsque la cible démarre. Il se compose du numéro d'esclave, de la configuration de communication et du nom du serveur DDE.

Horloge système

Puisque la cible ISaGRAF a été conçue pour être exécutée sur tout système, la référence temporelle utilisée pour la synchronisation du cycle, ainsi que pour la mise à jour des variables de temporisation, est le tic standard de 10 millisecondes.

Ainsi, il est impossible d'obtenir une précision plus grande que 10 ms pour les variables de temporisations. Pour la même raison, un temps de cycle de moins ou égal à 10 ms et différent de zéro génère un dépassement du temps de cycle (erreur 62). Voir le chapitre suivant pour plus d'informations.

Demandez une implémentation spécifique à votre fournisseur si votre application a besoin de plus de précision.

Durée de cycle et comportement de la cible

- A la fin d'un cycle ISaGRAF, juste avant le début d'un nouveau cycle, l'algorithme suivant est exécuté:

  Si un temps de cycle est spécifié (à partir de l'atelier: voir Guide d'utilisateur: Gestion des programmes) l'unité centrale est abandonnée pour la période de temps restante (temps de cycle spécifié - temps de cycle de l'application courante). Si la période de temps est négative, un dépassement est généré et l'unité centrale est abandonnée pour 1 tic afin de forcer le scheduling.

  Si le temps de cycle n'est pas spécifié ou si le temps restant est inférieur ou égal à 1 tic, l'unité centrale est abandonnée pour 1 tic afin de forcer le scheduling.

  La précision de temps de la cible correspond à un tic standard du système Windows NT.

En général, la spécification du temps de cycle sert à déclencher les cycles ou à mettre l'unité centrale à la disposition des autres tâches exécutées sur le système Windows NT.

Quitter ISaGRAF par le clavier

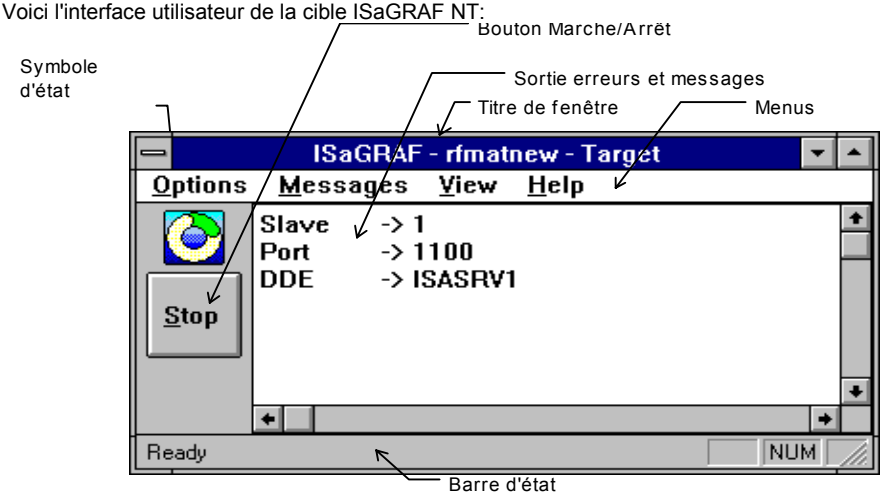
En testant une application sous des conditions non-industrielles sur un PC, l'utilisateur peut arrêter ISaGRAF en tapant une combinaison de touches, évitant des arrêts non voulus. Cette séquence de touches est la suivante:

alt + F4

Un effet secondaire dangereux de ces sorties rapides est que l'interface des cartes E/S n'est pas fermée. Pour cette raison la manière correcte d'arrêter la cible ISaGRAF est la suivante:
- arrêter l'application à partir du debugger ou avec le bouton Marche/Arrêt (les cartes E/S sont fermées)
- stopper la cible ISaGRAF à partir du menu système.

C.6.4 Interface utilisateur

Voici l'interface utilisateur de la cible ISaGRAF NT:



Voici les principaux éléments:

la barre de titre de la fenêtre
la barre de menu
le symbole d'état
le bouton Marche/Arrêt
la sortie erreurs et messages
la barre d'état.

Le titre de la fenêtre contient « ISaGRAF - name_of_appli - target », où name_of_appli est le nom de l'application courante. Si aucune application n'est exécutée, le titre contient « ISaGRAF - - Target ».

Barre de menu de la cible ISaGRAF NT:

La barre de menu contient 4 menus:

Options
Messages
View (Affichage)
Help (Aide)

- **Menu "Options"**

(voir aussi la première section de NT:Informations générales sur les options)

Le menu "**Options**" donne accès aux options d'exécution. Les options suivantes sont disponibles:

Slave number donne accés à la modification du numéro d'esclave. L'option modifiée sera activée au prochain démarrage de la cible seulement. Cette fonction n'est pas disponible si la cible a été lancée avec au moins une option dans la ligne de commande.

Communication donne accés à la configuration de communication. L'option modifiée sera activée au prochain démarrage de la cible seulement. Cette fonction n'est pas disponible si la cible a été lancée avec au moins une option dans la ligne de commande (sauf l'option -s).

DDE donne accés à la modification au DDE Advise Loop Rate. L'option modifiée sera activée au prochain démarrage de la cible seulement. Cette fonction n'est pas disponible si la cible a été lancée avec au moins une option dans la ligne de commande (sauf l'option -s).

I/O simulation valide la simulation des E/S.. L'option modifiée sera activée au prochain arrêt/démarrage de la cible seulement.

Priority donne accés à la modification des priorités. L'option modifiée sera activée immédiatement.

Default Setting récupère les options d'exécution par défaut, mais seulement:
- Communication
- DDE
- Coordinées de la fenêtre affichée à l'écran.

Les options modifiées seront activées au prochain démarrage de la cible seulement. Cette fonctionalité n'est pas disponible si la cible a été lancée avec au moins une option dans la ligne de commande (sauf l'option -s).

- **Menu "Messages"**

Le menu "Messages" sert à la gestion des sorties. Il contient les commandes suivantes:

Aknoledge (Confirmer) arrête le clignotement rouge en cas d'erreurs ou de messages.

Clear (Effacer) efface la sortie entière.

L'icone de la cible ISaGRAF NT:

L'icone reflète les états de la cible:

- Le symbole tourne, lorsque l'application est en état d'exécution.
- Le symbole ne tourne pas, s'il n'y a pas d'application (ou si l'application est arrêtée).
- Le milieu du symbole clignote rouge, quand il y a des erreurs ou messages affichés dans la fenêtre de sortie. Afin d'arrêter le clignotement, sélectionnez « Acknoledge » dans le menu « Messages » ou « Clear » dans le même menu (attention, cette commade efface la totalité des messages de sortie). Vous

trouverez des informations complémentaires sur les erreurs dans le chapitre "Gestion des erreurs et messages de sortie".

Les différents états sont résumés dans le tableau suivant:

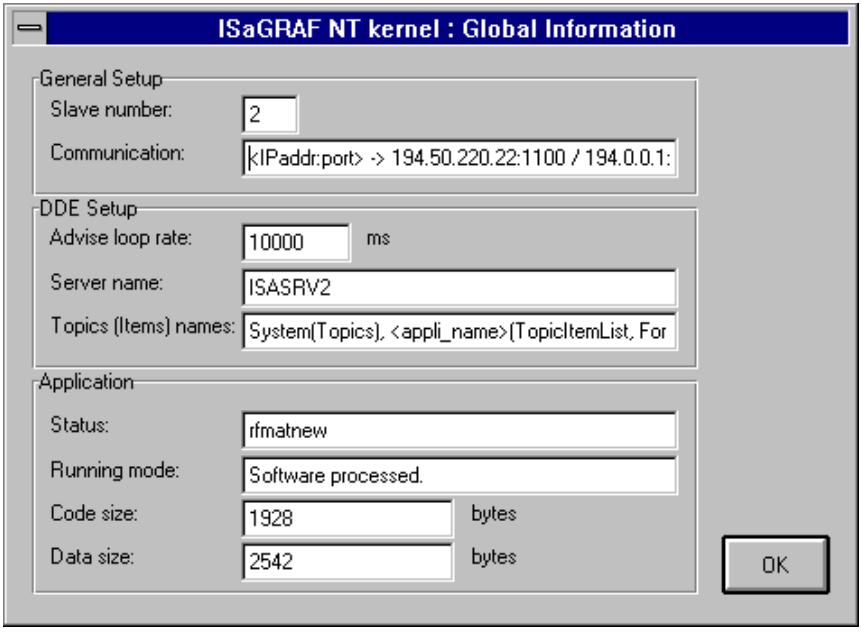
	Pas d'erreurs	Erreurs ou messages (milieu rouge)
Application en exécution		
Pas d'application		

Le bouton Marche/Arrêt de la cible ISaGRAF NT:

Le bouton Marche/Arrêt est identique à la fonction Marche/Arrêt du debugger. Le texte affiché sur le bouton reflète l'état d'exécution de l'application. Si l'application est en état d'exécution, le texte sera "Stop". Si l'application est arrètée (ou s'il n'y a pas d'application) le texte sera "Run". (Notez que s'il n'y a pas d'application disponible et la commande "Run" est donnée, le bouton se met en mode "Stop" et revient toute de suite en mode "Run").

Informations générales sur la cible ISaGRAF NT

La commande "View / Information" affiche la boîte de dialogue suivante qui donne des informations générales sur la configuration de la cible et l'application courante:



Il y a trois sujets:

a) General setup:
   - Numéro d'esclave
   - la configuration de la communication (si la liaison de communication est Ethernet une liste des adresses IP disponibles sur la cible courante est affichée en plus du numéro de port)

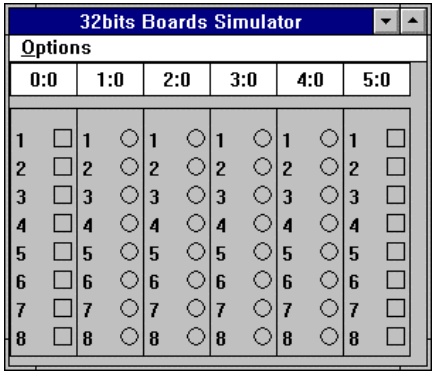
b) DDE setup
   - Advise Loop Rate
   - Nom du serveur DDE
   - Nom des sujets et éléments DDE. Ce sont des informations générales. Les valeurs réelles ne sont pas reflètées. En fait, les champs entre < > doivent être remplacés par les valeurs réelles.

c) Application
   - L'état de l'application, c'est à dire le nom de l'application en exécution ou la chaîne de caractères 'No application', s'il n'y a pas d'application en exécution.
   - Mode d'exécution de l'application qui indique si l'application est en exécution via le processeur logiciel. Dans ce cas la mention « Software processed » sera affichée. Si l'application a été compliée par un compilateur C la mention « C compiled » sera affichée. S'il n'y pas d'application en exécution la mention « No application » sera affichée.
   - La taille du code en octets. Si le mode d'exécution est « C compiled » ce champ affiche zéro.

- Taille des données en octets. Ceci est la somme des données internes d'exécution et de la base de données des variables.

Simulation des cartes virtuelles sur la cible ISaGRAF NT:

Si vous sélectionnez l'option « I/O Simulation » au prochain lancement de l'application la fenêtre suivante sera affichée:



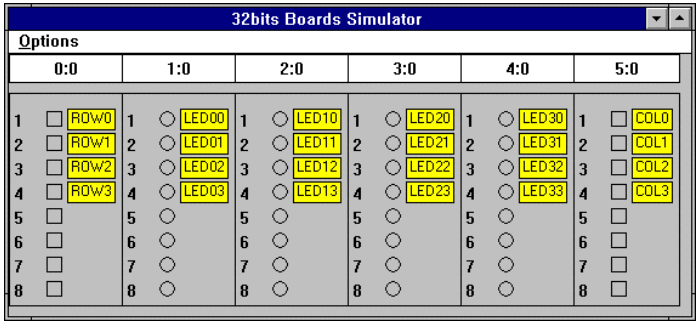
Suivant la configuration E/S un certain nombre de cartes et variables sont affichées. Les chiffres « s:b » au-dessus de chaque carte représentent les identificateurs de l'emplacement (s) et de la carte (b). On compte à partir de zéro (ne peut être modifié).
La fenêtre de 'Simulateur de cartes 32bits' travaille avec l'état d'application Marche/Arrêt. Donc, si une application en exécution possède (ou utilise) des cartes virtuelles et si l'option « I/O Simulation » est validée, la fenêtre s'affiche. Par contre, si on presse le bouton "Stop" la fenêtre se ferme aussitôt. Cette fenêtre travaille parallèlement aux appels E/S.

Le menu "**Options**" contient deux éléments:

Noms de variables affiche les noms des variables, mais seulement si la table des symboles a été téléchargée avant le code Tic.

Valeurs hexadécimales affiche chaque nombre entier en format hexadécimal au lieu du format décimal par défaut.

Les noms des variables auront l'apparence suivante:


32bits Boards Simulator
Options
0:0
1:0
2:0
3:0
4:0
5:0
1 ROW0
2 ROW1
3 ROW2
4 ROW3
5
6
7
8
1 LED00
2 LED01
3 LED02
4 LED03
5
6
7
8
1 LED10
2 LED11
3 LED12
4 LED13
5
6
7
8
1 LED20
2 LED21
3 LED22
4 LED23
5
6
7
8
1 LED30
2 LED31
3 LED32
4 LED33
5
6
7
8
1 COL0
2 COL1
3 COL2
4 COL3
5
6
7
8

C.7 Programmation "C"

C.7.1 Présentation

Ce manuel est dédié aux utilisateurs ayant déjà une bonne expérience des concepts ISaGRAF et des outils de l'atelier. Après avoir développé des applications purement "automate", puis des applications utilisant des **fonctions de conversion**, **fonctions en "C"** et des **blocs fonctionnel** des librairies standard de CJ International, l'utilisateur peut créer d'autres conversions, fonctions et blocs fonctionnels. Il est alors possible d'augmenter la puissance de l'automate cible ISaGRAF en créant de nouvelles librairies, pour tirer le meilleur parti de la flexibilité de l'atelier ISaGRAF et des ouvertures matérielles.

Avec un système de développement "C" et avec une bonne pratique du langage "C", ce manuel permet à l'utilisateur de faire évoluer son automate vers une machine de contrôle commande parfaitement dédiée à son métier. Les librairies développées permettent d'augmenter les performances de l'automate cible, et permettent au programmeur automaticien de développer plus facilement des applications de haute qualité.

Les informations contenues dans ce document ne sont pas dédiées à un système cible particulier. Toutefois, certaines fonctionnalités (telles que les applications multi-tâches) ne pourront être mises en oeuvre sur certains systèmes monotâche.

Fonctions de l'atelier ISaGRAF

L'atelier ISaGRAF inclut de nombreuses fonctions dédiées à la gestion des librairies de composants en "C" et à leur incorporation dans les applications d'automatisme. Pour le programmeur automaticien, une conversion, une fonction ou un bloc fonctionnel en "C" est une "**boîte noire**", entièrement définie par son interface.

Le Gestionnaire de Librairies d'ISaGRAF est utilisé pour la déclaration des nouveaux éléments de librairie, la définition de leur interface et de leur utilisation dans les programmes **ST** ou **FBD**. Le Gestionnaire de Librairies d'ISaGRAF offre également des fonctions de génération automatique de code source "C" pour le développement des conversions, fonctions et blocs fonctionnels, et inclut des outils d'édition des fichiers sources "C". Référez-vous à la première partie de ce manuel (**Utilisation de l'atelier**) pour plus d'information sur l'utilisation du Gestionnaire de Librairies.

Développement en langage "C"

L'atelier ISaGRAF n'inclut pas de compilateur "C" ni d'outil de développement croisé pour ce langage. L'utilisateur devra disposer d'outils de développement en langage "C", dédiés au système installé sur son automate cible, pour pouvoir intégrer de nouveaux composants en "C" au noyau cible ISaGRAF.

Si l'outil utilisé est un système de développement croisé, l'atelier ISaGRAF offre des points d'entrée disponibles pour lancer les commandes de compilation et d'édition de liens dans une fenêtre MS-DOS (fichier de commande .BAT). Dans ce cas, l'utilisateur doit s'assurer que les outils de développement "C" peuvent fonctionner dans le contexte d'une fenêtre MS-DOS de

Windows. Sinon, Windows devra être fermé pour lancer le compilateur dans un enrironnement strict MS-DOS.

Fiches techniques

Le Gestionnaire de Librairies d'ISaGRAF permet la saisie d'un texte de description associé à chaque élément des librairies. Cette **fiche technique** est le guide d'utilisation du composant développé en "C". Il est dédié au programmeur automaticien, et décrit comment utiliser la conversion, la fonction ou le bloc fonctionnel dans une application ISaGRAF.

La conversion, la fonction ou le bloc fonctionnel doit être précisément décrit dans sa fiche technique, afin que le programmeur automaticien (qui peut n'avoir aucune connaissance du langage "C") puisse réellement l'utiliser comme une fonction standard des langages d'ISaGRAF. Pour une fonction "C", la fiche technique doit décrire :

- ❑ l'exacte fonction réalisée
- ❑ la description détaillée des paramètres d'appel
- ❑ la signification du paramètre de retour
- ❑ le typage exact des paramètres d'appel et de retour
- ❑ le contexte d'utilisation

Pour un bloc fonctionnel "C", la fiche technique doit décrire :

- ❑ l'exacte fonction réalisée à l'activation du bloc
- ❑ la description détaillée des paramètres d'appel
- ❑ la signification des paramètres de retour
- ❑ le typage exact des paramètres d'appel et de retour
- ❑ le contexte d'utilisation

Pour une fonction de conversion, la fiche technique doit décrire :

- ❑ la conversion exacte appliquée sur une variable d'entrée
- ❑ la conversion exacte appliquée sur une variable de sortie
- ❑ les plages de valeurs traitées et le contexte d'utilisation

Les fiches techniques peuvent également contenir les informations suivantes :

- ❑ l'identification précise de l'élément
- ❑ toute information concernant ses évolutions et sa maintenance
- ❑ le ou les systèmes cibles supportés
- ❑ les aspects spéciaux inhérents au fonctionnement multi-tâches
- ❑ les services systèmes requis, mémoire, drivers...

C.7.2 Fonctions de conversion

L'atelier ISaGRAF intègre un outil de développement de **tables de conversions linéaires** pour le filtrage des E/S analogiques. Cet utilitaire ne nécessite aucun développement en "C". Il est limité aux conversions linéaires par segments, strictement croissantes ou décroissantes. Référez-vous au Guide de l'utilisateur d'ISaGRAF pour une description détaillée de cet outil.

Les fonctions de conversion permettent l'intégration de conversions complexes, intégrant des opérations spécifiques décrites en langage "C". Par définition, une fonction de conversion peut être utilisée aussi bien en **entrée** qu'en **sortie**. Même si une direction (entrée ou sortie) n'est pas utilisée, elle devra être implémentée et testée avant l'intégration de la conversion dans une application, afin de protéger le système cible ISaGRAF de toute mauvaise utilisation.

Les fonctions de conversion sont écrites en langage "C", compilées, et liées avec le noyau exécutif ISaGRAF. Il en résulte un nouveau noyau, qui doit être installé sur la cible avant de pouvoir utiliser les nouvelles conversions dans des applications ISaGRAF. Les nouvelles fonctions de conversion ne peuvent pas être intégrées au simulateur d'ISaGRAF. Les applications ISaGRAF devront être simulées **avant** l'utilisation dans leur dictionnaire de variables des nouvelles fonctions de conversion.

Le code source "C" des conversions standard livrées par CJ International est installé sur le système de développement avec l'atelier ISaGRAF. Elles peuvent servir d'exemple pour le développement de nouvelles conversions. Toutefois, il est recommandé de **ne jamais modifier** les conversions standard pour préserver la cohérence de l'atelier ISaGRAF. Les fonctions de conversions standard installées avec l'atelier ISaGRAF sont supportées par le simulateur ISaGRAF.

Attention: Les fonctions de conversion représentent des opérations **synchrones**, activées à l'exécution par le gestionnaire d'E/S du système cible ISaGRAF, pendant les phases d'entrée ou de sortie de chaque cycle. Le temps passé à exécuter une fonction de conversion est inclus dans le **temps de cycle** de l'application ISaGRAF. Le développeur doit s'assurer qu'aucune opération bloquante n'est programmée dans une fonction de conversion, afin de préserver le mécanisme d'exécution du noyau ISaGRAF.

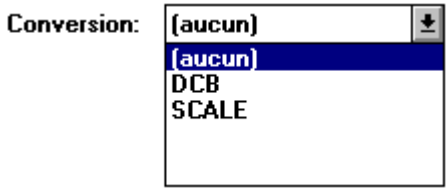
Ajouter une fonction à la librairie ISaGRAF

Le Gestionnaire de Librairie d'ISaGRAF doit être utilisé pour ajouter une nouvelle fonction de conversion à la librairie ISaGRAF. Avec l'atelier, utilisez la commande "**Nouveau**" du menu "**Fichiers**", quand la librairie des fonctions de conversion est sélectionnée. Aucun paramètre ne doit être saisi dans l'atelier, puisqu'une fonction de conversion travaille sur une interface prédéfinie et non modifiable.

Quand une nouvelle fonction de conversion est créée, sa **fiche technique** doit être rédigée. La trame du code source "C" pour la nouvelle fonction de conversion est automatiquement générée par le Gestionnaire de Librairie d'ISaGRAF.

Utiliser une conversion dans un projet ISaGRAF

Les fonctions de conversion définies peuvent être utilisées pour filtrer les variables d'entrée ou de sortie analogiques d'un projet. Pour attacher une fonction de conversion à une variable, utilisez l'outil d'édition du dictionnaire, sélectionnez la variable d'E/S analogique et ouvrez la boîte de dialogue pour l'édition de ses paramètres. Utilisez le champ "**conversion**" de la boîte de dialoque pour sélectionner la fonction de conversion qui doit être attachée à cette variable :



Les tables et les fonctions de conversion apparaissent ensemble dans la liste. Ceci implique que le même nom ne peut pas être donné à une table et une fonction. La même fonction de conversion peut être attachée à plusieurs variables. Une variable ne peut pas être attachée à une fonction de conversion qui n'a pas encore été définie et intégrée dans au noyau exécutif ISaGRAF.

Interface "C" standard

L'interface d'une fonction de conversion a toujours le même format. Les paramètres d'appel et de retour sont passés par l'intermédiaire d'une structure. Cette structure est définie dans le fichier "**TACN0DEF.h**":

```
/* Nom : tacn0def.h - Fichier de définitions standard - conversions */

#define TRUE                1
#define FALSE               0
#define CR_OK               0
#define CR_BAD              -1
#define DIR_INPUT           0       /* direction = conversion en entrée */
#define DIR_OUTPUT          1       /* direction = conversion en sortie */

typedef long T_ANA;             /* type ISaGRAF: analogique entier */
typedef float T_REAL;           /* type ISaGRAF: analogique réel */

typedef struct {                /* structure d'interface */
    unsigned short number;      /* numéro de conversion (réservé) */
    unsigned short direction;   /* direction de la conversion */
    T_REAL *before;             /* valeur avant la conversion */
    T_REAL *after;              /* valeur après la conversion */
  } str_cnv;

#define ARG_BEFORE          (*(arg->before))
#define ARG_AFTER           (*(arg->after))
#define DIRECTION           (arg->direction)

/* fin de fichier */
```

La structure "**str_cnv**" contient tous les éléments de l'interface. Le seul paramètre passé à la fonction "C" est un pointeur vers une telle structure. Le champ "**number**" contient le numéro logique de la conversion (numéro dans la librairie ISaGRAF) et n'est jamais utilisé dans la programmation de la fonction de conversion.

Le champ "**direction**" indique si la conversion doit être appliquée en entrée ou en sortie. Il prend la valeur **DIR_INPUT** pour une conversion en entrée, ou la valeur **DIR_OUTPUT** pour une conversion en sortie.

Le champ "**before**" pointe vers la valeur avant la conversion. Ce champ a une signification différente selon que la conversion est appliquée en entrée ou en sortie. Il représente la valeur électrique (lue sur le capteur) pour une conversion en entrée, quand le champ **direction** a la valeur **DIR_INPUT**. Il représente la valeur physique (utilisée dans les programmes de l'application) pour une conversion en sortie, quand le champ **direction** a la valeur **DIR_OUTPUT**.

Le champ "**after**" pointe vers la valeur après la conversion. Ce champ a une signification différente selon que la conversion est appliquée en entrée ou en sortie. Il représente la valeur physique (utilisée dans les programmes de l'application) pour une conversion en entrée, quand le champ **direction** a la valeur **DIR_INPUT**. Il représente la valeur électrique (envoyée à l'actionneur) pour une conversion en sortie, quand le champ **direction** a la valeur **DIR_OUTPUT**.

Le programmeur peut utiliser les définitions "**ARG_BEFORE**" et "**ARG_AFTER**" pour accéder directement aux champs **before** et **after** de la structure passée à la fonction "C". Les valeurs traitées sont des **valeurs flottantes en simple précision**. Le résultat est converti en un entier long (32 bits) si la conversion est appliquée à une variable analogique entière. Ceci implique qu'une même fonction de conversion pourra être utilisée pour des variables analogiques entières et réelles.

Code source

Parce que la conversion peut être appliquée à des variables analogiques d'entrée ou de sortie, le code source "C" de la fonction est divisée en deux parties : la conversion en entrée, et la conversion en sortie. Le champ **direction** est utilisé pour sélectionner le type de conversion à réaliser. Le Gestionnaire de Librairie d'ISaGRAF génère automatiquement toute la trame du code source "C" de la fonction, quand une nouvelle fonction de conversion est créée. Il génère également la structure "**IF**" principale pour la sélection de la direction. Voici le format standard du code source d'une fonction de conversion :

```
/* trame standard d'une conversion */
/* cet exemple montre une conversion nommée "SAMPLE" */

#include <tacn0def.h>          /* définition de la structure d'interface */
#include <tasy0def.h>          /* définitions communes au noyau ISaGRAF */

void CNV_sample (arg)          /* corps de la fonction de conversion */
str_cnv *arg;
{
  unsigned long value, image;

  if (DIRECTION == DIR_INPUT) { /* INPUT CONVERSION */
    /* Conversion en entrée */
   /* ARG_AFTER = _input_conv (ARG_BEFORE); */
  }
  else { /* OUTPUT CONVERSION */
    /* Conversion en sortie */
```

```
    /* ARG_AFTER = _output_conv (ARG_BEFORE); */
  }
}
```

/ La fonction suivante réalise le lien avec le gestionnaire d'E/S d'ISaGRAF, à l'aide du nom de la conversion. Cette fonction est entièrement générée par le gestionnaire de librairies d'ISaGRAF. Le type UFP est défini dans* le fichier *"tasy0def.h", comme un pointeur long vers une fonction "void". */*

```
UFP cnvdef_sample (char *name)
{
  strcpy (name, "SAMPLE");   /* rend le nom de la conversion */
  return (CNV_sample);       /* adresse de la fonction implémentée */
}
```

/ fin de fichier */*

Il est recommandé, pour compléter la trame générée automatiquement, d'écrire deux fonctions isolées (non exportées) dans le même fichier, pour les conversions en entrée et en sortie. Ces fonctions seront appelées par la fonction principale, dans la structure de sélection **IF**, comme le montre l'exemple précédent.

Le fichier "**TASY0DEF.H**" regroupe les définitions communes à tous les fichiers sources du noyau exécutif ISaGRAF. Il doit impérativement être inclus pour assurer l'indépendance par rapport au système d'exploitation de la cible. Il contient entre autre la définition du type **UFP**, qui représente un pointeur "far" vers une fonction de type "void", et qui est utilisé pour la fonction de déclaration.

Liens entre les projets et l'implémentation en "C"

Le lien logique entre l'implémentation en "C" d'une fonction de conversion et son utilisation dans les projets ISaGRAF est réalisé par le nom de la conversion. Une fonction de "déclaration" est ajoutée au code source "C" de la fonction de conversion. Cette fonction sera appelée une seule fois au lancement de l'application, pour réaliser un lien dynamique entre le gestionnaire d'E/S du noyau exécutif ISaGRAF et la fonction implémentée. Voici le format standard d'une telle fonction de déclaration :

```
UFP cnvdef_xxx (char *name)
{
  strcpy (name, "SAMPLE");       /* rend le nom de la conversion */
  return (CNV_sample);           /* adresse de la fonction implémentée */
}
/* (xxx est le nom de la conversion) */
```

Le nom de la fonction, utilisée dans l'instruction **strcpy** doit être écrit en **majuscules**. Il doit être écrit en minuscules dans le nom des fonctions d'implémentation et de déclaration.

L'utilisation des préfixes "**CNV_**" et "**cnvdef_**" pour les fonctions d'implémentation et de déclaration permet de créer des conversions portant le même nom qu'un mot clé du langage "C", ou qu'une fonction déjà définie dans les librairies du "C" et d'ISaGRAF.

D'autres instructions peuvent être ajoutées dans la fonction de déclaration, pour réaliser des opérations d'initialisation, pour cette conversion. Le système ISaGRAF assure que cette fonction est appelée **une seule fois** au lancement de l'application.

La fonction de déclaration sera appelée pour toutes les fonctions de conversion intégrées au noyau, même si elles ne sont pas utilisées dans l'application ISaGRAF.

Le noyau exécutif génèrera une erreur fatale si une fonction de conversion non intégrée est utilisée dans une application.

Avant l'édition de liens du noyau avec les nouvelles fonctions de conversion, le programmeur doit mettre à jour le fichier source "C" nommé "**GRCN0LIB.C**", et compléter la liste des fichiers pour la construction des librairies. Le fichier "**GRCN0LIB.C**" ne contient qu'un tableau des fonctions de déclaration. Ce tableau sera lu pendant la phase d'initialisation du gestionnaire d'E/S d'ISaGRAF, pour réaliser le lien dynamique avec les fonctions de conversion implémentées. Voici un exemple du contenu de ce fichier :

```
/* Fichier "GRCN0LIB.c" - Conversions de la librairie standard */

#include <tasy0def.h>           /* nécessaire pour les déclarations de types */

extern UFP cnvdef_scale ();     /* déclaration de la conversion SCALE */
extern UFP cnvdef_dcb ();       /* déclaration de la conversion DCB */

UFP_LIST CNVDEF[ ] = {          /* tableau des services de définition des */
  cnvdef_scale,                 /* conversions devant être intégrées au */
  cnvdef_dcb,                   /* noyau ISaGRAF */
NULL };

/* fin de fichier */
```

Le tableau **CNVDEF** doit être terminé par un pointeur NULL. Des problèmes sévères peuvent survenir à l'exécution si cette condition n'est pas réalisée. Les fonctions de conversions non incluses dans le tableau **CNVDEF** ne seront pas reconnues par le noyau, même si leur code est pris en compte dans l'édition de liens.

L'écriture de ce fichier permet la construction d'un nouveau noyau, intégrant les nouvelles fonctions de conversions. Il est également possible de créer un noyau dédié à un projet, en n'insérant dans le tableau **CNVDEF** que les conversions utilisées dans ce projet. Le fichier "**GRCN0LIB.C**" est automatiquement généré par le Générateur de Code d'ISaGRAF, quand le code d'une application est fabriqué. Le fichier généré est placé dans le répertoire du projet. Il ne groupe que les conversions référencées dans le dictionnaire de l'application.

Un autre fichier, nommé "**GRCN0LIB.68K**" est automatiquement généré par le Générateur de Code d'ISaGRAF, quand le code de l'application est construit. Ce fichier a exactement la même signification que le fichier "**GRCN0LIB.C**", mais il utilise des conventions d'écriture en assembleur pour déclarer le tableau **CNVDEF**. Il est placé dans le répertoire contenant les données du projet ISaGRAF, et ne regroupe que les conversions référencées dans le dictionnaire de l'application. Voici un exemple d'un tel fichier :

```
#include <tasy0def.h>
```

```
extern UFP cnvdef_scale ();
extern UFP cnvdef_dcb ();

_asm ("
CNVDEF:
      dc.l     cnvdef_scale
      dc.l     cnvdef_dcb
      dc.l     0
");
```

/ fin de fichier */*

Pour plus d'information sur ce fichier, reportez-vous au fichier "**Readme**" de la disquette contenant le logiciel cible ISaGRAF.

Limites

La librairie ISaGRAF peut contenir jusqu'à **128** fonctions de conversion. Tout type d'opération peut être réalisé dans une fonction de conversion. Il convient de se rappeler que ces fonctions sont appelées de façon **synchrone**, et que l'exécution des fonctions de conversion a une influence directe sur le temps de cycle de l'application.

C.7.3 Fonctions "C"

Les fonctions en "C" sont utilisées pour étendre les fonctionnalités des langages **ST** et **FBD**. Elles peuvent être utilisées pour réaliser tout calcul spécifique, pour accéder aux services du système, gérer des communications, ou installer un jeu de services pour le dialogue avec d'autres tâches utilisateur ou d'autres applications ISaGRAF. Ces fonctions sont écrites en langage "C", compilées, et liées avec le noyau exécutif ISaGRAF. Il en résulte un nouveau noyau, qui doit être installé sur la cible avant de pouvoir utiliser les nouvelles fonctions dans des applications ISaGRAF.

Les nouvelles fonctions ne peuvent pas être intégrées au simulateur d'ISaGRAF. Les applications ISaGRAF devront être simulées **avant** l'utilisation dans leurs programmes des nouvelles fonctions.

Attention: Les fonctions "C" réprésentent des opérations **synchrones**, activées par le noyau exécutif ISaGRAF, pendant l'exécution des programmes de l'application. Le temps passé à exécuter une fonction "C" est inclus dans le **temps de cycle** de l'application ISaGRAF. Le développeur doit s'assurer qu'aucune opération bloquante n'est programmée dans une fonction, afin de préserver le mécanisme d'exécution du noyau ISaGRAF.

Ajouter une fonction à la librairie ISaGRAF

Le Gestionnaire de Librairie d'ISaGRAF doit être utilisé pour ajouter une nouvelle fonction "C" à la librairie ISaGRAF. Avec l'atelier, utilisez la commande "**Nouveau**" du menu "**Fichiers**", quand la librairie des fonctions "C" est sélectionnée. Quand une nouvelle fonction est créée, sa **fiche technique** doit être rédigée. La trame du code source "C" pour la nouvelle fonction est automatiquement générée par le Gestionnaire de Librairie d'ISaGRAF.

La commande "**Paramètres**" du menu "**Edition**" permet de définir les paramètres d'appel et de retour de la nouvelle fonction.

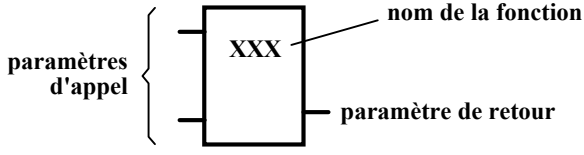
Utiliser une fonction "C" dans un projet ISaGRAF

Toute nouvelle fonction "C" intégrée au noyau peut être utilisée comme une fonction standard dans les programmes des projets ISaGRAF. Les fonction en "C" peuvent être utilisées avec les langages **ST** et **FBD**, et depuis certains énnoncés du langage **SFC**.

L'appel d'une fonction "C" en **ST** respecte les conventions d'appel de fonction du langage. Les paramètres d'appel de la fonction sont écrits après son nom, entre parenthèses, et séparés par des virgules. L'expression représente la valeur retournée par la fonction. Un appel de fonction peut être inséré dans tout énnoncé d'assignation ou expression complexe. Voici un exemple d'appel de fonction "C" dans un énnoncé d'assignation :

resultat := NomFonct (par1, par2, ... parN);

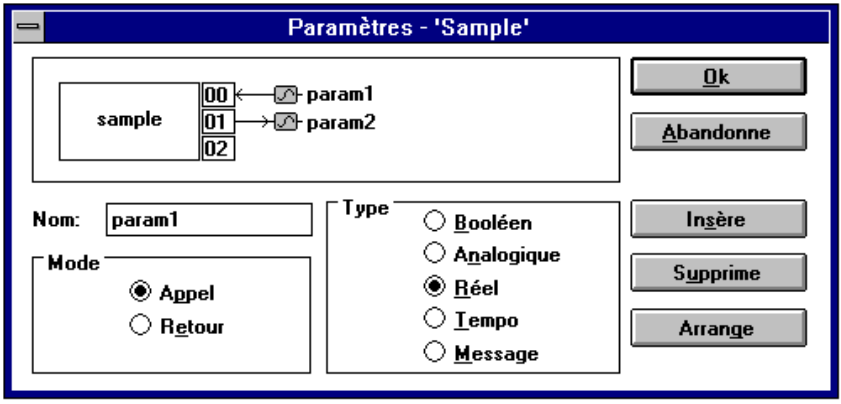
Un programme **FBD** peut appeler toute fonction en "C". Une fonction est représentée comme une boîte. Ses paramètres d'appel son connectés sur le bord gauche de la boîte. Le paramètre de retour est connecté sur le bord droit de la boîte : Voici l'aspect standard d'une fonction dans un diagramme FBD:



Une fonction en "C" peut être appelée dans tout bloc d'action **SFC** (avec l'attribut **P** ou **N**), ou dans toute expression booléenne attachée à une transition.

Définition de l'interface d'une fonction "C"

Utilisez la commande "**Paramètres**" du menu "**Edition**" du Gestionnaire de Librairies pour définir les paramètres d'entrée et de sortie d'une nouvelle fonction. Une fonction peut avoir jusqu'à **31** paramètres d'appel, et a toujours **un seul** paramètre de retour. La boîte de dialogue suivante est utilisée pour décrire les paramètres d'une fonction "C" :



La liste dans la partie supérieure de la fenêtre montre les paramètres de la fonction "C", dans l'ordre défini par le prototype d'appel de la fonction: d'abord les paramètres d'appel, finalement le paramètre de retour. La partie inférieure de la fenêtre montre la description détaillée du paramètre sélectionné dans la liste :

- le nom du paramètre
- son mode (appel ou retour)
- le type du paramètre

Tous les types de données d'ISaGRAF peuvent être utilisés pour un paramètre : booléen, analogique entier ou réel, temporisation ou message. Les analogiques entiers et réels doivent être différenciés. Voici la correspondance entre les types ISaGRAF et les types du langage "C":

BOOLEEN	unsigned long	mot de 32 bits non signé : 1=true / 0=false
ENTIER	long	mot signé de 32 bits
REEL	float	valeur flottante en simple précision (32 bits)
TEMPO	unsigned long	mot non signé de 32 bits (unité = 1ms)
MESSAGE	char *	chaîne de caractères

Quand un message est passé à une fonction "C", il ne peut pas contenir de caractère nul (code ASCII 00). La chaîne passée à la fonction "C" est toujours terminée par un caractère nul. Le paramètre de retour doit être le dernier de la liste. Les règles suivantes doivent être respectées pour la nomenclature des paramètres :

- la longueur du nom ne peut pas excéder **16** caractères
- le premier caractère doit être une **lettre**
- les suivants doivent être des **lettres**, des **chiffres** ou le souligné
- les majuscules et les minuscules ne sont pas différenciées

Le même nom ne peut pas être utilisé pour plusieurs paramètres de la même fonction. Un paramètre d'appel ne peut pas avoir le même nom que le paramètre de retour. Le même nom **peut** être utilisé pour des paramètres de fonctions différentes. Le nom proposé par défaut

pour le paramètre de retour est "**Q**". Ce nom peut être librement modifié. Le nom d'un paramètre est utilisé pour identifier la donnée correspondante dans le code source "C" de la fonction.

La commande "**Insère**" permet d'insérer un nouveau paramètre avant le paramètre sélectionné dans la liste. La commande "**Supprime**" efface le paramètre sélectionné. La commande "**Arrange**" trie automatiquement les paramètres, et positionne le paramètre de retour en fin de liste. Enfoncez le bouton "**Ok**" pour enregistrer la définition des paramètres et refermer la boîte de dialogue. Enfoncez le bouton "**Abandonne**" pour fermer la boîte de dialogue sans modifier la définition des paramètres.

Interface "C" d'une fonction

L'interface d'une fonction en "C" dépend de la définition de ses paramètres. Les paramètres d'appel et de retour sont passés dans une structure. Cette structure est définie dans le fichier"**GRUS0nnn.H**", où "**nnn**" est le numéro logique de la fonction (emplacement dans la librairie ISaGRAF). Voici un exemple de structure d'interface en "C", pour la fonction "**SIN**" (calcul de sinus) de la librairie standard :

```
/* Fichier : GRUS0255.h - fonction "sample" */

typedef  long          T_BOO;
typedef  long          T_ANA;
typedef  float         T_REAL;
typedef  long          T_TMR;
typedef  char          *T_MSG;

typedef struct {
  /* CALL   */         T_REAL   _param1;
  /* RETURN */         T_REAL   _param2;
} str_arg;

#define PARAM1     (arg->_param1)
#define PARAM2     (arg->_param2)

/* fin de fichier */
```

Voici la correspondance entre les types ISaGRAF et les types du langage "C". Les types ISaGRAF sont définis dans le fichier d'interface de la fonction et peuvent être utilisés dans la programmation en "C".

booléen	**T_BOO**	**long** (32 bits)
analogique	**T_ANA**	**long**
réel	**T_REAL**	**float** (32 bits - simple precision)
temporisation	**T_TMR**	**long**
message	**T_MSG**	**char *** (32 bits - pointeur de caractères)

Chaque champ de la structure "**str_arg**" correspond à un paramètre de la fonction. Le paramètre de retour est le dernier de la structure. Les paramètres d'appel apparaissent dans leur ordre de déclaration. Un identificateur en majuscules est défini pour chaque paramètre. Il permet d'accéder directement à la valeur du paramètre dans la structure passée à la fonction

"C". Les noms des paramètres sont ceux saisis lors de la description de l'interface de la fonction avec le Gestionnaire de Librairie d'ISaGRAF.

Le fichier de définitions en "C" est mis à jour automatiquement par le Gestionnaire de Librairie d'ISaGRAF à chaque modification de l'interface de la fonction. Ceci assure la cohérence entre le développement en "C" des fonctions et leur utilisation dans les programmes des applications ISaGRAF.

Code source

Voici la trame standard du code source d'une fontion "C":

```
/* Exemple de fonction - N° "255" - Nom: "SAMPLE" */

#include "tasy0def.h"          /* Définitions communes du noyau ISaGRAF */
#include "grus0255.h"          /* définition de l'interface de la fonction 255 */

void USP_sample (arg)
str_arg *arg;
{
   /* corps de la fonction */
}

/* La fonction suivante est utilisée pour l'initialisation de la fonction et la  déclaration de son
implémentation. Elle réalise le lien avec le noyau exécutif ISaGRAF, à l'aide du nom de la
fonction. Cette fonction est automatiquement générée par le Gestionnaire de Librairie
d'ISaGRAF. Le type UFP, défini dans le fichier "tasy0def.h" est un pointeur long vers une
fonction de type VOID */

UFP uspdef_sample (char *name)
{
  strcpy (name, "SAMPLE");     /* rend le nom de la fonction */
  return (USP_sample);         /* adresse de la fonction implémentée */
}

/* fin de fichier */
```

Le fichier "**TASY0DEF.H**" regroupe les définitions communes à tous les fichiers "source" du noyau exécutif ISaGRAF. Il doit impérativement être inclus pour assurer l'indépendance par rapport au système d'exploitation de la cible. Il contient entre autre la définition du type **UFP**, qui représente un pointeur "far" vers une fonction de type "void", et qui est utilisé pour la fonction de déclaration.

Liens entre les projets et l'implémentation en "C"

Le lien logique entre l'implémentation en "C" d'une fonction et son utilisation dans les projets ISaGRAF est réalisé par le nom de la fonction. Une fonction de "déclaration" est ajoutée au code source "C" de la fonction. Cette fonction sera appelée une seule fois au lancement de l'application, pour réaliser un lien dynamique entre le noyau exécutif ISaGRAF et la fonction implémentée. Voici le format standard d'une telle fonction de déclaration:

```
UFP uspdef_xxx (char *name)
{
  strcpy (name, "SAMPLE");      /* rend le nom de la fonction */
  return (USP_sample);          /* adresse de la fonction implémentée */
}

/* (xxx est le nom de la fonction) */
```

Le nom de la fonction, utilisé dans l'instruction **strcpy** doit être écrit en **majuscules**. Il doit être écrit en minuscules dans le nom des fonctions d'implémentation et de déclaration. L'utilisation des préfixes "**USP_**" et "**uspdef_**" pour les fonctions d'implémentation et de déclaration permet de créer des fonctions portant le même nom qu'un mot clé du langage "C", ou qu'une fonction déjà définie dans les librairies du "C" et d'ISaGRAF.

D'autres instructions peuvent être ajoutées dans la fonction de déclaration, pour réaliser des opérations d'initialisation, pour cette fonction. Le système ISaGRAF assure que cette fonction est appelée **une seule fois** au lancement de l'application. La fonction de déclaration sera appelée pour toutes les fonctions intégrées au noyau, même si elles ne sont pas utilisées dans l'application ISaGRAF. Le noyau exécutif génèrera une erreur fatale si une fonction non intégrée est utilisée dans une application.

Avant l'édition de liens du noyau avec les nouvelles fonctions, le programmeur doit mettre à jour le fichier source "C" nommé "**GRUS0LIB.C**", et compléter la liste des fichiers pour la construction des librairies. Le fichier "**GRUS0LIB.C**" ne contient qu'un tableau des fonctions de déclaration. Ce tableau sera lu pendant la phase d'initialisation du noyau exécutif ISaGRAF, pour réaliser le lien dynamique avec les fonctions implémentées. Voici un exemple du contenu de ce fichier :

```
/* Fichier "GRUS0LIB.c" - Exemple de fonctions trigonométriques */

#include <tasy0def.h>           /* requis pour les définitions de types */

extern UFP uspdef_fc1 ();       /* services de déclaration */
extern UFP uspdef_fc2 ();
extern UFP uspdef_fc3 ();
extern UFP uspdef_fc4 ();

UFP_LIST USPDEF[ ] = {          /* tableau des fonctions de déclarations */
                                /* pour les fonctions intégrées au noyau */
  uspdef_fc1,
  uspdef_fc2,
  uspdef_fc3,
  uspdef_fc4,

NULL };

/* fin de fichier */
```

Le tableau **USPDEF** doit être terminé par un pointeur NULL. Des problèmes sévères peuvent survenir à l'exécution si cette condition n'est pas réalisée. Les fonctions non incluses dans le tableau **USPDEF** ne seront pas reconnues par le noyau, même si leur code est pris en

compte dans l'édition de liens. L'écriture de ce fichier permet la construction d'un nouveau noyau, intégrant les nouvelles fonctions. Il est également possible de créer un noyau dédié à un projet, en n'insérant dans le tableau **USPDEF** que les fonctions utilisées dans ce projet. Le fichier "**GRUS0LIB.C**" est automatiquement généré par le Générateur de Code d'ISaGRAF, quand le code d'une application est fabriqué. Le fichier généré est placé dans le répertoire du projet. Il ne groupe que les fonctions référencées dans les sources de l'application.

Un autre fichier, nommé "**GRUS0LIB.68K**" est automatiquement généré par le Générateur de Code d'ISaGRAF, quand le code de l'application est construit. Ce fichier a exactement la même signification que le fichier "**GRUS0LIB.C**", mais il utilise une convention d'écriture en assembleur pour la déclaration du tableau **USPDEF**. Il est placé dans le répertoire du projet. Il ne groupe que les fonctions référencées dans les sources de l'application. Voici un exemple d'un tel fichier :

```
#include <tasy0def.h>

extern UFP uspdef_fc1 ();
extern UFP uspdef_fc2 ();

_asm ("
USPDEF:
      dc.l     uspdef_fc1
      dc.l     uspdef_fc2
      dc.l     0
");

/* fin de fichier */
```

Pour plus d'information sur ce fichier, reportez-vous au fichier "**Readme**" de la disquette contenant le logiciel cible ISaGRAF.

Limites

La librairie ISaGRAF peut contenir jusqu'à **255** fonctions. Tout type d'opération peut être réalisé dans une fonction. Il convient de se rappeler que ces fonctions sont appelées de façon **synchrone**, et que l'exécution des fonctions a une influence directe sur le temps de cycle de l'application.

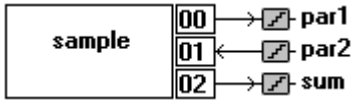
Exemple complet

Voici un exemple complet de programmation de la fonction "**sample**", qui effectue une simple addition analogique entière. Voici la fiche technique de la fonction :

nom:	SAMPLE
description:	Calcule une addition analogique entière
date de création:	1er juillet 1993
auteur:	CJ International
appel:	par1, par2: opérandes entiers
retour:	sum: résultat de l'addition

prototype:	sum = sample (par1, par2);

Voici la définition de l'interface de la fonction :



Voici le fichier de définitions "C" pour la fonction :

```
/* Fichier : GRUS0255.h - fonction "C" - Nom : sample */

/* définition des types standards ISaGRAF */

typedef long        T_BOO;
typedef long        T_ANA;
typedef float       T_REAL;
typedef long        T_TMR;
typedef char        *T_MSG;

/* définition de la structure des paramètres d'appel et de retour */

typedef struct {
  T_ANA _par1;    /* paramètre d'appel N°1 */
  T_ANA _par2;    /* paramètre d'appel N°2 */
  T_ANA _sum;     /* paramètre de retour */
} str_arg;

/* identificateurs utilisés pour accéder aux champs de la structure */

#define PAR1        (arg->_par1)
#define PAR2        (arg->_par2)
#define SUM         (arg->_sum)

/* fin de fichier */
```

Voici le fichier de code source "C" pour la fonction. Seules les lignes notées en gras ont été manuellement saisies par le programmeur.

```
/* Fichier : GRUS0255.c - fonction "C" - Nom : sample */

#include "tasy0def.h"       /* nécessaire pour les définitions de types */
#include "grus0255.h"       /* fichier de définitions associé */

/* service principal: calcule l'addition */

void USP_sample (arg)
str_arg *arg;
{
```

```
  SUM = PAR1 + PAR2;
}

/* déclaration pour le lien dynamique avec le noyau ISaGRAF */

UFP uspdef_sample (char *name)
{
  strcpy (name, "SAMPLE");
  return (USP_sample);
}

/* fin de fichier */
```

C.7.4 Blocs fonctionnels en "C"

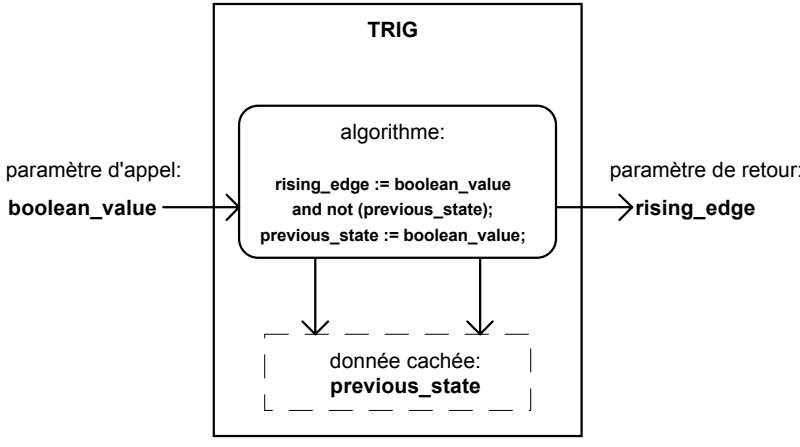
Les blocs fonctionnels en "C" associent des traitements et des données statiques. Ils sont le complément des fonctions "C", et permettent de traiter des objets ayant une durée de vie de plus d'un échantillon. Ils sont généralement mis en oeuvre pour étendre les possibilités offertes en standard par les langages **ST** et **FBD**. Contrairement aux fonctions, qui ne traitent que des valeurs, les blocs fonctionnels peuvent travailler sur des données statiques. Ceci signifie qu'un algorithme de bloc fonctionnel peut gérer l'évolution dans le temps de ces données.

Les blocs fonctionnels sont écrits en langage "C", compilés et liés avec le noyau ISaGRAF. Le nouveau noyau ainsi généré doit alors être installé dans l'automate cible, avant que les nouveaux blocs fonctionnels puissent être utilisés dans les applications ISaGRAF. Les nouveaux blocs fonctionnels ne peuvent pas être intégrés au simulateur d'ISaGRAF. Les applications ISaGRAF devront être simulées **avant** l'utilisation dans leurs programmes des nouveaux blocs.

Attention: Les blocs fonctionnels réalisent des opérations élémentaires **synchrones**, activées par le noyau exécutif ISaGRAF pendant le cycle de l'application. Le temps passé à exécuter un service d'activation ou d'interrogation d'un bloc fonctionnel est inclus dans le **temps de cycle** de l'application. Le développeur doit s'assurer qu'aucune opération bloquante n'est programmée dans un bloc fonctionnel, afin de préserver le mécanisme d'exécution du noyau ISaGRAF.

Déclaration des instances

Un bloc fonctionnel est un objet qui associe des traitements et des données. Voici l'exemple du bloc fonctionnel "**TRIG**" qui assure la détection des fronts montants d'un signal booléen :



La variable cachée "**previous_state**" est nécessaire pour le calcul du front. Cette variable doit être différenciée à chaque fois que le bloc fonctionnel "**R_TRIG**" est utilisé dans la même application. Les instances (copies) de blocs fonctionnels utilisés dans le langage ST doivent être déclarés dans le dictionnaire de l'application. Puisque le bloc inclut des données, chaque instance d'un bloc fonctionnel doit être identifiée par un nom unique. Le type de bloc fonctionnel est identifié avec le Gestionnaire de Librairies. Le nom des instances est donné avec l'outil d'édition du Dictionnaire. Les instances de blocs fonctionnels utilisées dans les diagrammes FBD n'ont pas besoin d'être déclarées, car l'éditeur FBD d'ISaGRAF déclare automatiquement une nouvelle instance à chaque fois qu'un bloc fonctionnel est inséré dans le diagramme. Les instances de blocs fonctionnels déclarées automatiquement par l'éditeur FBD sont toujours **LOCALES** au programme édité.

Ajouter un bloc fonctionnel à la librairie ISaGRAF

Le Gestionnaire de Librairie d'ISaGRAF doit être utilisé pour ajouter un nouveau bloc fonctionnel "C" à la librairie ISaGRAF. Avec l'atelier, utilisez la commande "**Nouveau**" du menu "**Fichiers**", quand la librairie des blocs fonctionnels est sélectionnée. Quand un nouveau bloc fonctionnel est créé, sa **fiche technique** doit être rédigée. La trame du code source "C" pour le nouveau bloc fonctionnel est automatiquement générée par le Gestionnaire de Librairie d'ISaGRAF. La commande "**Paramètres**" du menu "**Edition**" permet de définir les paramètres d'appel et de retour du nouveau bloc fonctionnel.

Utiliser un bloc fonctionnel dans un projet ISaGRAF

Tout nouveau bloc fonctionnel intégré au noyau ISaGRAF peut être utilisé dans les projets ISaGRAF. Les blocs fonctionnels peuvent être mis en oeuvre dans les programmes écrits en langage **ST** ou **FBD**.

L'appel d'un bloc fonctionnel depuis le langage **ST** respecte les conventions d'appel de blocs fonctionnels du langage. Les paramètres d'appel du bloc sont écrits entre parenthèses après le nom de l'instance, séparés entre eux par des virgules. Les paramètres de retour sont accédés séparément. Chaque paramètre de retour est identifié par un nom, qui combine le

nom de l'instance du bloc, et le nom logique du paramètre. Les deux parties du nom sont séparées par un point. Par exemple, le nom :

FBINSTNAME.parname

est utilisé pour représenter le paramètre de retour "**parname**", d'une instance de bloc nommée "**FBINSTNAME**".

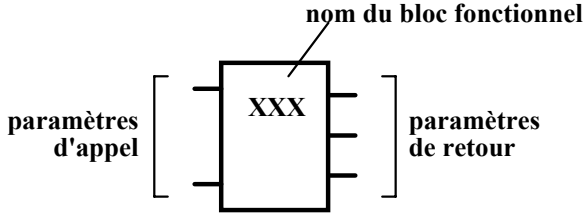
Les instances (copies) de blocs fonctionnels utilisés dans le langage ST doivent être déclarés dans le dictionnaire de l'application. Puisque le bloc inclut des données, chaque instance d'un bloc fonctionnel doit être identifiée par un nom unique. Voici un exemple de déclaration d'instances dans le dictionnaire :

instance:	TRIG1	type:	TRIG
	TRIG2		TRIG

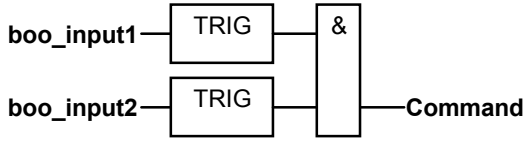
Et voici un exemple de programme ST utilisant les instances déclarées :

```
TRIG1 (boo_input1);
TRIG2 (boo_input2);
Command := (TRIG1.Q & TRIG2.Q);
```

Un programme **FBD** peut appeler tout bloc fonctionnel "C" de la librairie. Un bloc fonctionnel est représenté comme une boîte. Ses paramètres d'appel son connectés sur le bord gauche de la boîte. Les paramètres de retour sont connectés sur le bord droit de la boîte. Voici l'aspect standard d'un bloc fonctionnel dans un diagramme FBD:

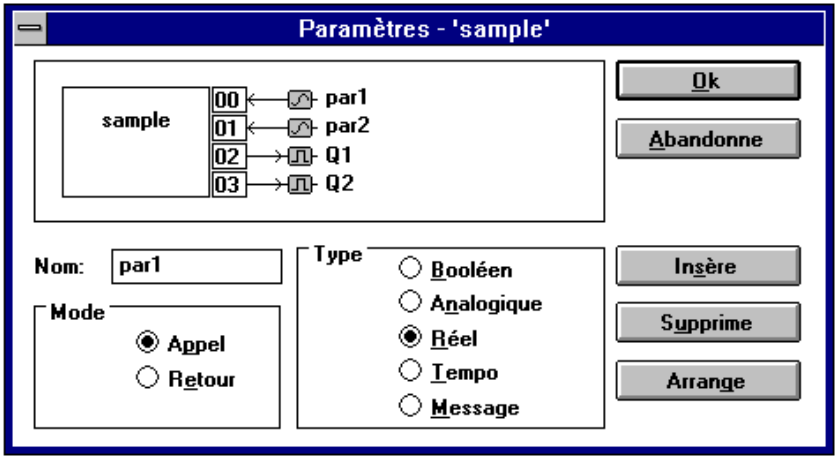


Les instances de blocs fonctionnels utilisées dans les diagrammes FBD n'ont pas besoin d'être déclarées, car l'éditeur FBD d'ISaGRAF déclare automatiquement une nouvelle instance à chaque fois qu'un bloc fonctionnel est inséré dans le diagramme. Les instances de blocs fonctionnels déclarées automatiquement par l'éditeur FBD sont toujours **LOCALES** au programme édité. Voici un exemple de diagramme FBD incluant des blocs fonctionnels :



Définition de l'interface "C" d'un bloc fonctionnel

La commande "**Paramètres**" du menu "**Edition**" du Gestionnaire de Librairies est utilisée pour définir les paramètres d'appel et de sortie d'un bloc fonctionnel. Un bloc fonctionnel peut avoir jusqu'à **32** paramètres, librement répartis en paramètres d'appel et de retour . Contrairement aux fonctions "C", un bloc fonctionnel peut avoir plusieurs paramètres de retour. La boîte de dialogue suivante est utilisée pour décrire les paramètres d'une fonction "C" :



La liste dans la partie supérieure de la fenêtre montre les paramètres du bloc fonctionnel, dans l'ordre défini par son prototype d'appel: d'abord les paramètres d'appel, ensuite les paramètres de retour. La partie inférieure de la fenêtre montre la description détaillée du paramètre sélectionné dans la liste :

- le nom du paramètre
- son mode (appel ou retour)
- le type du paramètre

Tous les types de données d'ISaGRAF peuvent être utilisés pour un paramètre : Booléen, Analogique entier ou réel, Temporisation ou Message. Les analogiques entiers et réels doivent être différenciés. Voici la correspondance entre les types ISaGRAF et les types du langage "C" :

BOOLEEN	unsigned long	mot de 32 bits non signé : 1=true / 0=false
ENTIER	long	mot signé de 32 bits
REEL	float	valeur flottante en simple précision (32 bits)
TEMPO	unsigned long	mot non signé de 32 bits (unité = 1ms)
MESSAGE	char *	chaîne de caractères

Quand un message est passé à une fonction "C", il ne peut pas contenir de caractère nul (code ASCII 00). La chaîne passée à la fonction "C" est toujours terminée par un caractère nul. Les paramètres de retour doivent être les derniers de la liste. Les règles suivantes doivent être respectées pour la nomenclature des paramètres :

- la longueur du nom ne peut pas excéder **16** caractères
- le premier caractère doit être une **lettre**
- les suivants doivent être des **lettres**, des **chiffres** ou le souligné
- les majuscules et les minuscules ne sont pas différenciées

Le même nom ne peut pas être utilisé pour plusieurs paramètres du même bloc fonctionnel. Un paramètre d'appel ne peut pas avoir le même nom qu'un paramètre de retour. Le même nom **peut** être utilisé pour des paramètres de blocs fonctionnels différents. Le nom d'un paramètre est utilisé pour identifier la donnée correspondante dans le code source "C" du bloc fonctionnel.

La commande "**Insère**" permet d'insérer un nouveau paramètre avant le paramètre sélectionné dans la liste. La commande "**Supprime**" efface le paramètre sélectionné. La commande "**Arrange**" trie automatiquement les paramètres, et positionne les paramètres de retour en fin de liste. Enfoncez le bouton "**Ok**" pour enregistrer la définition des paramètres et refermer la boîte de dialogue. Enfoncez le bouton "**Abandonne**" pour fermer la boîte de dialogue sans modifier la définition des paramètres.

Interface "C" d'un bloc fonctionnel

L'interface d'un bloc fonctionnel dépend de la définition de ses paramètres. Les paramètres d'appel sont passés au service implémenté en "C" à travers une structure. Cette structure est définie dans le fichier "**GRFB0nnn.H**", où "**nnn**" est le numéro logique du bloc fonctionnel dans la librairie. Les paramètres de retour sont identifiés par des numéros, également définis dans le fichier "**GRFB0nnn.h**". Voici un exemple d'interface "C", pour le bloc fonctionnel "**LIM_ALRM**" (alarmes de limites) :

```
/* interface de bloc fonctionnel - nom: sample */

/* types standard ISaGRAF */

typedef     long        T_BOO;
typedef     long        T_ANA;
typedef     float       T_REAL;
typedef     long        T_TMR;
typedef     char        *T_MSG;

/* structure des paramètres d'appel */

typedef struct {
  /* CALL  */  T_REAL      _par1;
  /* CALL  */  T_REAL      _par2;
} str_arg;

/* accès aux champs de la structure str_arg */

#define     PAR1        (arg->_par1)
#define     PAR2        (arg->_par2)

/* numérotation des paramètres de retour */
```

```
#define     FBLPNO_Q1   0
#define     FBLPNO_Q2   1
```

/ fin de fichier */*

La relation entre les types ISaGRAF et les types du langage "C" est donnée dans la table suivante. Chaque type ISaGRAF est défini comme un nouveau type du langage "C" dans le fichier de définitions du bloc fonctionnel.

booléen	**T_BOO**	**long** (32 bits)
entier	**T_ANA**	**long**
réel	**T_REAL**	**float** (32 bits - simple precision)
temporisation	**T_TMR**	**long**
message	**T_MSG**	**char *** (32 bits - pointeur de caractères)

Chaque champ de la structure "**str_arg**" correspond à un paramètre d'appel du bloc fonctionnel. Les paramètres apparaissent dans la structure selon l'ordre dans lequel ils ont été déclarés lors de la définition de l'interface du bloc. Un identificateur en majuscules est défini pour chaque paramètre d'appel. Il permet d'accéder directement à la valeur du paramètre dans la structure passée en argument dans le code "C". Les noms des paramètres sont ceux saisis lors de la description de l'interface du bloc fonctionnel avec le Gestionnaire de Librairie d'ISaGRAF.

L'ordre utilisé pour la numérotation des paramètres de retour est celui établi lors de la définition de l'interface du bloc. Le numéro logique du premier paramètre de retour est toujours **0**. Les identificateurs définis pour représenter ces numéros doivent être utilisés dans le source "C" (et non directement les numéros). Ceci assure que le fichier source "C" est facilement maintenable après une modification de l'interface du bloc fonctionnel.

Le fichier de définitions en "C" est mis à jour automatiquement par le Gestionnaire de Librairie d'ISaGRAF à chaque modification de l'interface du bloc fonctionnel. Ceci assure la cohérence entre le développement en "C" des blocs fonctionnels et leur utilisation dans les programmes des applications ISaGRAF.

Code source

L'implémentation en langage "C" d'un bloc fonctionnel est divisée en trois principaux services :

- ❑ service d'initialisation
- ❑ service d'activation - traitement des paramètres d'appel
- ❑ service de lecture - accès aux paramètres de retour

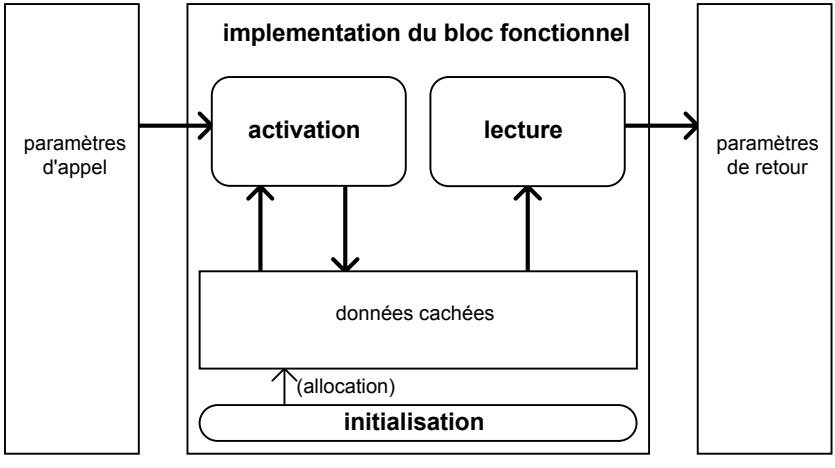
Le même code source est utilisé pour chacune des instances d'un même bloc, et n'est pas dupliqué en mémoire. Une structure de données statiques est associée à chaque instance. Ces données ne sont pas accessibles depuis les programmes ISaGRAF, et contiennent les "variables cachées" associées à l'instance.

Le "service d'activation" est appelé une fois pour chaque instance de chaque bloc utilisé, à chaque cycle d'exécution. Il traite les paramètres d'appel du bloc fonctionnel, et met à jour les

données cachées associées à l'instance. Ce service contient "l'algorithme principal" du bloc fonctionnel.

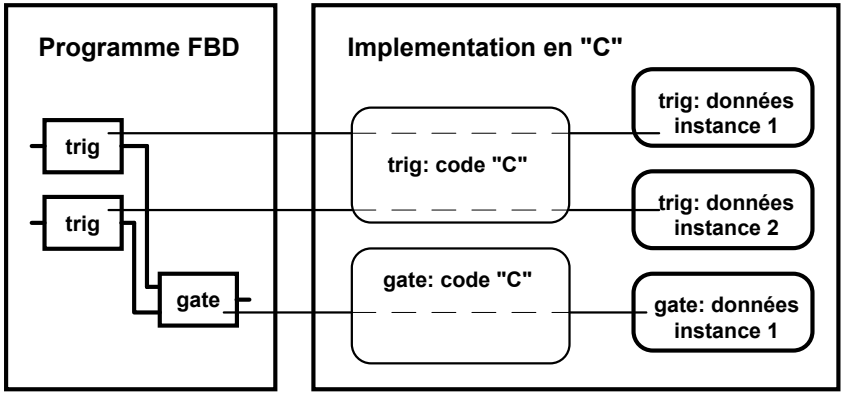
Le "service de lecture" est appelé par le noyau ISaGRAF pour récupérer la valeur courante d'un paramètre de retour d'une instance de bloc fonctionnel, à chaque fois que ce paramètre est référencé dans un programme ST ou FBD. Aucun traitement particulier ne doit être réalisé dans ce service. Il ne gère que le transfert des informations entre les données cachées de l'instance et l'application.

Schéma fonctionnel:



Données associées à une instance

Un bloc fonctionnel associe des traitements à des données statiques. Une structure de données est associée à chaque instance. Chaque fois qu'un bloc fonctionnel est utilisé dans un programme ST ou FBD, il correspond à une instance, et une structure de données associée. L'exemple suivant montre la correspondance entre les structures de données manipulées par le code "C" du bloc fonctionnel, et l'utilisation des instances de blocs dans un programme FBD :



La mémoire nécessaire pour stocker chaque structure de données est allouée par le système ISaGRAF, au lancement de l'application. Un pointeur vers la structure de l'instance concernée est passé en argument à chaque appel aux services d'activation et de lecture du bloc fonctionnel correspondant.

Le Gestionnaire de librairies d'ISaGRAF génère automatiquement la trame de définition du type de la structure dans le code source "C" du bloc fonctionnel. Ce type est toujours nommé "**str_data**". Le programmeur ne doit pas changer le nom de ce type, pour assurer la cohérence avec les en-têtes de services générés automatiquement par ISaGRAF. Les données cachées groupent généralement des variables internes de calcul et une image des paramètres de retour. Le service de "lecture" du bloc fonctionnel se contente de mettre en forme les données de cette structure pour fournir à l'application ISaGRAF la valeur d'un paramètre de retour. Le service de lecture pouvant être appelé plusieurs fois dans le même cycle, pour le même paramètre de la même instance, aucun autre traitement ne doit être réalisé qui modifie la valeur des variables cachées.

Le service d'initialisation

Le service "d'initialisation" d'un bloc fonctionnel est appelé par le système ISaGRAF quand l'application est lancée. Il permet au bloc fonctionnel d'indiquer au système ISaGRAF la taille de la mémoire devant être allouée pour une instance du bloc. Voici la programmation standard de ce servive :

```
word FBINIT_xxx (word hinstance)        /* "xxx" est le nom du bloc */
{
  return (sizeof (str_data));
}
```

L'argument "**hinstance**" est le numéro logique de l'instance concernée. Il est réservé aux traitements internes d'ISaGRAF, et ne doit pas être utilisé. Le service d'initialisation retourne le nombre d'octets de mémoire à allouer pour une instance du bloc fonctionnel. La taille de la mémoire allouée pour une instance ne doit pas excéder **64** Koctets. Aucune autre opération ne doit être effectuée dans ce service. Le code source "C" du service d'initialisation est automatiquement généré par le Gestionnaire de Librairies d'ISaGRAF quand le bloc fonctionnel est créé.

Le service d'activation

Le service "d'activation" est appelé à chaque cycle d'exécution, pour chaque instance de bloc fonctionnel utilisée dans l'application. Ce service prend en compte la valeur courante des paramètres d'appel, exécute l'algorithme principal du bloc fonctionnel, et met à jour les données internes cachées ainsi que l'image interne des paramètres de retour. Voici l'en-tête standard de ce service :

```
void FBACT_xxx (        /* "xxx" est le nom du bloc fonctionnel */
  word hinstance,       /* numéro logique de l'instance */
  str_data *data,       /* pointe vers les données de l'instance */
  str_arg *arg)         /* pointe vers la structure des paramètres d'appel */
{
  /* ... */
}
```

L'argument "**hinstance**" est le numéro logique de l'instance cocernée. Il est réservé aux traitements internes du noyau ISaGRAF, et ne doit pas être utilisé dans la programmation du service. L'argument "**data**" est un pointeur vers la structure de données contenant les "variables cachées" associées à l'instance concernée. L'argument "**arg**" est un pointeur vers une structure qui regroupe les valeurs courantes des paramètres d'appel du bloc fonctionnel. Le programmeur doit utiliser les identificateurs définis dans le fichier de définitions "C" associé pour accéder aux champs de la structure pointée par "**arg**".

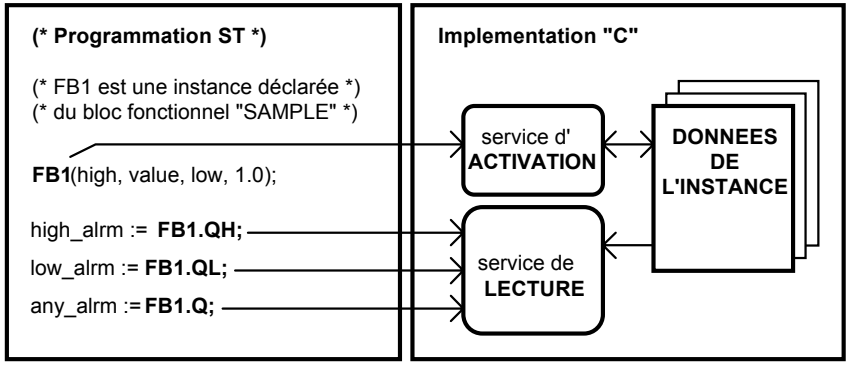
Typiquement, l'algorithme d'activation traite les paramètres d'appel du bloc (stockés dans la structure "**arg**"), et met à jour les champs de la structure "**data**". L'exemple suivant montre le service d'activation du bloc fonctionnel **R_TRIG** (détection de front montant) :

```
/* définitions écrites dans le fichier de définitions "C" associé */

typedef struct {            /* paramètres d'appel */
  T_BOO  _clk;              /* signal d'entrée */
} str_arg;

#define CLK                 (arg->_clk)

/* définition de la structure de données associée à une instance */

typedef
  struct {
    T_BOO prev_state;       /* état précédent du signal d'entrée */
    T_BOO edge_detect;      /* front: détection du front montant */
  } str_data;

/* service d'activation */

void FBACT_r_trig (word hinstance, str_data *data, str_arg *arg)
{
  data->edge_detect = (T_BOO)(CLK && !data->prev_state);
  data->prev = CLK;  /* paramètre d'appel */
}
```

L'en-tête du service d'activation est automatiquement généré par le Gestionnaire de Librairies d'ISaGRAF dans le fichier de code source "C" quand le bloc fonctionnel est créé.

Accès aux paramètres de retour

Le service de "lecture" est appelé à chaque fois qu'un paramètre de retour d'une instance de bloc fonctionnel est référencée dans un programme ST ou FBD. Un appel à ce service correspond à la lecture d'**un seul** paramètre de retour. L'exemple suivant montre l'appel aux services de "lecture" pendant l'exécution d'un programme ST :



Puisque le service de lecture peut être appelé plusieurs fois dans le même cycle d'exécution, pour accéder au même paramètre de sortie de la même instance de bloc, il ne doit effectuer aucun calcul qui modifie la valeur des données associées à l'instance. Il n'effectue qu'un transfert entre les données cachées de l'instance et l'application ISaGRAF. Voici la trame standard d'un service de lecture :

```
/* opération de changement de type pour la copie du paramètre */

#define BOO_VALUE       ((T_BOO *)value)
#define ANA_VALUE       ((T_ANA *)value)
#define REAL_VALUE      ((T_REAL *)value)
#define TMR_VALUE       ((T_TMR *)value)
#define MSG_VALUE       ((T_MSG *)value)

/* service de lecture: un seul paramètre lu lors d'un appel */

void FBREAD_xxx (           /* "xxx" est le nom du bloc fonctionnel */
  word hinstance,           /* numéro logique de l'instance */
  str_data *data,           /* pointeur vers les données de l'instance */
  word parno,               /* numéro du paramètre requis */
  void *value)              /* adresse du buffer où la valeur du paramètre */
                            /* doit être copiée */
{
  switch (parno) {
     case FBLPNO_XX: /* ... */ break;
```

```
    case FBLPNO_YY: /* ... */ break;
    /* .... */
  }
}
```

L'argument "**hinstance**" est le numéro logique de l'instance concernée. Il est réservé aux traitements internes du noyau ISaGRAF, et ne doit pas être utilisé dans la programmation du service. L'argument "**data**" est un pointeur vers la structure de données contenant les "variables cachées" associées à l'instance concernée.

L'argument "**parno**" est le numéro logique du paramètre demandé. Utilisez les identificateurs définis dans le fichier de définitions "C" associé pour distinguer les paramètres de retour. Ces identificateurs sont toujours préfixés par le label "**FBLPNO_**". L'argument "**value**" est l'adresse du buffer interne du noyau ISaGRAF où la valeur courante du paramètre demandé doit être copiée. Le type de donnée pointée par cet argument dépend du type déclaré pour le paramètre considéré. La table suivante donne la correspondance entre le type ISaGRAF du paramètre et le type "C" de la donnée pointée par l'argument "**value**" :

booléen	**long**	mot non signé de 32 bits (0=false / 1=true)
entier	**long**	mot signé de 32 bits
réel	**float**	valeur flottante simple précision (32 bits)
temporisation	**long**	mot non signé de 32 bits (unité = 1ms)
message	**char ***	tableau de caractères

Les macros suivantes sont utilisées pour accéder au buffer interne d'ISaGRAF, en accord avec le type du paramètre accédé :

```
#define     BOO_VALUE     ((T_BOO *)value)
#define     ANA_VALUE     ((T_ANA *)value)
#define     REAL_VALUE    ((T_REAL *)value)
#define     TMR_VALUE     ((T_TMR *)value)
#define     MSG_VALUE     ((T_MSG *)value)
```

Voici les opérations de copie typiquement programmées pour la copie de la valeur du paramètre de retour accédé :

```
/* pour un paramètre booléen: */
  *BOO_VALUE = parameter_value;
/* pour un paramètre analogique entier: */
  *ANA_VALUE = parameter_value;
/* pour un paramètre analogique réel: */
  *REAL_VALUE = parameter_value;
/* pour un paramètre temporisation: */
  *TMR_VALUE = parameter_value;
/* pour un paramètre message: */
  strcpy (*MSG_VALUE, parameter_value);
```

L'entête du service de lecture est automatiquement générée dans le fichier de code source "C" par le Gestionnaire de Librairies d'ISaGRAF quand le bloc fonctionnel est créé.

Exemple de code source "C"

Voici la trame standard complète d'un fichier source "C" pour la programmation d'un bloc fonctionnel :

```
/* bloc fonctionnel (xxx est le nom du bloc en librairie) */

#include <tasy0def.h>
#include <grfb0nnn.h>          /* nnn est le numéro du bloc en librairie */

/* structure de données associée à une instance du bloc */

typedef
  struct {
      /* définition des champs */
  } str_data;

/* service d'initialisation: rend la taille mémoire à allouer pour les données associées à une
instance du bloc fonctionnel */

word FBINIT_xxx (word hinstance)
{
  return (sizeof (str_data));
}

/* service d'activation: traite les paramètres d'appel */

void FBACT_xxx (word hinstance, str_data *data, str_arg *arg)
{
  /* ... */
}

/* changement de type: copie des paramètres de retour */

#define BOO_VALUE   ((T_BOO *)value)
#define ANA_VALUE   ((T_ANA *)value)
#define REAL_VALUE  ((T_REAL *)value)
#define TMR_VALUE   ((T_TMR *)value)
#define MSG_VALUE   ((T_MSG *)value)

/* lecture des paramètres de retour: appelé pour chaque paramètre */

void FBREAD_xxx      (word hinstance, str_data *data,
                      word parno, void *value)
{
 switch(parno)
 {
  case FBLPNO_XX: *???_VALUE = ...; break;
  case FBLPNO_YY: *???_VALUE = ...; break;
  ....
 }
}
```

```
/* Le service suivant est utilisé pour le lien dynamique entre les services implémentés et
l'application ISaGRAF. Le lien est effectué au lancement de l'application, à l'aide du nom du
bloc fonctionnel. Le code de ce service est complètement généré par le  Gestionnaire de
Librairie d'ISaGRAF. Les types "?BP" sont définis dans le fichier "grta0def.h", comme des
pointeurs vers des fonctions */

ABP fbldef_xxx (char *name, IBP *initproc, RBP *readproc)
 {
   strcpy (name, "XXX");
   *initproc = (IBP)FBINIT_xxx;
   *readproc = (RBP)FBREAD_xxx;
   return ((ABP)FBACT_xxx);
}

/* fin de fichier */
```

Le fichier "**TASY0DEF.H**" regroupe les définitions communes à tous les fichiers "source" du noyau exécutif ISaGRAF. Il doit impérativement être inclus pour assurer l'indépendance par rapport au système d'exploitation de la cible. Il contient entre autre la définition des types **?FP**, utilisés pour la fonction de déclaration.

Liens entre les projets et l'implémentation en "C"

Le lien logique entre l'implémentation en "C" d'un bloc fonctionnel et son utilisation dans les projets ISaGRAF est réalisé par le nom du bloc fonctionnel. Un service de "déclaration" est ajouté au code source "C" du bloc. Ce service sera appelé une seule fois au lancement de l'application, pour réaliser un lien dynamique entre le noyau exécutif ISaGRAF et les services implémentés. Voici le format standard d'un tel service de déclaration:

```
ABP fbldef_xxx (char *name, IBP *initproc, RBP *readproc)
 {
   strcpy (name, "XXX");                /* nom du bloc fonctionnel */
   *initproc = (IBP)FBINIT_xxx;         /* service d'initialisation */
   *readproc = (RBP)FBREAD_xxx;         /* service de lecture */
   return ((ABP)FBACT_xxx);             /* service d'activation */
}
/* xxx est le nom du bloc fonctionnel */
```

Le nom du bloc fonctionnel, utilisé dans l'instruction **strcpy** doit être écrit en **majuscules**. Il doit être écrit en minuscules dans le nom des services implémentés.

L'utilisation des préfixes "**FBL_**" et "**fbldef_**" pour les services implémentés permet de créer des blocs fonctionnels portant le même nom qu'un mot clé du langage "C", ou qu'une fonction déjà définie dans les librairies du "C" et d'ISaGRAF. Aucune autre opération ne doit être effectuée dans le service de déclaration.

Le service de déclaration sera appelé pour tous les blocs fonctionnels intégrés au noyau, même si ils ne sont pas utilisés dans l'application ISaGRAF. Le noyau exécutif génèrera une erreur fatale si un bloc fonctionnel non intégré est utilisé dans une application.

Avant l'édition de liens du noyau avec les nouveaux blocs fonctionnels, le programmeur doit mettre à jour le fichier source "C" nommé "**GRFB0LIB.C**", et compléter la liste des fichiers pour la construction des librairies. Le fichier "**GRFB0LIB.C**" ne contient qu'un tableau des services de déclaration. Ce tableau sera lu pendant la phase d'initialisation du noyau exécutif ISaGRAF, pour réaliser le lien dynamique avec les blocs fonctionnels implémentés. Voici un exemple du contenu de ce fichier :

```
/* Fichier: grfb0lib.c blocs fonctionnels implémentés */

#include <tasy0def.h>

extern ABP fbldef_fb1();
extern ABP fbldef_fb2();

FBL_LIST FBLDEF[ ] = {
  fbldef_fb1,
  fbldef_fb2,
NULL };

/* fin de fichier */
```

Le tableau **FBLDEF** doit être terminé par un pointeur NULL. Des problèmes sévères peuvent survenir à l'exécution si cette condition n'est pas réalisée. Les blocs fonctionnels non inclus dans le tableau **FBLDEF** ne seront pas reconnus par le noyau, même si leur code est pris en compte dans l'édition de liens.

L'écriture de ce fichier permet la construction d'un nouveau noyau, intégrant les nouveaux blocs fonctionnels. Il est également possible de créer un noyau dédié à un projet, en n'insérant dans le tableau **FBLDEF** que les blocs utilisés dans ce projet. Le fichier "**GRFB0LIB.C**" est automatiquement généré par le Générateur de Code d'ISaGRAF, quand le code d'une application est fabriqué. Le fichier généré est placé dans le répertoire du projet. Il ne groupe que les blocs fonctionnels référencés dans les sources de l'application.

Un autre fichier, nommé "**GRFB0LIB.68K**" est automatiquement généré par le Générateur de Code d'ISaGRAF, quand le code de l'application est construit. Ce fichier a exactement la même signification que le fichier "**GRFB0LIB.C**", mais il utilise une convention d'écriture en assembleur pour la déclaration du tableau **FBLDEF**. Il est placé dans le répertoire du projet. Il ne groupe que les blocs fonctionnels référencés dans les sources de l'application. Voici un exemple d'un tel fichier :

```
#include <tasy0def.h>

extern ABP fbldef_fb1 ();
extern ABP fbldef_fb2 ();

_asm ("
FBLDEF:
      dc.l     fbldef_fb1
      dc.l     fbldef_rb2
      dc.l     0
");
```

/* *fin de fichier* */

Pour plus d'information sur ce fichier, reportez-vous au fichier "**Readme**" de la disquette contenant le logiciel cible ISaGRAF.

Limites

La librairie ISaGRAF peut contenir jusqu'à **255** blocs fonctionnels "C". Tout type d'opération peut être effectué dans un bloc fonctionnel. Tout bloc fonctionnel peut être instancié (copié) jusqu'à **255** fois dans le même projet.

Il convient de se rappeler que ces blocs fonctionnels sont activés de façon **synchrone**, et que l'exécution des blocs a une influence directe sur le temps de cycle de l'application.

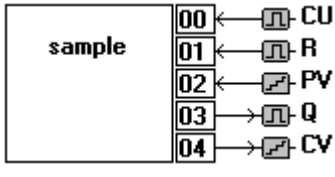
Exemple complet

Voici, à titre d'exemple, la programmation complète du bloc fonctionnel "**sample**", qui met en oeuvre un compteur incrémental. Voici la fiche technique de ce bloc fonctionnel :

nom:	SAMPLE
description:	Compteur incrémental
date de création:	01 Février 1994
auteur:	CJ international
appel:	CU : entrée de comptage R : commande de remise à zéro PV : valeur maximum programmée
retour:	Q : détection du maximum CV : résultat courant du comptage

prototype:	SAMPLE (count, reset_command, maximum_value);
	max_detect := SAMPLE.Q;
	count_result := SAMPLE.CV;

Voici la définition de l'interface du bloc fonctionnel :



Voici le fichier de définitions "C" pour le bloc fonctionnel :

/* *interface de bloc fonctionnel _ nom: SAMPLE* */

/* *définition des types standard ISaGRAF* */

```
typedef long            T_BOO;
typedef long            T_ANA;
typedef float           T_REAL;
typedef long            T_TMR;
typedef char            *T_MSG;

/* définition de la structure groupant les paramètres d'appel */

typedef struct {
  T_BOO  _cu;
  T_BOO  _r;
  T_ANA  _pv;
} str_arg;

/* identificateurs pour l'accès direct aux paramètres d'appel */

#define CU              (arg->_cu)
#define R               (arg->_r)
#define PV              (arg->_pv)

/* numérotation des paramètres de retour */

#define FBLPNO_Q    0
#define FBLPNO_CV   1

/* fin de fichier */
```

Voici le fichier de code source "C" du bloc fonctionnel. Seules les lignes montrées en gras ont été manuellement saisies par le programmeur.

```
/* bloc fonctionnel - nom: SAMPLE */

#include <tasy0def.h>       /* nécessaire pour les définitions de type */
#include <grfb0255.h>       /* définition "C" de l'interface */

/* définition de la structure de données associée à une instance */

typedef
  struct {
    T_BOO overflow;         /* vrai si valeur maximum détectée */
    T_ANA value;            /* résultat courant du comptage */
  } str_data;

/* service d'initialisation: demande l'allocation de la mémoire pour les données cachées */

word FBINIT_sample (word hinstance)
{
  return (sizeof (str_data));
}

/* service d'activation: algorithme de comptage */
```

```
void FBACT_sample (word hinstance, str_data *data, str_arg *arg)
{
  if (R) data->value = 0;
  else if (CU && data->value < PV) (data->value)++;
  data->overflow = (data->value >= PV) ? (T_BOO)1 : (T_BOO)0;
}

/* changement de type pour la copie des paramètres de retour */

#define BOO_VALUE       ((T_BOO *)value)
#define ANA_VALUE       ((T_ANA *)value)
#define REAL_VALUE      ((T_REAL *)value)
#define TMR_VALUE       ((T_TMR *)value)
#define MSG_VALUE       ((T_MSG *)value)

/* service de lecture des paramètres de retour */

void FBREAD_sample      (word hinstance, str_data *data,
                         word parno, void *value)
{
  switch (parno) {
    case FBLPNO_Q   : *BOO_VALUE = data->overflow; break;
    case FBLPNO_CV : *ANA_VALUE = data->value; break;
  }
}

/* déclaration pour le lien dynamique avec le noyau ISaGRAF */

ABP fbldef_sample (char *name, IBP *initproc, RBP *readproc)
 {
  strcpy (name, "SAMPLE");
  *initproc = (IBP)FBINIT_sample;
  *readproc = (RBP)FBREAD_sample;
  return ((ABP)FBACT_sample);
}

/* fin de fichier */
```

C.7.5 Techniques de compilation et d'intégration

L'atelier ISaGRAF n'inclut aucun outil de compilation "C" ou d'édition de liens. Toutefois, ce chapître présente les principales techniques qui peuvent être mises en oeuvre facilement pour utiliser les fichier générés par le Gestionnaire de Librairie d'ISaGRAF, et les prendre en compte pour la compilation "C" et l'édition de liens du noyau.

Fichiers de code "C"

Les fichiers de code source "C" des procédures et des fonctions de conversions sont placés par le Gestionnaire de Librairie d'ISaGRAF dans les répertoires **ISAWIN\LIB\DEFS** et **ISAWIN\LIB\SRC** (ou dans les répertoires spécifiés par les entrées **IsaSrc** et **IsaDefs**). Le

nom d'un fichier de code source est construit à partir du numéro de la procédure ou de la fonction de conversion. Ce numéro est un emplacement dans la librairie ISaGRAF. Voici les noms utilisés :

\isawin\lib\defs\tacn0def.h	fichier de définition standard pour toutes les fonctions de conversion
\isawin\lib\src\grcn0nnn.h	fichier source d'une conversion
\isawin\lib\defs\grus0nnn.h	fichier de définitions d'une fonction
\isawin\lib\src\grus0nnn.c	fichier source d'une fonction
\isawin\lib\defs\grfb0nnn.h	fichier de définitions d'un bloc fonctionnel
\isawin\lib\src\grfb0nnn.c	fichier source d'un bloc fonctionnel

(**nnn** est le numéro de la conversion, fonction ou bloc fonctionnel)

Attention: Si vous renommez ou copier un élément avec le Gestionnaire de Librairie d'ISaGRAF, les textes et lignes de programmation associés ne seront pas automatiquement actualisés avec le nouveau nom ou numéro de l'élément. Vous devrez donc manuellement corriger ces informations dans le fichier de code source "C".

Le fichier **\ISAWIN\LIB\USPNUMS** donne la relation entre les noms et les numéros pour toutes les fonctions de la librairie. Il est constamment tenu à jour par le Gestionnaire de Librairie d'ISaGRAF. Voici un exemple de ce fichier:

```
1     funct_A
10    funct_B
16    funct_C
```

Le fichier **\ISAWIN\LIB\FBLNUMS** donne la relation entre les noms et les numéros pour tous les blocs fonctionnels de la librairie. Il est constamment tenu à jour par le Gestionnaire de Librairie d'ISaGRAF. Voici un exemple de ce fichier:

```
1     fbl_A
2     fbl_B
```

Le fichier **\ISAWIN\LIB\CNVNUMS** donne la relation entre les noms et les numéros pour toutes les fonctions de conversions de la librairie. Il est constamment tenu à jour par le Gestionnaire de Librairie d'ISaGRAF. Voici un extrait de ce fichier pour les conversions de la librairie standard :

```
0     SCALE
1     BCD
```

Ces fichiers sont mis à jour automatiquement à chaque fois qu'un élément (conversion, fonction ou bloc fonctionnel) est créé, renommé, copié ou supprimé.

Le Générateur de Code d'ISaGRAF génère automatiquement les fichiers suivants quand le code d'une application est construit :

\isawin\apl\ppp\GRCN0LIB.C	Tableau de déclaration des conversions référencées dans le projet.
\isawin\apl\ppp\GRUS0LIB.C	Tableau de déclaration des fonctions référencées dans le projet.

\isawin\apl\ppp\GRFB0LIB.C	Tableau de déclaration des blocs fonctionnels référencés dans le projet.

(**ppp** est le nom du projet ISaGRAF)

Ces fichiers peuvent être repris pour les opérations d'édition de lien d'un noyau ISaGRAF dédié à un projet, qui n'intègre que les conversions, fonctions et blocs fonctionnels requis par ce projet.

Transfert des sources sur un système natif

Les fichiers de code source et de définitions en "C" générés par le Gestionnaire de Librairie d'ISaGRAF peuvent être transférés sur un système cible ISaGRAF, s'il supporte un système de développement natif. Pour ceci, utilisez le programme **TERMINAL** livré avec Windows. Pour certaines configurations de système cible, d'autres outils de transfert peuvent être fournis par CJ International.

Quand les fichiers sources sont gérés sur le système cible, les fichiers de définitions mis à jour doivent être transférés à nouveau, à chaque fois que l'interface d'une fonction ou dun bloc fonctionnel a été modifié avec le Gestionnaire de Librairie d'ISaGRAF.

Utilisation d'un compilateur croisé

Les fichiers sources peuvent également être directement gérés sur le PC où est installé l'atelier ISaGRAF, si la cible est aussi un PC, ou si vous disposez d'une chaîne de compilation croisée, fonctionnant sous MS-DOS et dédiée à votre système cible.

Dans ce cas, le programmeur peut utiliser les outils de l'atelier ISaGRAF pour compléter et mettre à jour les fichiers de code source des conversions, fonctions ou blocs fonctionnels. Le fichier de commandes "**OEM.BAT**", réservé à l'utilisateur, peut être utilisé pour lancer le compilateur et l'éditeur de liens.

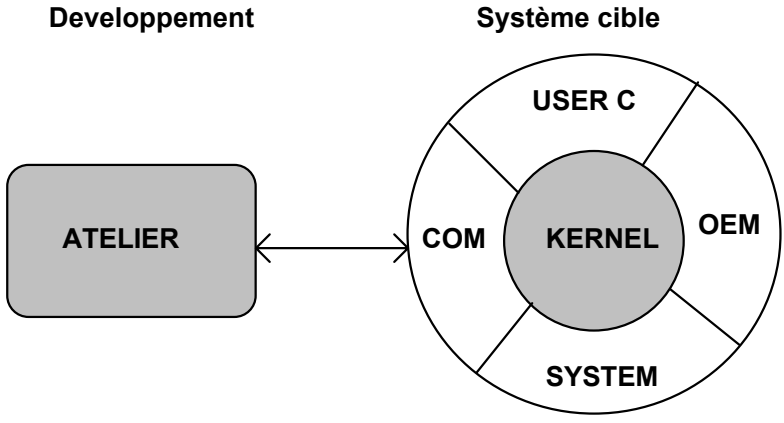
Quand les conversions, fonctions et blocs fonctionnels sont compilés sur le PC, il suffit de télécharger le nouveau noyau ISaGRAF dans la cible avant de lancer les applications qui utilisent les nouveaux éléments intégrés. Bien sûr, si la machine cible est un PC MS-DOS, le nouveau noyau ISaGRAF peut être transporté à l'aide d'une simple disquette.

Edition de liens avec le noyau ISaGRAF

La disquette du logiciel cible ISaGRAF contient de nombreaux utilitaires pour compiler les éléments développés, et les lier au noyau ISaGRAF. Le fichier de commandes "**ISACC**" est utilisé pour la compilation d'un fichier "C". L'utilisateur peut librement modifier ce fichier si les fichiers sources et relogeables sont installés dans d'autres répertoires. Le logiciel cible ISaGRAF est une application temps réel exécutée sur votre sytème cible (carte ou ordinateur industriel). Deux implementations sont disponibles:

- monotâche: toutes les fonctions sont exécutées par le même programme
- multitâches: une tâche séparée gère la communication

Dans tous les cas, les composants "C" développés sont groupés dans les mêmes librairies : pour le programmeur "C", aucune différence n'est faite entre les versions monotâche et multitâche. Dans la version monotâche, les composants "C" sont intégrés dans la tâche **isa**, alors que dans la version multitâche, ils sont intégrés dans la tâche **isaker**.

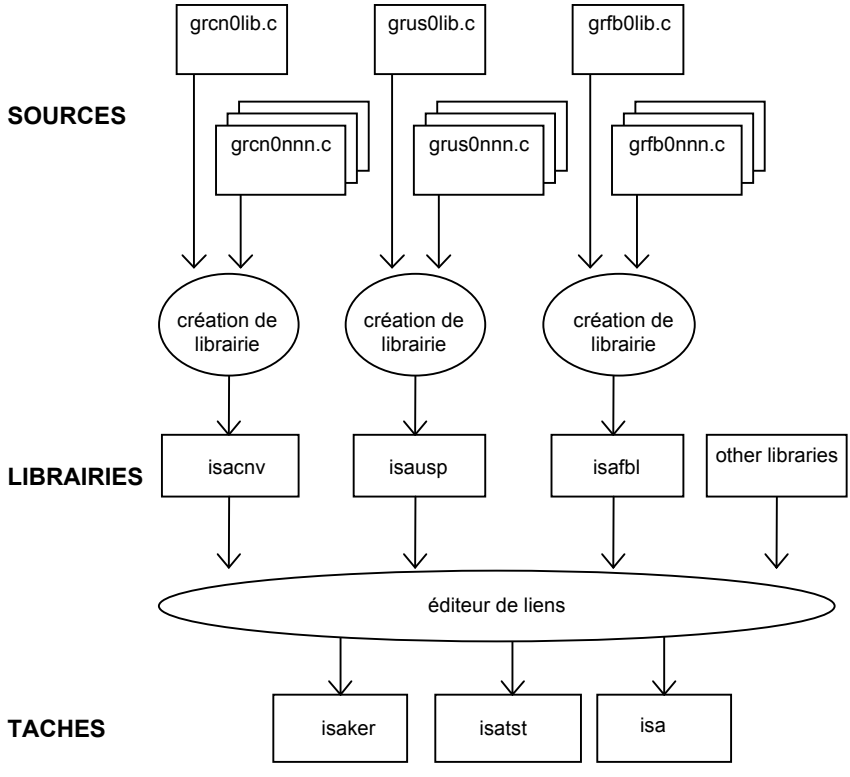


Le coeur d'ISaGRAF (KERNEL) est indépendant du matériel utilisé. Il exécute les instructions des langages IEC et gère sa propre base de données.

La première étape, lors du lien avec le noyau ISaGRAF, est de construire les librairies regroupant les conversions, les fonctions et les blocs fonctionnels intégrés :

librairie	contenu
ISAUSP	- GRUS0LIB (tableau des services de déclaration) - code objet des fonctions intégrées
ISAFBL	- GRFB0LIB (tableau des services de déclaration) - code objet des blocs fonctionnels intégrés
ISACNV	- GRCN0LIB (tableau des services de déclaration) - code objet des conversions intégrées

Le programmeur doit ensuite réaliser l'édition de liens entre ces librairies et les autres composants du logiciel cible ISaGRAF. Les différentes phases de l'intégration des composants "C" sont décrites sur le schéma suivant :



Voici la liste exacte des modules objets et des librairies qui interviennent dans l'édition de liens :

Module objet:	**tast0mai**	
Module objet:	**tast0com**	
Librairie noyau:	**isaker**	
Librairie noyau:	**isaoem**	
Librairie utilisateur:	**isausp**	fonctions
Librairie utilisateur:	**isacnv**	conversions
Librairie utilisateur:	**isafbl**	blocs fonctionnels
Librairie noyau:	**isasys**	
Librairies système:	(référez-vous au manuel du compilateur utilisé)	

Le programmeur doit respecter exactement l'ordre décrit ci-dessus pour les modules objets et les librairies. Les fichiers de code objets et de librairies sont suffixés par les extensions standard ("**.lib**", "**.obj**", "**.l**", "**.r**"...) définies par le système cible.

Options de compilation et d'édition de liens

Les options à spécifier dans les lignes de commandes du compilateur et de l'éditeur de lien dépendent du système cible, et du type d'opérations réalisées dans les conversions, fonctions et blocs fonctionnels. Certaines opérations nécessiteront d'autres librairies du système (mathématiques, graphiques...) pendant l'édition de lien.

Tous les fichiers de code source "C" du noyau ISaGRAF ont été compilés dans le modèle de mémoire **LARGE**. Le même modèle doit être sélectionné pour la compilation des conversions, fonctions et blocs fonctionnels.

Une constante spéciale doit être définie dans la ligne de commande lors de la compilation des procédures et des fonctions de conversion. Elle identifie le système et le processeur de la cible, et permet d'écrire des conversions, fonctions et blocs fonctionnels indépendants du système cible. Voici le nom de ces constantes :

DOS.....pour les systèmes de type PC (processeur INTEL)
ISAWNT .....pour les systèmes de type Windows NT (processeur INTEL)
OS9 .....pour les systèmes de type OS9 (processeur MOTOROLA)
VxWorks .....pour les systèmes de type VxWorks (processeur MOTOROLA)

Le fichier de commandes **ISACC** contenu dans la disquette de livraison du logiciel cible ISaGRAF montre comment définir la constante adaptée à votre système.

Compilateurs supportés

Les compilateurs suivants peuvent être utilisés pour le développement des procédures et des fonctions de conversion, et leur lien avec les librairies du noyau ISaGRAF :

Microsoft MSC 7.00 compiler.....pour les systèmes de type PC
Microsoft MSVC 4.00 compiler.....pour le système Windows NT
Microware ULTRA-C compiler.....pour le système OS9
Tornado 1.0; GNU Toolkit 2.6.....pour le système VxWorks

Contactez CJ International pour l'utilisation d'autres compilateurs.

Résumé

Voici le résumé des opérations à effectuer lors du développement d'un nouvel élément (conversion, fonction ou bloc fonctionnel).

⇨**1.** Avec le Gestionnaire de Librariries d'ISaGRAF, créez le nouvel élément: donnez-lui un nom et un commentaire. La trame du code source "C" est automatiquement générée.

⇨**2.** Avec le Gestionnaire de Librariries d'ISaGRAF, décrivez l'interface de l'élément (paramètres d'appel et de retour), s'il s'agit d'une fonction ou d'un bloc fonctionnel. Le fichier des définitions "C" est automatiquement généré.

⇨**3.** Avec le Gestionnaire de Librariries d'ISaGRAF, saisissez le texte de la fiche technique (guide d'utilisation) pour cet élément.

⇨**4.** Avec le Gestionnaire de Librariries d'ISaGRAF, complétez le fichier de code source "C" pour la fonction, de la conversion ou du bloc fonctionnel.

⇨**5.** Selectionnez l'option "**Afficher les numéros**" du Gestionnaire de Librairies, afin de connaître le numéro utilisé dans les noms des fichiers de code "C" associés à l'élément.

⇨**6.** Compilez le fichier source avec un compilateur "C" et corrigez les éventuelles erreurs de syntaxe. Le fichier de commandes "**ISACC**" est généralement utilisé.

⇨**7.** Insérez le nom du service de déclaration associé à l'élément dans le fichier "**GR??0LIB.C**" qui contient le tableau des éléments à intégrer.

⇨**8.** Compilez le fichier "**GR??0LIB.C**".

⇨**9.** Insérez le nom du fichier objet dans la liste utilisée pour la construction de la librairie correspondante.

⇨**10.** Lancez l'utilitaire de construction de la librairie. Lancez l'éditeur de liens.

⇨**11.** Installez le nouveau noyau généré sur le système cible.

⇨**12.** Ecrivez une application de test pour la validation et la maintenance de l'élément développé.

C.8 La liaison Modbus

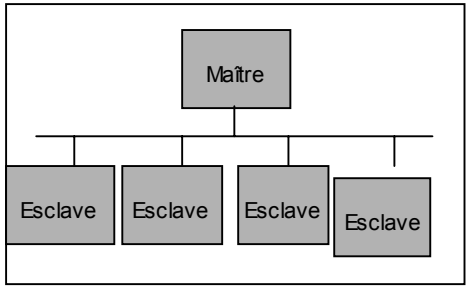
Une fois que l'application est complètement développée et testée, elle peut être connectée à un système de visualisation de procédé.

ISaGRAF est un système ouvert disposant d'une grande variété de possibilités de mise en réseau.
Le réseau industriel le plus simple est le protocole standard MODBUS/MODICON qui est disponible sur la plupart des systèmes de visualisation de procédé et qui ne nécessite qu'une liaison série (RS232, RS485, Current Loop).

Le protocole de communication du debugger ISaGRAF est compatible avec MODBUS permettant l'accès de lecture et d'écriture aux variables à partir d'un maître Modbus.

C.8.1 Réseau et protocole MODBUS

Un réseau Modbus se compose d'une seule station maître (habituellement un système de visualisation de procédé) et une ou plusieurs stations esclaves (habituellement des PLCs).



Le maître envoie une requête à la fois à un esclave (en utilisant son numéro d'esclave) et attend la réponse de l'esclave avant d'envoyer la requête suivante. Les esclaves non-concernés ne répondent pas.

Chaque bloc contient un numéro d'esclave, un numéro de requête et les données correspondantes, et un Checksum 16 bits (CRC).

Si aucune réponse n'est reçue aprés une durée de Time-Out, la requête peut être renouvellée plusieurs fois avant que le maître ne déclare la liaison comme "déconnectée".
La valeur du Time-Out et le nombre d'essais seront définis sur la station maître pour répondre aux besoin des esclaves (suivant l'application, ets...).

Si le traitement de la requête génère une erreur, l'esclave peut émettre un message d'erreur au lieu du bloc de réponse attendu.

Modbus est un protocole Modicon mais pas un standard international. Il existe de nombreuses implémentations de protocoles compatibles Modbus, avec des nombreuses possibilités, par exemple:
– Liste de codes fonctions implémentés
– Mapping d'adresses
– RTU (codes binaires) ou protocole ASCII
etc...

C.8.2 Implémentation d'ISaGRAF

Accès aux variables de l'application

La liaison de communication ISaGRAF reconnaît cinq codes de fonction Modbus:

1	lire N bits
3	lire N mots
5	écrire N bits
6	écrire 1 mot
16	écrire N mots

Il est possible d'accéder aux variables de l'application ISaGRAF via leurs 'adresses réseaux' si elles ont été définies dans le dictionnaire de l'atelier, bien sûr. Ces variables peuvent être:
- Variables booléennes ou analogiques
- Variables internes, d'entrée ou de sortie
- Variables locales ou globales.

Pour écrire une variable booléenne les fonctions 5, 6 ou 16 peuvent être utilisées. Une valeur d'écriture TRUE est toute valeur différente de zéro.

Pour lire une variable booléenne les fonctions 1 ou 3 peuvent être utilisées. Si la fonction 1 est utilisée les valeurs sont récupérées dans un champ bit. Si la fonction 3 est utilisée les valeurs sont récupérées en octets (une valeur TRUE correspond à 0xFFFF).

Pour écrire une variable analogique les fonctions 6 ou 16 peuvent être utilisées. La valeur est un entier de 16 bits compris entre -32768 et +32767 (les variables de la cible ISaGRAF ont 32 bits).

Pour lire une variable analogique on utilise la fonction 3. La valeur lue est un entier de 16 bits compris entre -32768 et +32767. Côté cible les variables analogiques ont 32 bits, alors une valeur sur la cible qui dépasse la limite de 16 bits (positive ou negative) sera lue avec la valeur maximale de 16 bits (positive ou negative).
Il n'est pas possible d'accéder aux variables réelles avec une requête Modbus.

Attention:
L'implémentation ISaGRAF ne gère pas les codes d'erreurs tels que "Adresse Modbus inconnue".

Notations:

slv	numéro d'esclave
nbw	nombre de mots
nbb	nombre d'octets
nbi	nombre de bits
addH	adresse réseau (octet de poids fort)
addL	adresse réseau (octet de poids faible)
vH	valeur (octet de poids fort)
vL	valeur (octet de poids faible)
V	valeur octet
bfd	champ de bits (nbb octets)
crcH	checksum (octet de poids fort)
crcL	checksum (octet de poids faible)

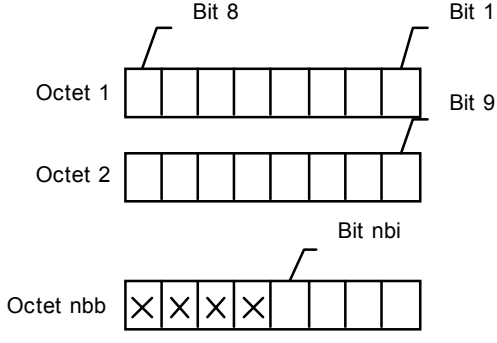
FONCTION 1: lire N bits

Lire nbi bits (booléens) en commençant à l'adresse de réseau addH/addL

Question	slv	01	addH	addL	00	nbi	crcH	crcL

Réponse	slv	01	nbb	bfd	...		crcH	crcL
				octet 1		octet nbb		

bfd est un champ de bits de nbb octets avec le fomat suivant:



Bit 1 correspond à la valeur de la variable à l'adresse réseau addH/addL.
Bit nbi correspond à la valeur de la variable à l'adresse réseau addH/addL + nbi -1.
X signifie valeur indéfinie.

FONCTION 3: lire N mots

Lire nbw mots en commençant à l'adresse de réseau addH/addL

Question	slv	03	addH	addL	00	nbw	crcH	crcL

Réponse	slv	03	nbb	vH	vL	...	crcH	crcL

nbb correspond au nombre d'octets vH, vL.

FONCTION 5: écrire 1 bit

Ecrire un bit (booléen) à l'adresse de réseau addH/addL

Question	slv	05	addH	addL	vH	00	crcH	crcL
Réponse	slv	05	addH	addL	vH	00	crcH	crcL

FONCTION 6: écrire 1 mot

Ecrire un mot à l'adresse de réseau addH/addL

Question	slv	06	addH	addL	vH	vL	crcH	crcL
Réponse	slv	06	addH	addL	vH	vL	crcH	crcL

FONCTION 16: écrire N mots

Ecrire nbw mots en commençant à l'adresse de réseau addH/addL (nbb = 2nbw)

Question	slv	10	addH	addL	00	nbw	nbb	vH	vL	...	crcH	crcL
Réponse	slv	10	addH	addL	00	nbw	crcH	crcL				

Exemples:

- FONCTION 1: lire15 bits en commençant à l'adresse de réseau 0x1020, sur esclave 1

Question	01	01	10	20	00	0F	79	04
Réponse	01	01	02	00	12	39	F1	

  La valeur lue est 0x0012, ce qui donne 00000000 00010010 en champ de bits.
  Dans cet exemple les variables 0x1029 et 0x102C sont TRUE, toutes les autres sont FALSE.

- FONCTION 16: écrire 2 mots à l'adresse de réseau 0x2100, sur esclave 1. Les valeurs écrites sont 0x1234 et 0x5678.

Question	01	10	21	00	00	02	04	12	34	56	78	1C	CA
Réponse	01	10	21	00	00	02	4B	F4					

Transfert de fichier

Comparé à des bus de terrain modernes le protocole Modbus offre des services très pauvres, s'il n'est pas complété par des codes de fonction fabricant spécifiques.

Dans notre cas, lorsque ISaGRAF est exécuté sur une base matérielle performante et flexible il existe deux réstrictions au protocole Modbus:
- On ne peut accéder qu'aux variables ISaGRAF
- Il est difficile d'exécuter un transfert rapide d'un grand nombre de données

Pour ces raisons ISaGRAF offre un ensemble de requêtes "de type Modbus" pour le transfert de fichiers ou un protocole de "gestion de fichiers remote". Ces fonctionnalités doivent être implémentées pour valider:
- le téléchargement des fichiers binaires ou ASCII
- la relecture des fichiers binaires ou ASCII
- l'échange dynamique de données via un fichier virtuel ou physique partagé.

De cette manière toute application "indépendante d'ISaGRAF" peut facilement communiquer avec une cible en utilisant la liaison de communication ISaGRAF.

Le protocole est basé sur les concepts suivants:
- Le fichier sur la cible ISaGRAF s'appelle **remote file**
- Le fichier sur l'ordinateur maître s'appelle **local file**
- L'accés à chaque octet d'un fichier s'effectue avec une **adresse de base** de 32 bits et une **adresse d'octets** de 16 bits.

Il existe des requêtes pour la sélection du nom du fichier remote, de l'adresse de base, et pour lire ou écrire les données du fichier remote, en utilisant l'adresse d'octets de 16 bits.

FONCTION 17: écrire des données

nbb correspond au nombre d'octets vH, vL

Question	slv	11	addH	addL	00	nbb	nbb	vH	vL	...	crcH	crcL

Réponse	slv	11	addH	addL	00	nbb	crcH	crcL

La signification de cette requête diffère suivant la plage d'adresses addH/addL:

- **0xF000: initialiser le nom du fichier remote**
  nbb correspond au nombre de caractères pour le nom du fichier, spécifié dans les champs (dans ce cas Haut et Bas n'ont pas de signification) et **incluant \0** pour la fin de la chaîne de caractères.
  Si le fichier n'existe pas, il est créé avec les attributs "writable + readable + executable" (écrivable, lisible, exécutable).

- **0xF002: Modifier l'adresse de base à la valeur spécifiée**
  nbb devrait être égal à 4. Le premier octet vH/vL Byte correspond au mot Haut de la valeur spécifiée. Toute valeur de 32 bit est acceptée.
  Toutes les requêtes futures de lecture ou d'écriture utiliseront cette adresse de base. Si cette requête n'est pas utilisée la valeur par défaut de l'adresse de base est zéro.

- **0xF004: Effacer le fichier**
  nbb devrait être égale à zéro.
  Le fichier est effacé s'il existe et si ceci est possible.

- **Plus grand que 0xF004: Réservé**

- **Plus petit que 0xF000: Ecrire des octets**
  L'adresse spécifiée à laquelle les octets doivent être écrits est spécifiée dans addH/addL. Elle doit être plus petite que F000. Les octets spécifiés (nbb octets dans les champs vH vL où Haut et Bas ne peuvent plus avoir de signification) sont écrits dans l'ordre donnée (de gauche à droite) au nom de fichier remote sélectionné auparavant. L'adresse de début à laquelle on écrit est l'adresse spécifiée ajoutée à l'adresse de base sélectionnée auparavant. Si l'accés d'adresses résultant dépasse la taille courante du fichier, le fichier sera étendu. Il n'est pas possible de réduire la taille du fichier.

FONCTION 18: Lire un fichier

Question	slv	12	addH	addL	00	nbb	crcH	crcL

Réponse	slv	12	nbb	V	V	...	crcH	crcL

L'adresse spécifiée à laquelle les octets doivent être lus est spécifiée dans addH/addL. Elle doit être plus petite que F000. Lire le nombre spécifié d'octets (nbb) du nom de fichier remote sélectionné auparavant, en commençant à l'adresse spécifiée (addH/addL avec toute valeur 16 bits) additionée à l'adresse de base sélectionnée auparavant.
Les valeurs sont récupérées (V champs de gauche à droite) selon l'ordre dans lequel elles sont lues dans le fichier.

Exemple:

Sélectionner le nom de fichier remote: 'target.fil'.

Question	01	11	F0	00	00	0B	0B	74	...	00	25	9F

Réponse	01	11	F0	00	00	0B	8F	0E

Sélectionner l'adresse de base: 0x10000.

Question	01	11	F0	02	00	04	04	00	01	00	00	76	11

Réponse	01	11	F0	02	00	04	6E	CA

4 Ecrire des octets: adresse absolue 0x107D0, valeurs 01,02,03,04.

Question	01	11	07	D0	00	04	04	01	02	03	04	28	6F

Réponse	01	11	07	D0	00	04	FC	87

4 Lire des octets: adresse absolue 0x107D0.

Question	01	12	07	D0	00	04	B8	87

Réponse	01	12	04	01	02	03	04	58	7D

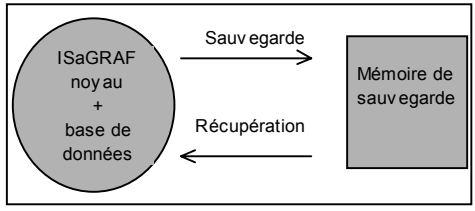
C.9 Gestion des pannes de courant

C.9.1 Bases

La gestion d'une panne de courant est extrèmement critique dans la vie d'une application, pour trois raisons:
– Elle dépend des spécifications du procédé
– Elle dépend des capacités du matériel
– Elle dépend des méthodes de programmation.

Ainsi, la solution d'ISaGRAF pour la gestion des pannes de courant n'est pas une méthode complète et universelle, mais un ensemble de principes, méthodes et outils qui doivent être combinés d'une façon particulière pour chaque application, ou au moins pour chaque matériel.

Afin de permettre à un système de visualisation de procédé de rédémarrer correctement après une coupure de courant, 3 problèmes doivent être résolus:



– Effectuer une sauvegarde des données
– Détecter l'occurence d'une panne de courant au démarrage
– Récupérer les données sauvegardées

Le deuxième problème ne permet pas de solution logicielle standard. Le fournisseur du matériel pourrait fournir les outils nécessaires pour accéder à l'état du matériel à partir d'une application ISaGRAF ou d'un programme en C.

Par ailleurs, il est important de décider quelles données sauvegarder et récupérer. Nous allons définir deux types de données:
– Les variables de l'application:
  - les variables du procédé, comme nombre des éléments traités, date de la panne de courant, valeurs des paramètres de l'application, etc....
  - les variables de programme, comme les compteurs, temporisations, valeurs intermédiaires et drapeaux.

– L'état du programme:
  comme les références des étapes actives, l'état de chaque programme C, etc...

Ces deux cas et les solutions ISaGRAF correspondantes sont décrits dans les chapitres suivants.

C.9.2 Sauvegarde des variables de l'application

Variables non-volatiles

L'éditeur de variables de l'atelier permet de sélectionner l'attribut "non-volatile" pour chaque variable interne (excepté E/S).

A la fin de chaque cycle de la cible les valeurs des variables non-volatiles sont copiées dans une mémoire spéciale, habituellement un RAM soutenue par pile.

Au lancement si au moins une variable a l'attribut "non-volatile" ISaGRAF cherche les variable non-volatiles:
- Si la même application était en exécution avant, ISaGRAF reconnaît les valeurs sauvegardées et les assigne à toutes les variables "non-volatiles".
- Si l'application précédente était différente ou si aucune application était en exécution ISaGRAF reconnaît que les variables retenues ne sont pas valables et remet toutes les variables "non-volatiles" à zéro.

La spécification de la zone de mémoire utilisée pour la sauvegarde des différents types de variables est spécifiée dans l'atelier, dans le menu **Paramètres d'exécution** ; **variables non-volatiles.**
La chaîne de caractères spécifiée doit avoir le format suivant:

boo_add , boo_size , ana_add , ana_size , tmr_add , tmr_size , msg_add , msg_size

avec:

boo_add: Adresse hexadécimale utilisée pour stocker des variables booléennes. Doit toujours être différente de zéro.

boo_size: Taille hexadécimale en octets disponible à cette adresse. Un octet est nécessaire par variable booléenne à sauvegarder.

ana_add: Adresse hexadécimale utilisée pour stocker des variables analogiques. Doit toujours être différente de zéro.

ana_size: Taille hexadécimale en octets disponible à cette adresse. Un minmum de 4 octets est toujours nécessaire et 4 octets par variable analogique à sauvegarder

tmr_add: Adresse hexadécimale utilisée pour stocker des variables de temporisation. Doit toujours être différente de zéro.

tmr_size: Taille hexadécimale en octets disponible à cette adresse. 5 octets sont nécessaires par variable de temporisation à sauvegarder.

msg_add: Adresse hexadécimale utilisée pour stocker des variables messages. Doit toujours être différente de zéro.

msg_size: Taille hexadécimale en octets disponible à cette adresse. 256 octets sont nécessaires par variable message à sauvegarder.

Exigences:

- Tous les champs de tous les types doivent être spécifiés, même si tous les types de variables ne doivent pas être sauvegardés. Dans ce cas la taille doit être égale à zéro (à l'exception des variables analogiques, pour lesquelles une taille de 4 octets doit être spécifiée) et une adresse quelconque différente de zéro doit être spécifiée pour le(s) type(s) de variable à ne pas sauvegarder.

Exemple:
Supposons que les variables suivantes sont à sauvegarder:
20 Variables booléennes
0 Variables analogiques
0 Variables de temporisations
3 Variables messages
La mémoire de sauvegarde se trouve à l'adresse hexadécimale 0xA2F200.
Supposons que:
Les variables booléennes sont sauvegardées à l'adresse 0xA2F200 avec la taille requise exacte de 20 octets.
Une taille minimum de 4 octets pour les analogiques est sauvegardée à l'adresse 0xA2F214.
L'adresse dummy pour les temporisations est 0xA2F200 et spécifiée avec une taille égale à zéro.
Les messages sont sauvegardées à l'adresse 0x A2F218 avec la taille requise exacte de 256*3 octets.
Alors la chaîne entrée au niveau de l'atelier devrait avoir l'apparence suivante:

```
A2F200,14,A2F214,4,A2F200,0,A2F218,300
```

Appel de la fonction SYSTEM

Si la plupart des variables de l'application doivent être sauvegardées il est préférable d'utiliser les facilités de la fonction SYSTEM pour la manipulation d'un ensemble de variables (voir le Guide d'utilisateur pour plus d'informations sur la fonction SYSTEM). Notez que dans ce cas la sauvegarde et la récupération sont gérées par le programmeur au niveau de l'application.

Tout d'abord il est nécessaire de définir la zone d'adresses pour la sauvegarde du type de variable spécifié ou pour tous les types de variables:

<nouvelle_adresse> := SYSTEM(SYS_INITxxx,<adresse>);

où:
- <adresse> est l'adresse de sauvegarde dans la mémoire (valeur 16# pour format hexadécimal). L'adresse doit être paire, sinon l'opération ne réussit pas.
- SYS_INITxxx peut être:
  - SYS_INITBOO pour définir l'adresse de sauvegarde de toutes les variables booléennes
  - SYS_INITANA pour définir l'adresse de sauvegarde de toutes les variables analogiques.
  - SYS_INITTMR pour définir l'adresse de sauvegarde de toutes les temporisations.
  - SYS_INITALL pour définir l'adresse de sauvegarde de toutes les variables booléennes, analogiques et temporisations.
- <nouvelle_adresse> trouve la prochaine adresse libre, c'est à dire <adresse> + taille des variables sauvegardées (en octets) selon SYS_INITxxx. Ceci permet de vérifier la taille de

la mémoire de sauvegarde nécessaire. Si l'opération ne réussit pas <nouvelle_adresse> obtient zéro.

Ensuite vous pouvez demander une sauvegarde. Cette procédure peut être appelée à tout moment dans l'application. La sauvegarde ne s'effectue qu'une seule fois à la fin du cycle courant. Si le matériel dispose d'une entrée booléenne ou d'une fonction C informant l'utilisateur quand la panne de courant arrive, et permet au moins un délai de cycle ISaGRAF avant la coupure, alors la sauvegarde est seulement possible lorsqu'une panne de courant a été détectée:

<erreur> :=SYSTEM(SYS_SAVxxx,0);

où:
- SYS_SAVxxx peut être:
  - SYS_SAVBOO, pour demander la sauvegarde de toutes les variables booléennes.
  - SYS_SAVANA, pour demander la sauvegarde de toutes les variables analogiques.
  - SYS_SAVTMR, pour demander la sauvegarde de toutes les temporisations.
  - SYS_SAVALL , pour demander la sauvegarde de toutes les variables booléennes, analogiques et temporisations.
- <erreur> obtient un status d'erreur différent de zéro si l'opération a échouée (SYS_INITxxx n'a pas été appelé).

Enfin les variables doivent être restituées. Cette procédure peut être appelée à tout moment dans l'application. La restitution ne s'effectue qu'une seule fois à la fin du cycle courant. Pour s'assurer que les données sauvegardées sont valables, une variable analogique doit être réglée à une valeur constante et utilisée comme signature:

<erreur> := SYSTEM(SYS_RESTxxx,0);

où:
- SYS_RESTxxx peut être:
  - SYS_RESTBOO pour la restitution de toutes les variables booléennes.
  - SYS_RESTANA pour la restitution de toutes les variables analogiques.
  - SYS_RESTTMR pour la restitution de toutes les temporisations.
  - SYS_RESTALL pour la restitution de toutes les variables booléennes, analogique et temporisations.
- <erreur> obtient un status d'erreur différent de zéro si l'opération a échouée (SYS_INITxxx n'a pas été exécuté).

Voici le résumé des commandes de la fonction SYSTEM pour la gestion de la sauvegarde des variables:

Commande		Signification
mot-clé prédéfini	**valeur**	
SYS_INITBOO	16#20	init sauvegarde des booléennes
SYS_SAVBOO	16#21	sauvegarder les booléennes
SYS_RESTBOO	16#22	restituer les booléennes
SYS_INITANA	16#24	init sauvegarde des analogiques
SYS_SAVANA	16#25	sauvegarder les analogiques
SYS_RESTANA	16#26	restituer les analogiques
SYS_INITTMR	16#28	init sauvegarde des temporisations
SYS_SAVTMR	16#29	sauvegarder les temporisations

SYS_RESTTMR	16#2A	restituer les temporisations
SYS_INITALL	16#2C	init sauvegarde de tous les types
SYS_SAVALL	16#2D	sauvegarder tous les types
SYS_RESTALL	16#2E	restituer tous les types

Commande (mot-clé prédéfini)	**Argument**	**Valeur de retour**
SYS_INITxxx	adresse mémoire	prochaine adresse libre
SYS_SAVxxx	0	zéro si OK
SYS_RESTxxx	0	zéro si OK

Implémentation personalisée

Enfin, les fonctions ou blocs fonctionnels en C permettent de développer des procédures utilisateurs spécifiques pour accéder à une mémoire de sauvegarde soutenue par batterie, afin de stocker et de restituer des variables à tout moment dans l'application.

Exemples:

1) Procédure dédiée à une application:
backup, restore_temp, restore_date, restore_cnt sont des procédures utilisateurs en C.

```
backup(temperature, date, cnt);   stocker 3 données critiques

temperature := restore_temp();    restituer la temperature
date := restore_date();           restituer la date
cnt := restore_cnt();             restituer le compteur
```

2) Procédures à usage générale:
backup_init, backup, backup_link, restore sont des procédures utilisateurs en C.

```
save_id := backup_init(address, size);   assigner un champs sauvegardé en mémoire.
backup(save_id, cpt1, 3);                sauvegarder cpt1 comme 3ème élément.

rest_id := backup_link(address, size)    liaison de la mémoire sauvegardée.
cpt1 := restore(rest_id, 3);             restituer la valeur sauvegardée de cpt1.
```

C.9.3 Sauvegarde de l'état d'un programme

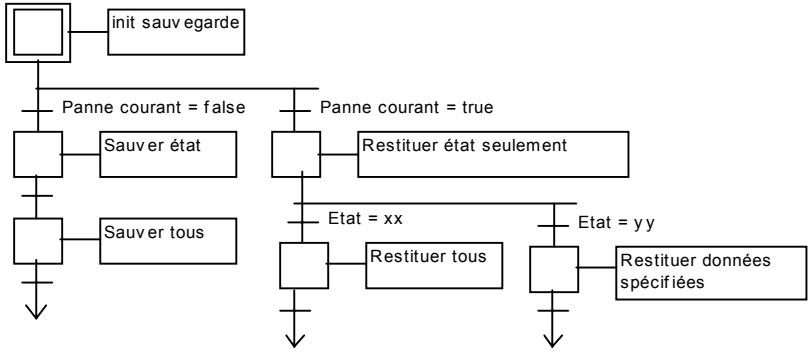
Il est tout à fait possible de stocker chaque état de chaque programme de l'application, mais il peut être dangereux de restituer chaque programme avec l'état de la dernière sauvegarde. Pour ceci il y a au moins trois raisons:

- Certains processeurs demandent des opérations spécifiques avant de rédemarrer.
- Il est ennuyeux de s'occuper de chaque état d'une application complète.
- Certaines ressources externes, comme les programmes C, des périphéries, etc. ne peuvent pas être redémarrées automatiquement.

La meilleure solution semble la sauvegarde des variables analogiques et booléennes pour décrire l'état du procédé, si le programmeur pense que ces informations pourront être utiles dans les étapes de redémarrage.

Alors il devrait être possible, à partir d'une "image" simple et intelligente du procédé, de lancer, tuer ou figer des programmes SFC et d'initialiser des variables pour mettre l'application dans un état adéquat. Mais aucune procédure de démarrage automatique n'est offerte par ISaGRAF.

Exemple:



C.10 Annexe: Liste d'erreurs et description

Liste d'erreurs:

Code	Message	Type
1	nicht genug Speicher für die Datenbank	Système
2	unzusammenhängende Datenbank (Motorola/Intel)	Application
3	nicht genug Speicher für die Kommunikations-Mailbox	Système
4	Verknüpfung mit der Datenbank unmöglich	Système
5	Time-Out während einer Anfrage an den Kernel	Système
6	Time-Out während einer Antwort vom Kernel	Système
7	Installation der Kommunikation unmöglich	Système
8	cannot allocate memory for retained variables	Application
9	application stopped	Application
10	too many simultaneous N or P actions	Application
11	too many simultaneous setting actions	Application
12	too many simultaneous resetting actions	Application
13	unknown TIC instruction	Application
16	cannot answer read data request	Système
17	cannot answer write data request	Système
18	cannot answer debugger session request	Système
19	cannot answer modbus request	Système
20	cannot answer debugger application request	Système
21	cannot answer debugger	Système
23	unknown request code	Système
24	Ethernet communication error	Système
25	communication synchro error	Système
28	cannot allocate memory for application	Système
29	cannot allocate memory for application update	Système
30	unknown OEM key code	Application
31	cannot init boolean input board	Application
32	cannot init analog input board	Application
33	cannot init message input board	Application
34	cannot init boolean output board	Application
35	cannot init analog output board	Application
36	cannot init message output board	Application
37	cannot input boolean board	Application
38	cannot input analog board	Application
39	cannot input message board	Application
40	cannot output boolean output variable	Application
41	cannot output analog output variable	Application
42	cannot output message output variable	Application
43	cannot operate boolean variable	Application
44	cannot operate analog variable	Application
45	cannot operate message variable	Application
46	cannot open board	Application

47	cannot close board	Application
50	cannot overwrite boolean output variable	Programme
51	cannot overwrite analog output variable	Programme
52	cannot overwrite message output variable	Programme
61	unknown system request code	Programme
62	sampling period overflow	Programme
63	user function not implemented	Application
64	integer divided by zero	Programme
65	conversion function not implemented	Application
66	function block not implemented	Application
67	standard function not implemented	Application
68	real divided by zero	Programme
69	invalid operate parameters	Application
72	application symbols cannot be modified	Application
73	cannot update: different set of boolean variables	Application
74	cannot update: different set of analog variables	Application
75	cannot update: different set of timer variables	Application
76	cannot update: different set of message variables	Application
77	cannot update: cannot find new Anwendung	Application
> 100	specific OEM error code, ask your supplier for more details	

Ici on distingue trois types d'erreurs:

– Erreur système:
De tels problèmes sont probablement de nature logicielle ou matérielle et n'ont rien à voir avec le réglage de l'application ou l'exécution des programmes.
Essayez de remettre votre cible (couper le courant) et d'exécuter d'autres applications.
En cas de telles erreurs, notifiez le support technique ISaGRAF.

– Erreur d'application:
De tels problèmes sont causés par les paramètres, la taille et le contenu de l'application.
Ces erreurs devront disparaître lorsqu'une application connue et préalablement validée est lancée. Si le problème persiste, il fait partie des erreurs systèmes (voir ci-dessus).

– Erreur de programme:
De telles erreurs sont causées par une séquence spécifique d'un programme.
Ces erreurs devront disparaître lorsque l'application est exécutée en mode cycle-à-cycle ou lorsque le programme critique est stoppé.

Description des erreurs:

1. cannot allocate memory for run time data base	***Système***

Pas assez de mémoire disponible. Vérifiez votre matériel.

2. incorrect application data base (Motorola/Intel)	***Application***

Le fichier d'application téléchargé ou télélu n'est pas correct. L'erreur se produit si l'application est générée pour INTEL et téléchargée sur MOTOROLA (ou l'inverse) ou si le fichier a été modifié.

3. cannot allocate communication mailbox	***Système***

Cette erreur est générée par la tâche de communication si celle-ci ne peut pas assigner l'espace 3 pour la communication inter-tâches.

4. cannot link kernel data base	***Système***

Cette erreur est générée par la tâche de communication, si elle ne peut pas trouver un noyau en exécution avec le numéro d'esclave spécifié dans sa ligne de commande.

5. time-out sending question to kernel	***Système***

La tâche de communication ne peut pas envoyer une requête au noyau. Le noyau n'est probablement pas en exécution ou occupé.

6. time-out waiting answer from kernel	***Système***

La tâche de communication ne peut pas recevoir une réponse du noyau. Le noyau n'est probablement pas en exécution ou occupé.

7. cannot init communication	***Système***

Ce message est affiché lorsque la couche de communication ne peut pas initialiser la liaison physique. Ce message s'affiche aussi si le chemin de communication n'est pas spécifié. Ceci n'empêche pas l'exécution correcte de la cible, mais celle-ci ne peut pas communiquer.

8. cannot allocate memory for retained variables	***Application***

ISaGRAF ne peut pas gérer les variables non-volatiles. Il peut y avoir deux raisons:
- la chaîne de caractères passé comme un paramètre à la cible hôte contient des erreurs de syntaxe.
- la taille de mémoire spécifiée pour chaque bloc est insuffisante.
Vérifiez la syntaxe de votre paramètre "variables non-volatiles" et essayez de réduire le nombre de variables non-volatiles.

9. application stopped	***Application***

Ce message s'affiche à chaque fois que l'application est arrêtée à partir du debugger.

10. too many simultaneous N or P actions	***Application***

Cette erreur est générée quand un des cycles de la cible doit exécuter trop d'actions P ou blocs cycliques non-stockés. Il est possible de localiser le problème en mode CC. Le nombre maximal d'actions simultanées est 2 + 4 par programme SFC.

11. too many simultaneous setting actions	***Application***

Cette erreur est générée quand un des cycles de la cible doit exécuter trop d'actions Set (exécutées quand l'étape devient active). Procédez comme ci-dessus.

12. too many simultaneous resetting actions	***Application***

Cette erreur est générée quand un des cycles de la cible doit exécuter trop d'actions Reset (exécutées quand l'étape devient inactive). Procédez comme ci-dessus.

13. unknown TIC instruction	***Application***

Le noyau a détecté une erreur dans le code d'application d'un programme (appelé Target Independent Code). Il peut y avoir deux raisons:
- un programme externe écrit dans le code d'application. Essayez de localiser le problème en mode cycle-à-cycle et assurez-vous que toutes les interfaces E/S ont des paramètres corrects.
- Votre cible travaille avec un ensemble d'instructions réduit et votre application utilise une instruction ou un type de variable qui n'est pas autorisé.

16. cannot answer read data request	***Système***

Détection d'une erreur de communication en répondant à une fonction de requête spécifique ISaGRAF Modbus code 18 (lire fichier). Vérifiez la connection et la configuration système côté cible et côté maître.

17. cannot answer write data request	***Système***

Détection d'une erreur de communication en répondant à une fonction de requête spécifique ISaGRAF Modbus code 17 (écrire fichier). Vérifiez la connection et la configuration système côté cible et côté maître.

18. cannot answer debugger session request	***Système***

Détection d'une erreur de communication en répondant à une requête du debugger. Vérifiez la connection et la configuration système côté cible et côté maître.

19. cannot answer modbus request	***Système***

Détection d'une erreur de communication en répondant à une requête Modbus. Vérifiez la connection et la configuration système côté cible et côté maître.

20. cannot answer debugger application request	***Système***

Détection d'une erreur de communication en répondant à une requête du debugger. Vérifiez la connection et la configuration système côté cible et côté maître.

21. cannot answer debugger	***Système***

Détection d'une erreur de communication en répondant à une requête du debugger. Vérifiez la connection et la configuration système côté cible et côté maître.

23. unknown request code	***Système***

Une requête de debugger n'a pas de sense.

24. Ethernet communication error	***Système***

Ce message apparaît chaque fois que la connection est fermée quand le debugger est fermé: le système marche OK. Autrement ce message signifie qu'une erreur Ethernet a été détectée. Vérifiez la connection et la configuration système côté cible et côté maître.
Un message supplementaire peut apparaître:
1: error while sending or receiving
2: error while creating the socket
3: error while binding or listening the socket
4: error while accepting a new client

25. communication synchro error	***Système***

Mauvaise synchronosation entre la tâche de communication sur la cible et le maître. Vérifiez la connection et la configuration système (paramètres de communication) côté cible et côté maître.

28. cannot allocate memory for application	***Système***

Pas de mémoire disponible. Vérifiez le matériel par rapport à la taille de l'application.

29. cannot allocate memory for application update	***Système***

Pas de mémoire disponible. Vérifiez le matériel par rapport à la taille de l'application.

30. unknown OEM key code	***Application***

L'application utilise une carte qui a un code fabricant non-reconnu par la cible. Vérifiez la connection E/S dans l'atelier et utilisez l'attribut 'VIRTUEL' pour localiser la mauvaise carte. Il est possible que la librairie de votre atelier ne correspond pas à la version de votre cible.

31. cannot init boolean input board	***Application***

L'initialisation d'une carte d'entrée booléenne a échouée. Vérifiez la connection E/S dans l'atelier et les paramètres de vos cartes d'entrée booléennes.

32. cannot init analog input board	***Application***

L'initialisation d'une carte d'entrée analogique a échouée. Vérifiez la connection E/S dans l'atelier et les paramètres de vos cartes d'entrée analogiques.

33. cannot init message input board	***Application***

L'initialisation d'une carte d'entrée message a échouée. Vérifiez la connection E/S dans l'atelier et les paramètres de vos cartes d'entrée messages.

34. cannot init boolean output board	***Application***

L'initialisation d'une carte de sortie booléenne a échouée. Vérifiez la connection E/S dans l'atelier et les paramètres de vos cartes de sortie booléennes.

35. cannot init analog output board	***Application***

L'initialisation d'une carte de sortie analogique a échouée. Vérifiez la connection E/S dans l'atelier et les paramètres de vos cartes de sortie analogiques.

36. cannot init message output board	***Application***

L'initialisation d'une carte de sortie message a échouée. Vérifiez la connection E/S dans l'atelier et les paramètres de vos cartes de sortie messages.

37. cannot input boolean board	***Application***

Une erreur a été détectée pendant l'actualisation d'une carte d'entrée booléenne. Vérifiez la connection E/S dans l'atelier et les paramètres de vos cartes.

38. cannot input analog board	***Application***

Une erreur a été détectée pendant l'actualisation d'une carte d'entrée analogique. Vérifiez la connection E/S dans l'atelier et les paramètres de vos cartes.

39. cannot input message board	***Application***

Une erreur a été détectée pendant l'actualisation d'une carte d'entrée message. Vérifiez la connection E/S dans l'atelier et les paramètres de vos cartes.

40. cannot output boolean output variable	***Application***

Une erreur a été détectée pendant l'actualisation d'une carte de sortie booléenne. Vérifiez la connection E/S dans l'atelier et les paramètres de vos cartes.

41. cannot output analog output variable	***Application***

Une erreur a été détectée pendant l'actualisation d'une carte de sortie analogique. Vérifiez la connection E/S dans l'atelier et les paramètres de vos cartes.

42. cannot output message output variable	***Application***

Une erreur a été détectée pendant l'actualisation d'une carte de sortie message. Vérifiez la connection E/S dans l'atelier et les paramètres de vos cartes.

43. cannot operate boolean variable	***Application***

Une erreur a été détectée pendant l'appel OPERATE d'une variable booléenne. Vérifiez vos paramètres OPERATE et la notice d'utilisateur de la carte.

44. cannot operate analog variable	***Application***

Une erreur a été détectée pendant l'appel OPERATE d'une variable analogique. Vérifiez vos paramètres OPERATE et la notice d'utilisateur de la carte.

45. cannot operate message variable	***Application***

Une erreur a été détectée pendant l'appel OPERATE d'une variable message. Vérifiez vos paramètres OPERATE et la notice d'utilisateur de la carte.

46. cannot open board	***Application***

L'application utilise une référence de carte inconnue par la cible. Vérifiez la connection E/S dans l'atelier. Il est possible que la librairie de votre atelier ne correspond pas à la version de votre cible.

47. cannot close board	***Application***

L'application utilise une référence de carte inconnue par la cible. Vérifiez la connection E/S dans l'atelier.

50. cannot overwrite boolean output variable	***Programme***

Deux séquences SFC écrivent la même variable de sortie booléenne dans le même cycle de la cible. Ceci devrait être évité pour empêcher un comportement hazardeux des E/S. En cas d'un tel conflit la priorité est donnée au programme le plus haut dans l'hierarchie. Si les deux programmes SFC se trouvent au même niveau, le résultat est imprévisible.

51. cannot overwrite analog output variable	***Programme***

Deux séquences SFC écrivent la même variable de sortie analogique dans le même cycle de la cible. Voir ci-dessus.

52. cannot overwrite message output variable	***Programme***

Deux séquences SFC écrivent la même variable de sortie message dans le même cycle de la cible. Voir ci-dessus.

61. unknown system request code	***Programme***

Un programme utilise l'appel SYSTEM avec un code invalide.

62. sampling period overflow	***Programme***

Le temps de cycle est plus long que spécifié dans le menu de l'atelier.
Sur un système multi-tâche ceci signifie que l'unité centrale n'a pas assez de temps pour exécuter un cycle, même si le "temps de cycle actuel" est plus court que le temps spécifié.
Sur un système mono-tâche ceci signifie toujours qu'il y a trop d'opérations dans un des cycles de la cible.

Il existe de nombreuses possibilités pour faire disparaître ce message:
- Réduire le nombre d'opérations exécutées au moment de l'affichage du message.
- Réduire le nombre de jetons et transitions valides, optimiser les traitements complexes, etc.
- Réduire la charge de l'unitée centrale des autres tâches pour donner plus de temps à ISaGRAF.
- Réduire le traffic de communication pour donner plus de temps à ISaGRAF
- Utiliser la modification dynamique de la durée de cycle, pour adapter la durée de cycle aux différentes étapes de procédé.
- Mettre le temps de cycle à zéro pour faire marcher le noyau ISaGRAF le plus vite possible sans aucun contrôle de dépassement.

63. user function not implemented	***Application***

Un programme utilise une fonction c inconnue dans la cible. Il est possible que la librairie de l'atelier ne correspond pas à la version de votre cible.

64. integer divided by zero	***Programme***

Un programme essaie de diviser une valeur analogiques entière par zéro. Votre application devrait l'empêcher car il peut y avoir des effets imprévisibles.
Si cela arrive ISaGRAF place la valeur analogique la plus grande comme résultat.
Si l'opérande est négative, le résultat est inversé.

65. conversion function not implemented	***Application***

Un programme utilise une fonction de conversion C inconnue dans la cible. Il est possible que la librairie de l'atelier ne corresponde pas à la version de votre cible.
Dans ce cas ISaGRAF ne convertie pas la valeur.

66. function block not implemented	***Application***

Un programme utilise un bloc fonctionnel C inconnu dans la cible. Il est possible que la librairie de l'atelier ne correspond pas à la version de votre cible.

67. standard function not implemented	***Application***

Un programme utilise un bloc fonctionnel C inconnu dans la cible, bien qu'il devrait être disponible sur la plupart des cibles. Renseignez-vous auprès de votre fournisseur.

68. real divided by zero	***Programme***

Un programme essaie de diviser une valeur analogique réelle par zéro. Votre application devrait l'empêcher car il peut y avoir des effets imprévisibles.
Si cela arrive ISaGRAF place la valeur analogique réelle maximale comme résultat.
Si l'opérande est négative, le résultat est inversé.

69. invalid operate parameters	***Application***

Votre application utilise un appel OPERATE avec de mauvais paramètres. En général, ceci est filtré par le compilateur. Il s'agit peut-être d'un paramètre de temporisation ou d'une variable, qui n'est pas une entrée ou une sortie.

72. application symbols cannot be modified	***Application***

En essayant d'effectuer une mise à jour de l'application, l'application modifiée ne peut pas démarrer parce que les symboles sont différents. Par rapport à l'application courante, une ou plusieurs variables ou instances de blocs fonctionnels ont peut-être été ajoutés, effacés ou modifiés.

73. cannot update: different set of boolean variables	***Application***

L'application modifiée ne peut pas démarrer, car des variables booléennes ont été ajoutées ou effacées par rapport à l'application courante.

74. cannot update: different set of analog variables	***Application***

L'application modifiée ne peut pas démarrer, car des variables analogiques ont été ajoutées ou effacées par rapport à l'application courante.

75. cannot update: different set of timer variables	***Application***

L'application modifiée ne peut pas démarrer, car des variables de temporisation ont été ajoutées ou effacées par rapport à l'application courante.

76. cannot update: different set of message variables	***Application***

L'application modifiée ne peut pas démarrer, car des variables messages ont été ajoutées ou effacées par rapport à l'application courante.

77. cannot update: cannot find new application	***Application***

Ne peut pas trouver l'application modifiée dans la mémoire. La raison peut être un mauvais téléchargement.

D. Glossaire

Action (FC) Composant d'un diagramme Flow Chart. Une action représente une liste d'instructions à exécuter lorsque le flux rencontre le symbole de l'action.

Action booléenne Action SFC: une variable booléenne est assignée avec le signal d'activité de l'étape.

Action impulsionnelle Action SFC: liste d'énoncés exécutée une seule fois quand l'étape devient active.

Action normale Action SFC: liste d'énoncés exécutée à chaque cycle pendant l'activité de l'étape.

Action Liste d'énoncés ou d'instructions exécutée quand une étape d'un programme SFC est active.

Activité d'une étape Attribut d'une étape marquée par un jeton SFC. Les actions d'une étape sont exécutées en fonction de son activité.

Adresse réseau Adresse hexadécimale optionnellement définie par le programmeur pour chaque variable. Cette adresse sera utilisée à l'exécution pour identifier la variable dans un réseau de style Modbus, pour la communication avec un système externe.

Agenda Fichier de texte contenant toutes les annotations concernant l'évolution d'un programme. Chaque annotation est complétée avec sa date et son heure de saisie.

Analogique Type de variables. Ce sont des variables continues, réelles ou entières.

Attribut Classe de variables. Les attributs disponibles sont interne, entrée et sortie.

Barre d'alimentation Barre verticale principale limitant à gauche ou à droite un réseau LD.

Barre d'outils Petite fenêtre associée à une fenêtre d'édition, qui regroupe les principaux boutons pour la sélection des composants graphiques.

Bloc fonction Composant graphique d'un réseau FBD, qui représente une fonction élémentaire définie dans les librairies ISaGRAF.

Booléen Type de variables. Ces variables ne peuvent prendre que les états vrai (true) ou faux (false).

Câblage des E/S Définition du lien logique entre les variables d'une application et les voies des cartes d'E/S implantées dans l'automate cible.

Carte d'E/S Ressource matérielle. Une carte d'E/S est caractérisée par un type et une direction (entrées ou sorties). Les paramètres d'une carte d'E/S sont décrits dans la librairie ISaGRAF.

Carte réelle Carte d'E/S physiquement connectée aux dispositifs d'E/S de la machine cible.

Carte Virtuelle Carte d'E/S déconnectée des dispositifs d'E/S physique, et dont le fonctionnement est simulé par les commandes du debugger.

Cellule Elément d'une matrice carrée pour l'édition des langages graphiques SFC, FBD ou LD.

Chaîne Suite de caractères stockés dans une variable de type message.

Cible Machine cible ISaGRAF, supportant le noyau exécutif ISaGRAF.

Clé OEM (Carte d'E/S) Code hexadécimal sur 16 bits qui définit le constructeur d'une carte d'E/S dans la librairie ISaGRAF.

Code source C Fichier de texte qui contient le code source "C" d'une fonction, bloc fonctionnel ou conversion.

Commentaire (SFC) Texte associé à une étape ou une transition SFC, et qui n'a aucun impact sur l'exécution du programme.

Commentaire Texte inclus dans le code source d'un programme, et qui n'a aucun impact sur l'exécution du programme.

Commun Portée prédéfinie. Les objets communs peuvent être utilisés par tous les programmes de tous les projets.

Condition (transition) Expression booléenne associée à une transition SFC. Une transition valide ne peut être franchie que si la condition prend la valeur TRUE.

Connecteur (FC) Composant d'un diagramme Flow Chart, qui représente un lien (flux) d'un point du diagramme vers une action ou un test. Le symbole graphique d'un connecteur est un petit cercle contenant le numéro du symbole de destination.

Contact Composant graphique d'un réseau LD. Il représente l'évaluation de l'état d'une variable booléenne.

Conversion Filtre logiciel attaché à une variable analogique d'entrée ou de sortie. La conversion est automatiquement appliquée à chaque acquisition ou mise à jour de la variable d'E/S.

Cycle Ensemble des opérations exécutées pendant un cycle du noyau exécutif ISaGRAF. Les cycles sont cadencés à une période programmable.

Cyclique Attribut d'un programme exécuté systématiquement à chaque cycle.

Décision (FC) (Appelée aussi test) Composant d'un diagramme Flow Chart associé à une expression booléenne. Le flux est dirigé vers la sortie YES ou la sortie NO du symbole selon la valeur de l'expression booléenne.

Définitions C Fichier de texte qui contient les définitions de constantes et de types "C" requis pour la programmation des des fonctions, blocs fonctionnels et conversion.

Définition Identificateur unique utilisé pour remplacer une expression dans le code source d'un programme.

Dictionnaire Ensemble des variables internes, d'entrée ou de sortie, et des définitions de mots équivalents, pour un projet.

E/S verrouillée Variable d'entrée ou de sortie, déconnectée du dispositif d'E/S correspondant par une commande de "Verrouillage" du debugger.

Enoncé Opération complète du langage ST.

Entier Classe de variables analogiques, stockées dans un format entier signé (32 bits).

Entrée Attribut de variable. Une variable d'entrée est reliée à un dispositif d'entrée (capteur).

Erreur d'exécution Erreur d'une application détectée à l'exécution par le noyau ISaGRAF.

Etape de début Première étape du corps d'une macro-étape, sans lien avec une transition en amont.

Etape de fin Dernière étape du corps d'une macro-étape, sans lien avec une transition en aval.

Etape initiale Etape d'un graphe SFC, activée par le système quand le programme SFC est lancé.

Etape Composant graphique d'un graphe SFC. Une étape représente un état stable du procédé. Elle est représentée par un carré, et référencée par un numéro. L'activité de l'étape est utilisée pour contrôler l'exécution des actions définies dans l'étape.

Etiquette (IL) Identificateur posé au début d'une ligne d'instruction IL, qui identifie l'instruction et peut être utilisé comme opérande pour les opérations de saut.

Expression constante Expression littérale utilisée pour décrire une valeur constante. Une expression constante est toujours dédiée à un type.

Expression Ensemble d'opérateurs et d'identificateurs représentant l'évaluation d'une valeur, dans le langage ST.

FBD Abréviation de "Function Block Diagram". (Langage en blocs fonctionnels).

Fiche technique Guide d'utilisation d'un élément des librairies ISaGRAF (procédure, fonction de conversion ou carte d'E/S). La fiche technique est écrite par le concepteur de l'élément.

Fonction de conversion Fonction écrite en langage "C" pour la mise en oeuvre d'une conversion analogique. Une fonction de conversion peut être attachée à toute variable d'entrée ou de sortie analogique.

Franchissement de transition Opération réalisée par le noyau ISaGRAF à l'exécution d'un programme SFC. Toutes les étapes en amont de la transition sont désactivées. Toutes les étapes en aval sont activées.

Front Changement d'état d'une variable booléenne. Un front montant est une variation de l'état "false" à l'état "true". Un front descendant est une variation de l'état "true" à l'état "false".

Function Block Diagram Langage en blocs fonctionnels. Les équations sont construites avec les boîtes fonctions standards des librairies ISaGRAF, reliées dans un réseau graphique.

Global Portée prédéfinie. Les objets globaux peuvent être utilisés par tous les programmes d'un projet.

Hiérarchie Architecture d'un projet, divisé en plusieurs programmes. L'arbre de hiérarchie représente les liens existant entre les programmes.

Identificateur Mot unique utilisé pour représenter une variable, une expression constante ou un mot clé dans les langages littéraux.

IL Abrév. de "Instruction List". (liste d'instructions).

Instruction List (Liste d'instructions) Langage littéral de bas niveau, représentant une liste séquentielle d'instructions élémentaires.

Instruction Opération élémentaire du langage IL, décrite sur une ligne de code source.

Interne Attribut de variable. Ce sont des variables stockées en mémoire, qui ne sont pas reliées à un dispositif d'entrée ou de sortie.

Jeton (SFC) Marqueur graphique utilisé pour indiquer l'activité d'une étape SFC à l'exécution.

Ladder Diagram (Schéma à contacts) Langage graphique combinant des contacts et des relais (bobines) dans un réseau, pour la modélisation des équations booléennes.

Langage C Langage littéral de haut niveau utilisé pour la description des composants informatiques tels que les procédures et les fonctions de conversion.

LD Abbréviation de "Ladder Diagram". (schéma à contacts).

Librairie Ensemble de ressources matérielles ou logicielles, pouvant être directement utilisées dans une application ISaGRAF.

Local Portée prédéfinie. Les objets locaux ne peuvent être utilisés que par un seul programme.

Macro-étape Composant graphique d'un graphe SFC. C'est la représentation par un symbole unique d'un groupe unique d'étapes et de transitions, décrit séparément dans le même programme SFC.

Matrice Division logique d'une zone de saisie graphique en cellules rectangulaires, pendant l'édition d'un programme avec un langage graphique.

Message Type de variable. Ces variables contiennent des chaînes de caractères de longueur variable.

MODBUS Protocole de type maître-esclave. Un système cible ISaGRAF peut être un esclave Modbus, relié à un système externe (tel qu'un superviseur).

Mode cycle à cycle Mode d'exécution: dans ce mode, les cycles sont exécutés un par un, déclenchés par les commandes du debugger ISaGRAF.

Mode temps réel Mode d'exécution normal: les cycles sont automatiquement cadencés conformément au temps de cycle programmé.

Modificateur (IL) Caractère complétant le nom d'un opération IL, et qui modifie la signification de l'instruction.

Mot clé Mot réservé du langage.

Niveau 1 du SFC Description générale d'un programme SFC. Le niveau 1 regroupe le dessin du graphe, les numéros de référence et les commentaires attachés aux étapes et aux transitions.

Niveau 2 du SFC Description détaillée d'un programme SFC. C'est la programmation des actions des étapes et des conditions associées aux transitions.

Numéro de référence (SFC) Nombre décimal (de 1 à 65535) qui identifie une étape ou une transition dans un programme SFC.

Opérande (IL) Variable ou expression constante traitée par une instruction du langage IL.

Opération (IL) Instruction de base du langage IL. Une opération est généralement associée à un opérande dans une ligne d'instruction.

Opération différée (IL) Opération du langage IL, modifiée par le caractère modificateur "(", qui ne sera exécutée qu'à l'apparition de la prochaine instruction ")".

Paramètre (carte d'E/S) Paramètre constant ou programmable d'une carte d'E/S. Un paramètre programmable sera renseigné lors du câblage des E/S pour une application.

Paramètre (fonction) Argument en entrée ou en sortie d'une fonction "C". Un paramètre est défini par un type et un nom de paramètre formel.

Paramètre OEM (carte d'E/S) Paramètre d'une carte d'E/S, défini par le concepteur de la carte. Il peut s'agir d'une valeur constante, ou d'un paramètre variable qui sera renseigné pendant le câblage des E/S pour chaque application.

Point d'arrêt Marque posée pendant la mise au point, sur une étape ou une transition SFC. L'exécution de l'application s'arrêtera quand un jeton SFC rencontrera le point d'arrêt.

Portée Ensemble des programmes qui peuvent utiliser un objet. Les portées prédéfinies par ISaGRAF sont: commun, global ou local.

Procédure C (ou fonction) Fonction écrite en langage "C", appelée depuis les programmes ISaGRAF (écrits en ST, FBD...), de façon synchrone.

Programme père Programme écrit en langage quelconque, qui contrôle (appelle) un autre programme (non SFC) appelé sous-programme.

Programme principal Programme situé au sommet de l'arbre de hiérarchie. Un programme principal est activé par le système.

Programme SFC fils Programme SFC contrôlé par un autre programme, appelé programme SFC père, et fonctionnant en parallèle.

Programme SFC père Programme SFC qui contrôle (lance, tue...) d'autres programmes SFC appelés programmes SFC fils.

Programme Unité de programmation. Un programme est décrit avec un langage, et a une place définie dans l'arbre de hiérarchie qui modélise l'architecture du projet.

Projet Aire de programmation qui regroupe toutes les informations (programmes, variables, code cible...) pour une application ISaGRAF.

Réel Classe de variables analogiques, stockées en format signé flottant sur 32 bits (simple précision), et pouvant prendre des valeurs non entières.

Références croisées Informations calculées par l'atelier ISaGRAF concernant le dictionnaire de données d'un projet, et l'utilisation de ces données dans les programmes du projet.

Registre (IL) Résultat courant d'une séquence de programmation en langage IL.

Relais Composant graphique d'un réseau LD, utilisé pour représenter le forçage d'une variable de sortie.

Renvoi à une étape Composant graphique d'un graphe SFC, qui représente une liaison orientée depuis une transition vers une étape. Le symbole correspondant est une flèche, référencée par le numéro de l'étape de destination.

Résultat courant (IL) Résultat d'une instruction IL. Le résultat courant est modifié par chaque instruction, et peut être utilisé pour forcer une variable ou conditionner une instruction.

Section Début	Groupe de programmes cycliques exécutés au début de chaque cycle.
Section Fin	Groupe de programmes cycliques exécutés à la fin de chaque cycle.
Section Séquentiel	Groupe de programmes exécutés conformément aux règles d'évolution du langage SFC.
Section	Groupe de programmes exécutés avec les mêmes règles.
Séparateur	Caractère spécial (ou groupe de caractères) utilisé pour séparer deux mots dans un langage littéral. Un séparateur peut avoir un rôle d'opérateur.
Sequential Function Chart	Langage graphique (GRAFCET): le procédé est divisé en un nombre connu d'étapes reliées par des transitions. Des actions sont associées aux étapes. Le franchissement des transitions est détaillé par des conditions booléennes.
SFC	Abbréviation de "Sequential Function Chart". (GRAFCET).
Situation initiale	Ensemble des étapes initiales d'un programme SFC, qui représentent l'état du procédé quand le programme est activé.
Sortie	Attribut de variable. Ces variables sont reliées à des dispositifs de sortie (actionneurs).
Sous-programme	Programme décrit en langage non SFC, activé par un autre programme nommé programme père, et exécuté de façon synchrone.
Structured Text	(Texte structuré) Langage littéral structuré de haut niveau, combinant des énoncés d'assignation, des structures de contrôle telles que If/Then/Else, et des appels de fonctions.
ST	Abréviation de "Structured Text". (Texte structuré)
Table de conversion	Ensemble de points définissant une conversion linéraire par segments. Une table de conversion peut être appliquée à toute variable analogique d'entrée ou de sortie.
Temporisation	Type de variables. Ces variables prennent une valeur temporelle, et peuvent être automatiquement rafraîchies (mises à jour) par le système ISaGRAF à l'exécution.
Temps de cycle	Période d'un cycle d'exécution.
Test (FC)	(Appelé aussi décision) Composant d'un diagramme Flow Chart associé à une expression booléenne. Le flux est dirigé vers la sortie YES ou la sortie NO du symbole selon la valeur

	de l'expression booléenne.
Transition	Composant graphique du langage SFC. Une transition représente le passage d'un groupe d'étapes vers un autre groupe d'étapes, et est conditionnée par une expression booléenne.
Type	Classe de variables ayant le même format. Les types de base prédéfinis sont: booléen, analogique, temporisation et message.
Valeur de retour d'un sous-programme	Valeur retournée par un sous-programme à la fin de son exécution. La valeur de retour est utilisée dans les équations du programme appelant.
Validité d'une transition	Attribut d'une transition. Une transition est validée quand toutes les étapes directement précédentes sont actives.
Variable d'E/S	Variable connectée à un dispositif d'E/S (capteur ou actionneur). Une variable d'E/S doit être connectée à une voie d'une carte d'E/S.
Variable	Représentation d'une donnée élémentaire, traitée par les programmes d'un projet.
Voie d'E/S	Point de connexion élémentaire d'une carte d'E/S. (emplacement pour la connexion d'une variable d'E/S)
FC	Abréviation de "Flow Chart". (Diagramme de flux)
Flow Chart	(Diagramme de flux) Langage graphique permettant la représentation d'un flux matérialisé par des actions à effectuer et des décisions pouvant diriger le flux vers un point ou un autre du diagramme. Le Flow Chart permet l'écriture de boucles exécutées sur plusieurs cycles.

E. Index général

-, B-247
%, A-87, B-179
&, B-244
) (opérateur IL), B-238
*, B-248
/, B-249
:=, A-133
+, B-246
<, B-252
<=, B-253
<>, B-256
=, B-255
=1, B-246
>, B-254
>=, B-255
>=1, B-245
1 gain, B-243

A

ABS, B-282
ACOS, B-286
Action, A-43, A-47, B-189, B-194, B-200, B-201, D-429
Action booléenne, A-40, B-190, D-429
Action impulsionnelle, B-191, D-429
Action normale, B-192, D-429
Action spécifique, B-200, B-201
Activer, A-106
Activité d'une étape, B-184, B-185, B-197, B-229, D-429
Addition, B-246
Addition de messages, B-261
Adresse réseau, A-74, A-76, A-79, D-429
Afficher texte, A-118
Agenda, A-26, D-429
Alias, A-56
ANA, B-258
Analogique, B-176, B-177, B-180, C-369, C-370, D-429
Analyseur de cycle, A-129
AND, B-244
AND_MASK, B-250
AnyTarget, A-99
appli.tst, C-328, C-338, C-351, C-359
appli.x6m, C-338, C-350
appli.x8m, C-328, C-359
Arc cosinus, B-286
Arc tangente, B-287
Archive, A-22, A-140, A-147, A-155
ARCREATE, B-309
Argument, A-144
ARREAD, B-310
Arrêter, A-106
ARWRITE, B-311
ASCII, B-300
Assignation, B-223, B-243
ATAN, B-287
Atteindre, A-38, A-47, A-66
Attribut, D-429
AVERAGE, B-275

B

Barre d'alimentation, A-50, A-51, A-58, B-208, D-429
Barre d'outils, D-429
Base, B-176
Base de temps, B-280
Begin, B-199
Binaire, B-176
BinaryFile, A-98
Bitmap, A-118
BLINK, B-279
Bloc fonction, B-204, D-429
Bloc fonctionnel, A-23, A-26, A-51, A-59, A-63, A-74, A-77, A-139, A-143, B-173, B-207, C-383

Bloc fonctionnel (appel en IL), B-240
Bloc fonctionnel (appel en ST), B-221
Bloc fonctionnel C, A-145, C-368
BOO, B-257
Booléen, A-77, B-176, B-180, D-430
Branche, A-36
BY, B-227

C

Câblage des E/S, D-430
Câblage des E/S, A-29, A-85
CAL (opérateur IL), B-240
Carte, A-85, A-86, A-88
Carte d'E/S, A-142, D-430
Carte réelle, A-86, D-430
Carte virtuelle, A-86, A-157, C-363, D-430
CASE, B-225
Cat, B-261
Cellule, D-430
Chaîne de caractères, B-177, B-181, D-430
Champ de bits, A-119
CHAR, B-300
Chercher, A-38, A-47, A-55, A-62, A-66, A-71
Cible, A-94, D-430
Clé de protection, A-14
Clé OEM, A-142, D-430
Clignotant, B-279
CLKRATE, C-342
CMP, B-273
Code source C, A-96, A-140, C-372, C-379, C-388, C-399, D-430
Code source embarqué, A-124
Coins, A-60
Coller FBD, A-62
Coller FC, A-46
Coller LD, A-55
Coller SFC, A-37
Coller texte, A-66
Coller variable, A-76
Commentaire, B-181, B-219, B-234, D-430
Commentaire (SFC), B-184, B-185, D-430
Commentaire de programme, A-25
Commentaire de voie, A-87
Commentaire d'échelle, A-53, A-56
Commentaire FBD, A-61
Commentaire FC, A-45, B-202
Commun, A-147, D-430
Communication, A-31, A-108, A-123, A-160, C-326, C-331, C-333, C-336, C-341, C-354
Comparateur, B-273
Compilateur C, C-368, C-399
Compiler, A-28, A-93, A-139, A-144
Compression, A-148
Compteur croissant, B-267
Compteur croissant / décroissant, B-269
Compteur de temps, B-181
Compteur décroissant, B-268
Concaténation de message, B-261
Condition, B-200
Condition (transition), B-195, D-430
Configuration d'E/S, A-19, A-140
Connecteur, A-45, B-202, D-430
Connexion, A-60, A-61, A-62
Connexion inversée, A-60, A-61
Contact, A-50, A-59, B-210, D-431
Contact avec front, B-211
Contact direct, B-210
Contact inversé, B-211
Contrôle de fin de cycle (VxWorks), C-343, C-346
Convergence, A-34, A-36, B-186
Conversion, A-90, D-431
Conversion ASCII -> caractère, B-300
Conversion caractère -> ASCII, B-300
Conversion en booléen, B-257
Conversion en entier, B-258
Conversion en message, B-260
Conversion en réel, B-258
Conversion en temporisation, B-259
Copier FBD, A-62
Copier FC, A-46
Copier LD, A-55
Copier librairie, A-139

Copier programme, A-27
Copier SFC, A-37
Copier texte, A-66
Copier variable, A-76
Corps d'une macro-étape, B-189
COS, B-288
Cosinus, B-288
Couper FBD, A-62
Couper FC, A-46
Couper LD, A-55
Couper SFC, A-37
Couper texte, A-66
Coupert variable, A-76
Courbe, A-118, A-119, A-122
CTD, B-268
CTU, B-267
CTUD, B-269
Cycle, A-134, B-170, B-174, D-431
Cycle à cycle, A-29, A-107
Cyclique, B-170, D-431

D

Date courante, B-308
DAY_TIME, B-308
DDE, A-113
DDE (cible NT), C-359, C-363, C-365
Débit de transmission, A-32
Debugger, A-30, A-105, A-126
Début, A-23
Décalage à droite, B-292
Décalage à gauche, B-292
Décimal, B-176, B-177
Décision, A-43, A-46, A-47, B-200, D-431
Déclaration, A-26, A-73
Décompteur, B-268
Définition, B-182, D-431
Définitions C, C-371, C-378, C-387, C-399, D-431
DELETE, B-301
Démarrer, A-106
Déplacer carte, A-85
Déplacer FBD, A-61
Déplacer FC, A-46
Déplacer programme, A-27
Déplacer projet, A-19
Déplacer SFC, A-37
Déplacer SpotLight, A-120
DERIVATE, B-279
Descripteur, A-19, A-20, A-29
Déverrouiller, A-108
Diagnostic, A-126
Dictionnaire, A-26, A-68, A-73, A-103, A-144, C-370, C-383, D-431
Différentiation, B-279
Disque, A-12
Dissocier, A-120
Divergence, A-34, A-36, B-186
Division, B-249
DO, B-226, B-227
Document, A-150
Dossier, A-20, A-30, A-150
Droit d'accès, A-154
Dump, A-116
Durée d'activation, B-185, B-229

E

E/S, A-29, A-85, A-86, A-87, A-104, A-108, A-127, A-140, A-141, A-142, A-156, A-157
E/S verrouillée, D-431
Echelle, A-50, A-51, A-55, A-61
Editer le descripteur, A-20
Editeur de texte, A-66
Editeur FBD, A-68
Editeur FBD, A-58
Editeur Flow Chart, A-43
Editeur IL, A-68
Editeur LD, A-50
Editeur Quick LD, A-68
Editeur SFC, A-33, A-68
Editeur ST, A-68
ELSE, B-224, B-225
ELSIF, B-224
Emplacement, A-86
EN, A-51
En ligne, A-30, A-105
End, A-137, B-199

END_CASE, B-225
END_FOR, B-227
END_IF, B-224
END_REPEAT, B-226
END_WHILE, B-226
ENO, A-52
Enoncé, B-219, D-431
Enregistrer liste, A-115
Entier, A-77, B-176, D-431
Entrée, A-85, A-104, A-127, A-129, A-142, B-174, D-431
EQ (opérateur IL), B-255
Equipement complexe d'E/S, A-141
Equivalence, B-182
Erreur, A-96
Erreur d'exécution, A-29, A-108, B-262, D-431
Espace de travail, A-31
Espionner, A-115, A-117, A-118
Espionner variable, A-115
Est différent de, B-256
Est égal à, B-255
Etape, A-33, A-38, A-109, B-184, D-432
Etape de début, A-35, A-37, B-189, D-431
Etape de fin, A-35, A-37, B-189, D-431
Etape initiale, B-185, B-197, D-432
Ethernet, A-32
Etiquette, A-53, A-59, A-135, B-205, B-216
Etiquette (IL), B-234, D-432
Exécuter un cycle, A-107
Exécution, A-29
EXIT, B-228
Exponentiation, B-283
Export, A-80
Exporter bloc fonctionnel, A-28
Exporter fonction, A-28
Expression, D-432
Expression constante, B-176, D-432
EXPT, B-283

F

F_CLOSE, B-313
F_EOF, B-314
F_ROPEN, B-312
F_TRIG, B-266
F_WOPEN, B-312
FA_READ, B-315
FA_WRITE, B-317
FALSE, A-77, B-176
FBD, A-58, B-204, C-376, C-385, D-432
FC, A-43, B-199, D-437
FC connecteur, A-45
FC sous-programme, A-24
FEDGE, B-222
Fenêtre de messages, A-71
Fiche technique, A-86, A-139, C-369, C-370, C-375, C-384, D-432
Fichier (détection de fin de fichier), B-314
Fichier (écriture), B-317, B-320
Fichier (fermeture), B-313
Fichier (lecture), B-315, B-319
Fichier (ouverture), B-312, B-313
Fichier d'archive, A-148
Fichier de définition des ressources, A-97
Fils, A-24
Fils SFC, B-171
Fin, A-23
FIND, B-302
Flow Chart, A-43, B-199, D-437
Flux, A-45, B-199, B-202, B-203
FM_READ, B-318
FM_WRITE, B-320
Fonction, A-23, A-26, A-139, A-143, B-171, B-196, B-207
Fonction (appel en IL), B-239
Fonction (appel en ST), B-220
Fonction C, A-145, C-368, C-375
Fonction de conversion, A-145, C-368, C-369, D-432
Fonction SYSTEM, C-415
FOR, B-227
Franchissement de transition, B-197, D-432
From, A-100

Front, D-432
Front descendant, B-211, B-215, B-222
Front montant, B-211, B-215, B-222
Function Block Diagram, B-204, D-432

G

gain 1, B-243
Galerie, A-41
GE (opérateur IL), B-255
Génération de code, A-93
Générer, A-28, A-93
Gestionnaire de librairie, A-138
Gestionnaire de librairies, C-368, C-370, C-375, C-384
Gestionnaire de programmes, A-23
Gestionnaire de projets, A-19
GFREEZE, B-198, B-231
GKILL, B-198, B-231
Global, B-178, D-432
Goto, A-136
GRAFCET, A-33
Graphique, A-118, A-122
Grille, A-52
Groupe, A-13, A-21
Groupe de projets, A-21
Grouper, A-120
GRST, B-198, B-232
GSTART, B-198, B-230
GSTATUS, B-198, B-232
GT (opérateur IL), B-254

H

heure courante, B-308
Hexadécimal, B-176
Hiérarchie, A-23, A-27, B-170, B-197, D-432
Histogramme, A-118
Historique, A-20, A-30
Horloge, B-181
HYSTER, B-276
Hysteresis, B-276, B-277

I

Icone, A-13, A-119
Identificateur, D-433
If, A-136
IF, B-202, B-224
IL, A-66, A-117, B-194, B-195, B-234, D-433
Image de fond de plan, A-118
Import, A-80
Importer bloc fonctionnel, A-28
Importer fonction, A-28
Imprimer, A-30, A-75, A-150, A-152
Imprimer programme, A-70
Impulsionnelles, A-39
Imrpimer, A-20
Insérer carte, A-85
Insérer contact, A-54
Insérer échelle, A-55
Insérer élément FBD, A-60
Insérer élément FC, A-44
Insérer FBD, A-63
Insérer fichier, A-66
Insérer relais, A-54
Insérer variable, A-39
INSERT, B-303
Installer, A-12
Instance, A-74, A-77
Instance de bloc fonctionnel, C-383
Instruction, B-234, D-433
Instruction List, B-234, D-433
INTEGRAL, B-278
Interface, A-26, A-144
Interne, D-433
Inversion booléenne (FBD), B-206
ISA.EXE, C-326
ISA.O (VxWorks), C-341
isa_main, C-343, C-346
isa_register_slave, C-342
ISAGRAF.INI (Cible NT), C-353
ISAKERET.O (VxWorks), C-341, C-344
ISAKERSE.O (VxWorks), C-341, C-344
ISAMOD (VxWorks), C-341

ISAMOD.EXE, C-326
ISAx0, C-337
ISAx1, C-337
ISAx1 ·(NT), C-358
ISAx1 ·(VxWorks), C-349
ISAx2, C-337
ISAx3, C-337
ISAx4, C-337
ISAx5, C-337
ISAx6, C-337
ISAx6 (NT), C-358
ISAx6 (VxWorks), C-350

J

Jeton (SFC), B-184, D-433
JMP (opérateur IL), B-237
Jour courant, B-308

L

Ladder Diagram, B-208, D-433
Langage, A-24, B-174
Langage C, C-368, C-371, C-372, C-378, C-387, C-388, C-399, D-433
LD, A-41, A-48, A-50, A-58, B-208, D-433
LD (opérateur IL), B-236
LE (opérateur IL), B-253
LEFT, B-304
Liaison, A-31, A-123, A-160
Liaison (FBD), B-205
Liaison (LD), B-208
Liaison (SFC), B-185
Liaison série, A-32
Librairie, A-22, A-28, A-86, A-103, A-128, A-138, A-147, C-368, D-433
Lien, A-60, A-61, A-62, A-108, B-199, B-202, B-203
Lien FC, A-45
LIM_ALRM, B-277
LIMIT, B-294
Limite de la taille de l'application, C-330
Liste de variables, A-115, A-117, A-120
Liste des projets, A-19, A-21
Local, A-144, B-178, D-433
LOG, B-283
Logarithme, B-283
Longueur chaîne de caractère, B-306
LT (opérateur IL), B-252

M

Macro-étape, A-35, A-37, B-188, D-433
Marquage des modifications, A-64
Masque sur bits d'entier (et), B-250
Masque sur bits d'entier (ou exclusif), B-251
Masque sur bits d'entier (ou), B-250
Masque sur les bits d'un entier (inversion), B-252
Matrice, D-433
MAX, B-293
Maximum, B-294
Mémoire, A-12
Menu Outils, A-30
Message, A-77, A-116, B-177, B-181, D-433
Message (longueur), B-305
Message de compilation, A-96
Message d'erreur, A-71
Metafile, A-118
MID, B-305
MIN, B-293
Minimum, B-293
Mise à jour, A-107
Mise au point, A-30, A-105, A-126
MLEN, B-305
MOD, B-295
MODBUS, A-79, C-406, D-433
Mode cycle à cycle, D-434
Mode temps réel, D-434
Mode terminal, C-340
Modificateur (IL), B-234, B-235, D-434
Modification en ligne, A-107, A-110
Modifier variable, A-76
Modulo, B-295
Mot clé, D-434
Mot de passe, A-20, A-89, A-139, A-154
Mot équivalent, A-73, A-77

Mots clés, B-178
Moyenne, B-276
MSG, B-260
Multiplexeur 4 entrées, B-296
Multiplexeur 8 entrées, B-297
Multiplication, B-248
MUX4, B-296
MUX8, B-297

N

N qualificatif, A-39
NE (opérateur IL), B-256
NEG, B-243
Négation, B-243
Niveau 1 du SFC, B-184, B-185, D-434
Niveau 2, A-38, A-47
Niveau 2 du SFC, B-189, D-434
Niveau de priorité (Cible NT), C-356
Niveau de protection, A-154
Nom de variable, B-178
Nom d'échelle, A-53
Nombre aléatoire, B-298
Non mémorisée, A-39
Non-volatile, C-414
NOT, A-60, A-61
NOT_MASK, B-252
Nouveau bloc fonctionnel, A-25
Nouveau programme, A-25
Nouveau projet, A-19
Nouvel élément de librairie, A-138
Nouvelle échelle, A-53
Nouvelle fonction, A-25
Nouvelle variable, A-76
NT (clé de protection), A-15
Numéro de référence, B-184, B-185, B-186, B-189, D-434
Numéro d'esclave, A-31, C-326, C-332, C-333, C-334, C-342, C-345, C-354, C-363, C-365
Numéro logique de communication, C-333, C-334, C-345

O

Octal, B-176
ODD, B-298
OF, B-225
Opérande (IL), B-234, B-235, D-434
OPERATE voie d'E/S, B-263
Opérateur (IL), B-234, B-235
Opération (IL), D-434
Opération différée (IL), B-235, B-238, D-434
Optimiseur, A-95
Options de compilation, A-29, A-94, A-124
OR, A-58, B-245
OR_MASK, B-250
Ordre d'exécution, A-63
OS-9 shell, C-340
Ouvrir programme, A-26, A-69, A-104
Ouvrir projet, A-20

P

P qualificatif, A-39
P0 qualificatif, A-40
P1 qualificatif, A-40
Page, A-152
Panneau de contrôle, A-105
Paramètre, A-26, A-144, C-376
Paramètre (bloc fonctionnel), C-386
Paramètre (carte d'E/S), D-434
Paramètre (procédure), D-434
Paramètre OEM, A-142
Paramètre OEM (carte d'E/S), D-434
Paramètres de carte, A-87, A-142
Parenthèse, B-220
Parenthèse (IL), B-235
Parité, A-32
Parité (test valeur impaire / paire), B-298
Partie entière, B-285
Pile d'analogiques entiers, B-274
Plus grand ou égal, B-255
Plus grand que, B-254
Plus petit ou égal, B-253
Plus petit que, B-252

Point, A-90, A-91
Point d'arrêt, A-107, A-109, D-434
Police, A-152
Portée, A-73, A-75, D-435
POW, B-284
Print, A-133
PrintTime, A-134
Priorité, C-363
Procédure C, D-435
Programme, A-23, A-68, A-129, B-170, D-435
Programme père, D-435
Programme principal, A-23, D-435
Programme SFC fils, B-197, B-228, D-435
Programme SFC père, B-197, D-435
Projet, A-19, A-147, D-435
Protection, A-20, A-89, A-139, A-154
Puissance, B-284

Q

Quick LD, A-41, A-48, A-50

R

R (reset) (opérateur IL), B-237
R_TRIG, B-265
Racine carrée, B-285
RAND, B-298
REAL, B-258
REDGE, B-222
Redimensionner FC, A-46
Redimensionner SpotLight, A-120
Réel, A-77, B-177, D-435
Références croisées, A-29, A-103, D-435
Registre (IL), D-435
Règles d'évolution SFC, B-196
Relais, A-50, A-59, B-210, D-435
Relais direct, B-212
Relais inversé, B-212
Relais négatif, B-215
Relais positif, B-214
Relais Reset, B-214
Relais Set, B-213
Remplacer, A-38, A-47, A-55, A-62, A-66
Renommer librairie, A-139
Renuméroter, A-38, A-47
Renvoi à une étape, B-186, D-435
REPEAT, B-202, B-226
Répertoire, A-160
Répertoire d'archivage, A-148
REPLACE, B-306
Ressource, A-29, A-97
Restitution, A-22, A-140, A-147, A-148, A-155
Résultat courant (IL), B-235, D-435
RET (opérateur IL), B-238
RETURN, A-51, A-60, B-205, B-216, B-224
RIGHT, B-307
ROL, B-290
ROR, B-291
Rotation à droite, B-291
Rotation à gauche, B-290
RS, B-264

S

S (set) (opérateur IL), B-237
Saut, A-51, A-59, B-205, B-216
Saut à une étape, A-35
Sauvegarde, A-22, A-140, A-147, A-148, A-155
Schéma à relais, A-50
Scientifique, B-177
Script, A-130, A-132
Section, A-23, D-436
Section Début, D-436
Section Fin, D-436
Section Séquentiel, D-436
SEL, B-299
Sélecteur binaire, B-299
Sélection FC, A-45
Sélectionner élément FBD, A-60
Sélectionner SpotLight, A-120
SEMA, B-267
Sémaphore, B-267
Séparateur, B-219, D-436

Séparateur de projets, A-19
Séquence '$', B-178
Sequential Function Chart, B-184, D-436
Séquentiel, A-23, B-170, B-184
SFC, A-33, A-95, A-109, A-152, B-184, C-376, D-436
SFC fils, A-24, A-40
SFC galerie, A-41
SHL, B-291
SHR, B-292
SIG_GEN, B-280
Signal (génération), B-280
Simulateur, A-30, A-127, A-129, A-130, C-370, C-375, C-383
Simuler une modification, A-29, A-93
SIN, B-289
Sinus, B-289
Situation initiale, B-185, B-197, D-436
SlavesLink, C-348
Sortie, A-85, A-104, A-127, A-142, B-174, D-436
Sous-chaîne (extraction à droite), B-307
Sous-chaîne (extraction à gauche), B-304
Sous-chaîne (extraction au milieu), B-305
Sous-chaîne (insertion), B-303
Sous-chaîne (recherche), B-302
Sous-chaîne (remplacement), B-306, B-307
Sous-chaîne (suppression), B-301
Sous-programme, A-24, B-171, B-193, B-196, B-201, B-207, D-436
Sous-programme (appel en IL), B-239
Sous-programme (appel en ST), B-220
Sous-programme FC, B-201
Soustraction, B-247
SpotLight, A-118
SQRT, B-285
SR, B-264
SSR[x][1].space, C-350
ST, A-41, A-66, A-117, B-219, C-376, C-384, D-436
ST (opérateur IL), B-236
STACKINT, B-274
Structured Text, B-219, D-436
Style, A-64, A-121
Style Modifié, A-64
Style Normal, A-64
Style Supprimé, A-64
Supprimer carte, A-86
Supprimer FBD, A-62
Supprimer FC, A-46
Supprimer LD, A-55
Supprimer librairie, A-139
Supprimer programme, A-27
Supprimer SFC, A-37
Supprimer texte, A-66
Syntaxe d'un programme, A-68
SYSTEM, B-261

T

Table de conversion, A-90, A-91, D-436
Table des matières, A-150
Table des symboles, A-162
Tableau (création), B-309
Tableau (écriture), B-311
Tableau (lecture), B-310
Tâche ISA (OS9), C-331, C-342
Tâche ISANET (OS9), C-333
Taille de l'application, C-365
TAN, B-289
Tangente, B-289
Target, A-99
Taux d'horloge système (VxWorks), C-342
Télécharger, A-106
Télélecture, A-123, A-124
Télélecture (options), A-124
temporisateur à déclenchement, B-271
Temporisateur à impulsion, B-272
Temporisation, A-77, B-177, B-181, D-436
Temps de cycle, A-29, A-107, A-129, B-262, C-370, C-375, C-383, D-436
Temps réel, A-29, A-107
Test, A-43, A-46, A-47, A-105, A-127, B-200, D-436

TextFile, A-99
THEN, B-224
Time-out, A-31
TMR, B-259
To, A-100
TO, B-227
TOF, B-271
Touches pour quitter (cible), C-329, C-361
TP, B-272
Transition, A-33, A-38, A-109, B-185, D-437
Transition invalide, B-197
Transition valide, B-197
Trier, A-76
TRUE, A-77, B-176
TRUNC, B-285
TSK_FUNIT, C-342, C-345
TSK_NBTCKSCHED, C-343, C-346
tst_main_ex, C-346
TSTART, B-181, B-229
TSTOP, B-181, B-230
Type, A-73, A-75, A-85, A-103, A-142, B-176, D-437
Type de carte, A-86
Type de contact, A-55
Type de relais, A-55

U

ULongData, A-97
Unité de sauvegarde de fichier (VxWorks), C-342, C-345
Unité de temps, B-177
UNTIL, B-226

V

Valeur absolue, B-282
Valeur de retour, D-437
Validité d'une transition, D-437
Variable, A-26, A-39, A-54, A-60, A-63, A-66, A-73, A-103, A-104, A-109, A-144, A-156, B-178, B-204, D-437
Variable d'E/S, C-369, C-370, D-437
Variable représentée directement, A-87, B-179
VarList, A-98
Vérifier, A-28, A-93, A-139, A-144
Verrouiller, A-108, A-157
Version Numéro de version, A-107
Voie, A-87, A-88, A-142, A-156
Voie d'E/S, D-437
Voie d'E/S OPERATE, B-263

W

Wait, A-135
WHILE, B-202, B-226
WISAKER.EXE (NT), C-353

X

XOR, B-246
XOR_MASK, B-251

Z

Zoom, A-49, A-56, A-64