USER'S MANUAL

IsaGraf Application Workbench
Model CC030A
User's Guide

DATE	REVISION	COMMENTS
Jan 2002	1	Initial Issue
March 2002	2	Update to Issue B

SOFTWARE COPY AVAILABLE

This manual is available in printed form or in Adobe Acrobat pdf format.

The pdf file is named UMCC030AUGR01.pdf

SCOPE

This User Guide provides information on how to install, configure and use the Omniflex ISaGraf Application Workbench.

This manual provides general IEC61131-3 programming assistance for all features of the Omniflex ISaGraf Application Workbench Version 3.46.1 as relevant for all Omniflex IEC61131-3 programmable products.

This manual is companion to the Omniflex IsaGraf Language Reference.

This manual does not cover specific programming information relevant to the specific products. This information is available in the specific User Manuals for these products.

This manual covers the following product:

Model	Description
CC030B	Omniflex ISaGraf Application Workbench Version 3.46.1

Introduction

The Omniflex IsaGraf Application Workbench is development software that runs under Microsoft Windows for developing application programs using the IEC61131-3 standard programming languages.

These programs can be run on a range of Omniflex products. Consult your Omniflex distributor for a list of available products.

For program development, the Application Workbench provides powerful and intuitive Windows based graphical and textual editors with drag-and-drop, and cut-and-paste to enhance ease of use.

The Application Workbench offers the following features:

- Project Management
- I/O Definition
- Modular Programming
- Simulation
- Real-time on-line debugging
- Document Generation
- Remote Programming over any Conet network
- Simultaneous Target Register Viewing and Editing using “DITview”

Table of Contents

1. Getting Started

This chapter covers the installation and setup of the Omniflex “ISaGRAF” Application Workbench. It also includes a short step-by-step example of an IEC61131-3 application, giving the user a brief outline of the product’s main features providing a jump start to the use of the Omniflex ISaGRAF Application Workbench.

1.1 Introduction to IEC61131-3 Programming

1.1.1 The IEC61131 Standard

The IEC61131 standard was created in the 1990’s in recognition of the need for some form of standardisation in PLC programming languages.

The IEC61131 standard is divided into a number of parts:

Part 1 General information Definition of basic terminology and concepts.

Part 2 Equipment requirements and tests Electronic and mechanical construction and verification tests. - published 1992

Part 3 Programmable languages PLC software structure, languages and program execution.

Part 4 User guidelines Guidance on selection, installation, maintenance of PLCs.

Part 5 Messaging service specification Software facilities to communicate with other devices using communications based on MAP Manufacturing Messaging Services.

Part 6 Communications via fieldbus Software facilities of PLC communications using IEC fieldbus

Part 7 Fuzzy control programming Software facilities, including standard function blocks for handling fuzzy logic within PLCs - published 1997

Part 8 Guidelines for the implementation of languages for programmable controllers Application and implementation guidelines for the IEC61131-3 languages.

The part applicable to PLC programming is IEC61131-3. The Omniflex ISaGraf Application Workbench conforms to this IEC standard for programming languages.

1.1.2 The IEC61131-3 Programming Languages

The IEC61131-3 standard defines 5 programming languages:

- Sequential Flow Chart (SFC)
  A graphical language for depicting sequential behaviour of a control system. It is used for defining control sequences that are time- and event-driven.
  Sequential Function Chart (SFC), the core language of the IEC 61131-3 standard, divides the process cycle into a number of well-defined steps, separated by transitions. The other languages are used to describe the actions performed within the steps and the logical conditions for the transitions. Parallel processes can easily be described using SFC.
- Function Block Diagram (FBD)
  A graphical language for depicting signal and data flows through function blocks - re-usable software elements. FBD is very useful for expressing the interconnection of control system algorithms and logic.
- Ladder Diagram (LD)
  A graphical language that is based on the relay ladder logic - a technique commonly used to program current generation PLCs. However, the IEC Ladder

Diagram language also allows the connection of user defined function blocks and functions and so can be used in a hierarchical design.

- Structured Text (ST)
  A high level textual language that encourages structured programming. It has a language structure (syntax) that strongly resembles PASCAL and supports a wide range of standard functions and operators.
  This language is primarily used to implement complex procedures that cannot be easily expressed with graphical languages (e.g. IF / THEN / ELSE, FOR, WHILE...).
- Instruction List (IL)
  A low level 'assembler like' textual language that is based on similar instruction list languages found in a wide range of today's PLCs.

The Application Workbench supports all 5 of these IEC61131-3 languages as well as a sixth language called “Flow Chart”.

- Flow Chart (FC)
  Recognizing that virtually every engineer graduating from college today has programmed in Flow Chart, the Workbench fully supports graphical Flow Chart programming. The Flow Chart is an easy to read decision diagram where actions are organized in a graphic flow. Binary decisions are used to control the flow. The Flow Chart Editor has full support for connectors and sub-programs. Actions and tests can be programmed in LD, ST or IL.

1.2 The Omniflex ISaGraf Application Workbench

1.2.1 Hardware and software requirements

The ISaGRAF Workbench can be installed on any computer running the following Microsoft operating systems:

- Windows 98
- Windows ME
- Windows NT
- Windows 2000
- Windows XP

1.2.2 Software Components supplied

The Omniflex ISaGraf Application Workbench is supplied on CD with the following software components:

1. IsaGraf Programmer’s Workbench
2. Conet Communications Server
3. DITview Configuration Utility

1.2.3 Installing the ISaGraf Application Workbench

To install ISaGRAF, the following steps must be performed:

- Insert the ISaGRAF CD-ROM into the appropriate drive

- From the Program Manager or the Start menu, run "SETUP.EXE" on the root folder of the CD-ROM, or "A:\INSTALL.EXE" in the case of floppy disks.
- Follow the on-line instructions to complete the installation. It is recommended that the ISaGRAF Workbench be installed in a new directory to avoid confusing files with files from other ISaGRAF versions.

The installation process will ask whether the following components are required:

- ISaGRAF executable programs
- On line information and help files
- ISaGRAF standard libraries
- ISaGRAF sample applications
- Conet Personal Server
- DITview Configuration Utility.

It is recommended that when installing ISaGRAF for the first time all components be included. Further components can, however, be added at a later date by re-installing the ISaGRAF Workbench.

The default name for the ISaGRAF main directory is "\ISAWIN". Refer to the "ISaGRAF directories" section in the "Advanced techniques" chapter for more about ISaGRAF disk architecture.

These are the Items that will be installed into the Start Menu in the ISaGraf Folder:

Projects: Project management

Libraries: Library management

Book: On-line information about ISaGRAF

Diagnosis: Diagnosis tool for end user

Read Me: Information about the ISaGRAF new version

Report: Standard Bug report form

In case you encounter a problem, use the standard bug report form. Open it, fill the items requested and use the File/Save As menu command to save it with a given file name. Then send this file to CJ International, using Fax or e-mail.

1.2.4 The protection key

A hardware key protects the ISaGRAF software against illegal copies. However, most functions of the ISaGRAF workbench are still available when the key is not inserted. The protection key also defines the option of the ISaGRAF Workbench, and defines the maximum size of developed applications. When the key is not inserted or not properly connected, some of the ISaGRAF Workbench functions will not run. This is NORMAL behaviour. To ensure that the key is properly connected, select the "About..." choice of the "**Help**" menu in any ISaGRAF window. The available option of the ISaGRAF workbench is displayed.

The key can be connected to any parallel port on the computer. If the machine has more than one parallel port, it is preferred to connect the key and the printer to

different ports. For some PC/printer configurations, the key may not be recognised when its output is connected to an "off-line" printer. In this case, disconnect the printer, or start it in the "on-line" state, and restart the ISaGRAF Workbench.

Note that no key is needed for the ISaGRAF-32 Workbench.

Important for Windows NT/Windows 2000/Windows XP users:

On these systems, the Sentinel/Rainbow Driver has to be installed in order for the protection key to be seen. A separate diskette is provided.

1.3 Using on-line information

On-line information is installed with the ISaGRAF workbench, for the following topics:

- ISaGRAF languages reference
- Complete user's guide (for any ISaGRAF tool)
- Technical note for elements in the libraries

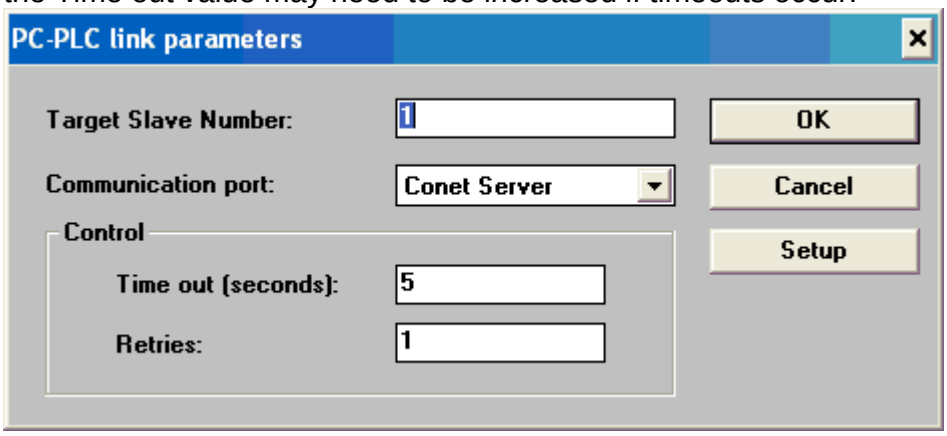
From any ISaGRAF window, select the choices of the "Help" menu to display online information.

1.4 Setting up the Application Workbench

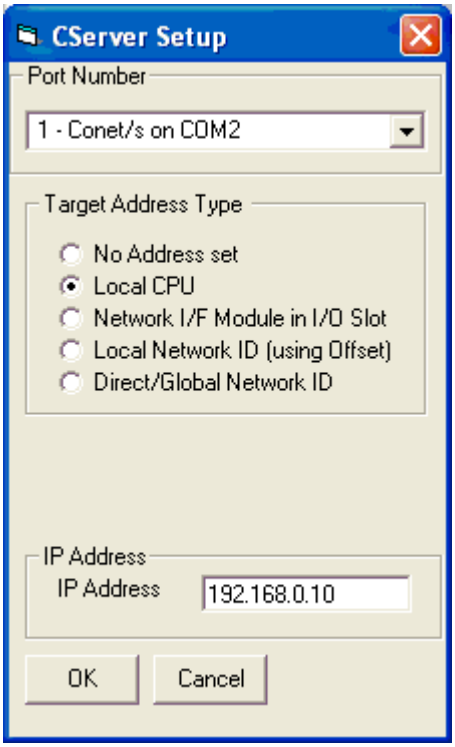
The following settings in the workbench must be set prior to use:

1.4.1 PC-PLC Link Parameters

1. Select “IsaGraf Projects” from the Start menu.
2. Start a new Project or open an existing Project.
3. Select “Link Setup” on the “Debug” menu.
4. Select all parameters as shown below. For remote programming over slow links, the Time out value may need to be increased if timeouts occur:



5. Select “Setup” and then choose the desired Conet Port for communicating with the P3 CPU. Programming can be performed over Conet/s (through a serial port on the computer), Conet/c(through a Conet/c Interface Card if installed on the computer and Conet/e(through an Ethernet port if installed on the computer).



An IP Address is only required if communicating using Conet/e over Ethernet.

6.

1.4.2 I/O Slot and Channel Numbering

Ensure that the Slot Numbering starts from 0, and the Channel Numbering starts from 1

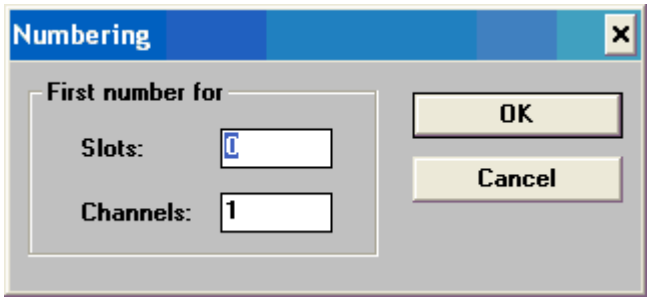
To set the Slot and channel numbering proceed as follows:

Open your Project

Open the “I/O Connection” Window from the “Project” menu.

Select “Numbering” from the “Options” menu in this window

Check that your settings match the following:

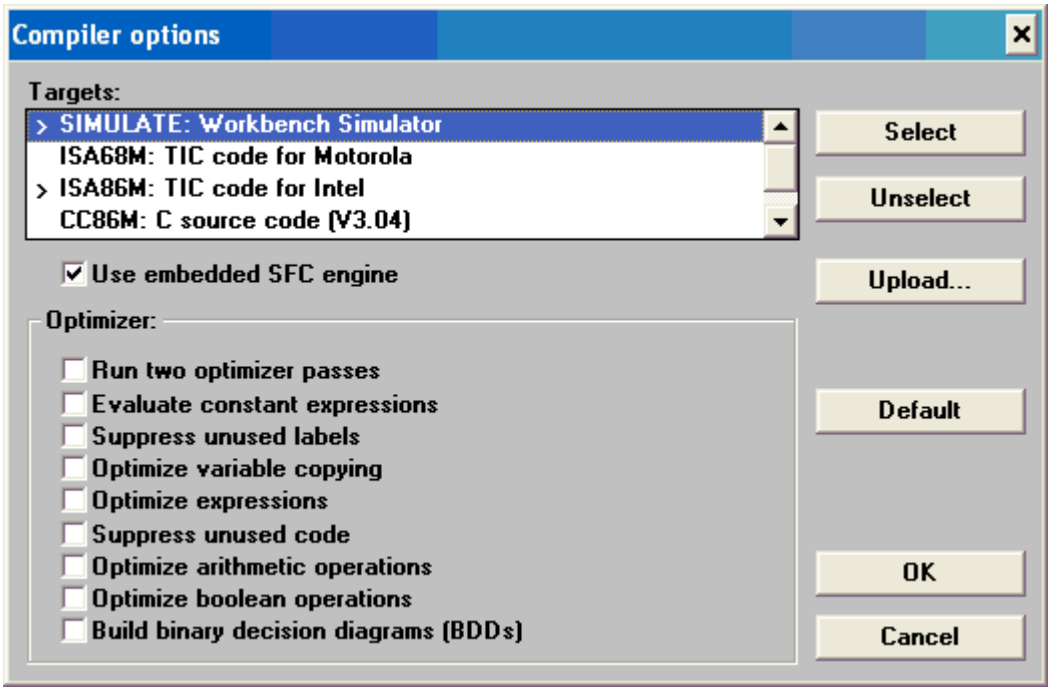


1.4.3 Compiler Options

Ensure that the following compiler options are selected in your Application Workbench in accordance with the requirements of the target product. See the target product specific User Manual for further information.

To select the compiler options proceed as follows:

1. Select “IsaGraf Projects” from the Start menu.
2. Start a new Project or open an existing Project.
3. Start a new Program or open an existing program in the project (any language).
4. Select “Compiler options” from the “Options” menu



Select the desired options according to the specific product being programmed. See the target product User Manual for more details.

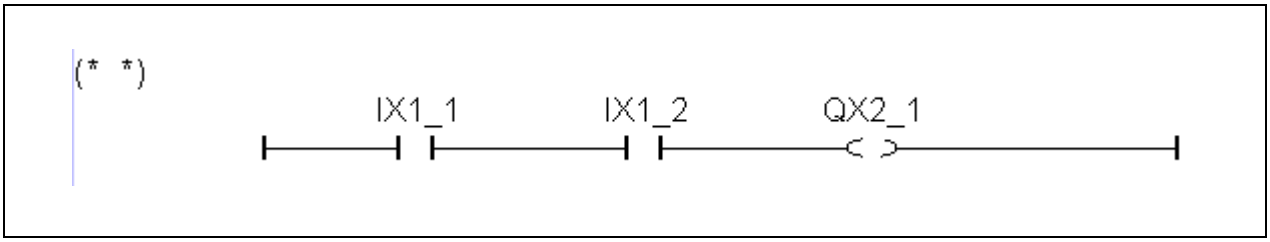
1.5 A sample application

This chapter explains, step by step, all the basic operations required to make, design, generate and test a short but complete multi-language application.

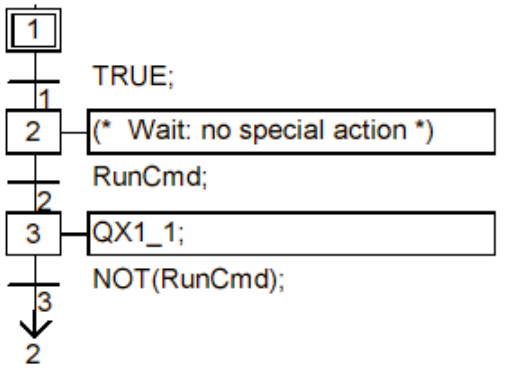
Below are the complete specifications of this application, mixing LD and SFC languages:

Boolean variables:	
IX0_1, IX0_2	input variables for process command
RunCmd	internal "run/stop" command
QX1_1	output variable: status of the process

Program Command:	Cyclic begin section - LD language (Evaluates the internal "run/stop" command)



Program RunStop: Sequential section - SFC language
(Controls the process)



BEGIN: ***Running the ISaGRAF workbench***

To run the ISaGRAF Workbench, run the "Projects" command, in the "ISaGRAF" group, from the Start menu of Windows.

Creating the project

Create the project (called "RunStop") using the "New" command of the "File" menu or the New button. In the open dialog box:

Enter project name: "**RunStop**"
Select I/O configuration: "**Sim_Boo**"
Press the "**OK**" button.
The project has now been created.

Opening the project

The programs of the project are defined by opening the ISaGRAF program management window. Use the "Open" command of the Project management window, or double click the mouse on the name of the project, or use the Edit button.

Creating the programs

The Program Management window is now open and empty (no programs defined).

The first program is created using the "New" command of the "File" menu or the "New" button. In the open dialog box:

Enter the name of the program: "**Command**".

Select the "**Quick LD**" language.
Select the "**Beginning of cycle**" section.
Press the "**OK**" button to create the program.
The same operation must be repeated for the second program:
Use the "**New**" command of the "**File**" menu, or the "**New**" button. In the open dialog box:
Enter the name of the program: "**RunStop**".
Select the "**SFC**" language.
Select the "**Sequential**" section.
Press the "**OK**" button to create the program.
The programs are now created. They appear in the Program Management window.

Declaring the variables

Before entering the programs, the internal variable to be used in the programming must be declared. This is done using the "Dictionary" command on the "File" menu, or using the Dictionary button.

I/O variables are automatically declared when the project is created.

The dictionary window is now opened. From the "**File**" menu, select "**Other**", then "**Global variables**" and then "**Booleans**".

This selects the "**Global**" boolean dictionary.

Press the “Global” button and select the “Boolean” Tab to achieve the same result.

The "New" command of the "Edit" menu is used to create new Boolean variables.

You can also use the “Insert objects” button.

In the open dialog box, enter the description of the internal variable:

name: **`RunCmd`**
comment: **`Run/Stop command: internal`**
attribute: Select the "**Internal**" attribute

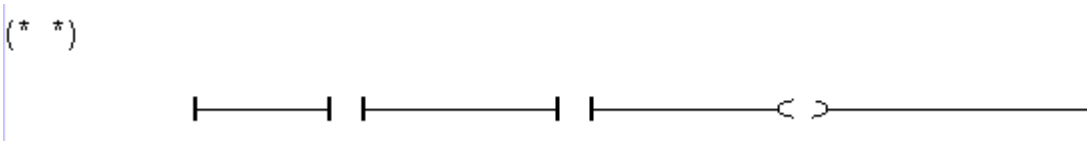
Press the "Store" button: the variable is created.
Press the "Cancel" button to exit the dialog box.
Finally, exit the dictionary editor and save the modifications entered:
Menu "**File**" - Command "**Exit**". Click on "**YES**" to save modifications.

Editing the Quick LD program

To start editing the "**Command**" LD program, double click on its name in the Program Management window or use the Edit button.

The ISaGRAF Quick LD Editor window is now open.
To increase the working area, resize the window to use the full screen size.

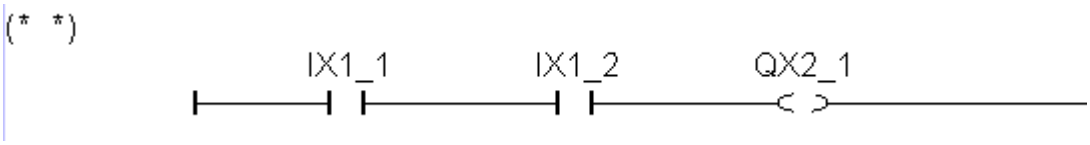
F2 F3 Press F2 and F3 key:



Enter Associate variables to the LD symbols:
Move the cursor using the keyboard arrows.
Place the cursor on each symbol and press Enter key. The variable section

dialog box is opened.
For the first contact, type in the variable selection box: IX1_1 then Enter.
For the second contact, type in the variable selection box: IX1_2 then Enter.
For the coil, type in the variable selection box: RunCmd then Enter.

The program is now complete. Here is the result:



Exit from the editor, and save the modifications entered:
Menu "**File**" – Command "**Exit**". Click on "**YES**" to save modifications.

Editing the SFC program

To start editing the "**RunStop**" SFC program, double click on its name in the Program Management window or use the Edit button.

The SFC Editor window is now open.

To increase the working area, resize the window to use the full screen size.

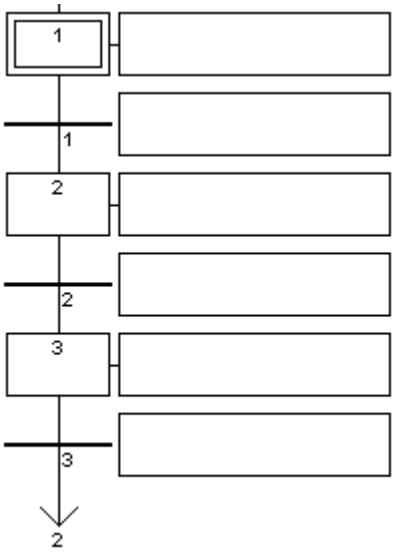
The initial step already exists and is selected. Press the "Down" keyboard arrow to select the empty cell after the initial step (0,1).

F4 F3 Press F4 then F3 to insert a step and a transition.

F4 F3 Press F4 then F3 to insert one more step and transition.

F4 F5 Press F4 then F5 to insert another transition and a jump to a step and select GS2 as the destination of the jump.

The chart is now complete. Press the "Zoom In" button in the toolbar to increase size of cells and give space to display level 2 instructions. Here is the chart:



Enter To enter the programming of transition "2", select it using the keyboard arrows and press "Enter" key. The Level 2 programming window is open. Enter level 2 programming for transition 2:

```
RunCmd;
```

^TAB Press "Control + Tab" keys to move focus back to the SFC chart, move

selection on step 3, and press "Enter" key to edit its level 2 text:

```
QX1_1;
```

And do the same to enter text of transition 3:

```
Not (RunCmd);
```

^F4 Press "Control + F4" keys to close the level 2 window.

The SFC program is now complete. Exit from the editor with Menu "**File**" and Command "**Exit**", and save the modifications entered clicking on "**YES**".

Building the application code

Use the "Make" menu and command "**Make Application**" from the Program Management window to build the application code or the button in the Toolbar.

When the code generation is complete, a dialog box appears, which asks you to exit the code generation now or to continue working with it: Press the button "**Exit**".

Simulation

Use the "**Debug**" menu and command "**Simulate**" from the Program Management window to run the ISaGRAF kernel simulator or the button in the Toolbar.

When the Simulator window appears, the application can be tested. In this example, both inputs 1 and 2 (green buttons) must be pressed to run the process (output red LED lights).

Close the Debugger window to exit from simulation:

Menu "**File**" - Command "**Exit**".

2. Managing Projects

To run the ISaGRAF project management tool, double click the mouse on the "Projects" icon, in the ISaGRAF group. The "Project Management" window is then opened. A project corresponds to one PLC loop run on a target PLC. The upper window contains the list of the existing projects. The text descriptor of the selected project is displayed in the lower window.

Building the application code

Use the "Make" menu and command "**Make Application**" from the Program Management window to build the application code or the button in the Toolbar.

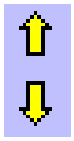
When the code generation is complete, a dialog box appears, which asks you to exit the code generation now or to continue working with it: Press the button "**Exit**".

Resizing windows

Just click on the separator (splitter) between project list and descriptor to resize corresponding windows. The descriptor window cannot be completely hidden. It always contains at least one line of text.

Inserting separators

A separator line can be inserted before any project name. This allows grouping some projects attached to the same application in the list layout. Use the "**Edit/Toggle separator**" command to insert or delete a separator before the selected project.



Moving projects in the list

To move a project in the list, you first have to select (highlight) it. Then click on its name and drag it to a new location in the list. When dragging the project, a small arrow on the left margin indicates where it will be placed. You can also use the "**Move**" commands of the "**Edit**" menu to move the selected project line by line.

Note that if a separator is placed before the selected project, it is moved with the project.

2.1 Creating and Working with Projects

The commands of the project manager menu are used to create new projects, edit them and manage existing projects.

Creating a New Project

To create a new project, first enter its name. An empty project is then created, with no object in it. An I/O configuration can be attached to the new created project. This I/O configuration must be defined in library. If a configuration is chosen, ISaGRAF will automatically set-up the I/O connection and declare the corresponding I/O variables in the new project dictionary. When creating or renaming a project, you have to conform the following naming rules:

- name cannot exceed 8 characters
- the first character must be a letter
- the following characters can be letters, digits or underscore character

- the project's name is case insensitive

When a project is created, use the "**Edit** / **Set comment text**" command to enter the text to be displayed with the project name in the list.

Editing the project descriptor

The "**Project** / **Project descriptor**" command is used to edit the project text descriptor. This document fully identifies the project from the others on the project list. The project descriptor can also be used to record any remarks during the project lifetime.

Editing project

The "**File** / **Open**" command opens the Program Management window for the selected project. From this window, all the contents (programs, application parameters...) of the project, can be managed. It is also possible to double click on a project name, to edit it.

The history of modifications

The ISaGRAF system stores any modification relative to a component of a project in a history file. Each modification is identified in the history by a title, a date and a time. The history file contains the last 500 modifications. There is one history file for each project. The history of modifications for the project is the complement of the "diary" files attached to the programs of the project. The "Project / History" command allows the user to view or print the history of modifications for the selected project. The user can select one or more items in the main list, and press the following buttons:

OKcloses this window

Print ..... sends the contents of the list to the printer
Help..... displays help about this dialog box
[erase] Selected . removes (deletes) the selected lines from the list
[erase] All ..... clears the complete list
Find..... finds a pattern in the list

The input box above the "Find" button is used to enter a search pattern. This function is case insensitive. When the search reaches the bottom of the list, it continues from the top of the list to the starting position.

Printing a complete document

The "**Project** / **Print**" command allows the user to build and print a complete document about the selected project. This document can group any component (program, variable, parameters...) of the selected project. To build a specific (incomplete) document, the user only has to define its table of contents.

Password protection

The "**Project** / **Set password**" command enables the user to define password protection for tools and data of the selected project. Refer to the "**Password protection**" section, at the end of the first part in this manual for further information about password levels and data protection. Passwords are only relative to the

selected project. They have no influence on other projects and ISaGRAF libraries.

2.2 Working with several groups of projects

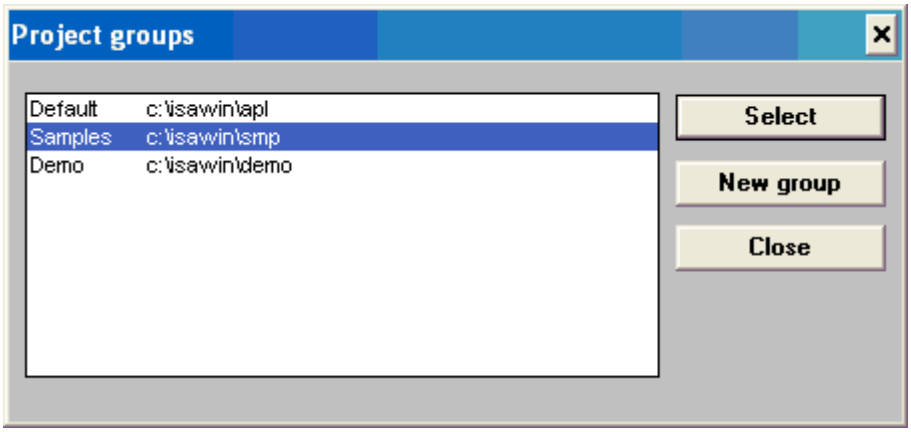
An ISaGRAF project corresponds to one directory on the disk, where all the project files are store. A "Project Group" corresponds to a list of project directories grouped together under the same root directory. A project group is identified by a name. As default, ISaGRAF creates two project groups:

"Default" ..... on "\ISAWIN\APL": your working area

"Samples"..... on "\ISAWIN\SMP": sample applications delivered with ISaGRAF workbench

The name of the currently selected project group is written in the toolbar, close to the button used to select a project group:

You can also run the "**File** / **Select project group**" to select an existing group or create a new one. The following dialog box is open:



Select a group in the list and press "Select" to activate it in the project management list. You can also double click on its name to select it. Use the "New group" command to create a new group. This command can be used either to assign a group name to an existing directory, or to create a new group with a new directory.

Note: No group can be selected or created when other ISaGRAF windows (program manager, editors...) are open.

2.3 Options

The commands of the "**Options**" menu are used to display or hide the toolbar, select the character font for text, and set the Project Manager "auto close" mode.

The character font selected is the one used to display the project descriptor, and is also used by all ISaGRAF text editors.

When the "**Keep Project Manager open**" option is removed, the Project Manager window is automatically closed when a project is entered.

2.4 Tools

The commands of the "**Tools**" menu are used to run other ISaGRAF applications. The "Tools / Archive Projects" command runs the ISaGRAF archive manager to save or restore projects. The "Tools / Archive Common data" command is used to save or restore files used by all projects (such as common defined words).

The "**Tools / Libraries**" command runs the ISaGRAF library manager in a separate window.

The "**Tools / Import IL program**" can be used to import a project described as a single IL program in a text file, according to PLC Open file exchange format.

3. Managing Programs

The Program Management window shows the programs (also called modules or programming units) of the application and groups into its menus the available commands, to create the project architecture, run editors, compiler and debugger.

This window is the workbench kernel during the development of an application. The Program Management window opens when running the "Open" command in the Project Management window.

3.1 The components of a project

The components of a project are called programs. A program is a logical entity that describes one part of the control execution. Global variables (such as I/O variables) can be used by any program in the application. Local variables may be used by only one program. Programs are listed in a hierarchy tree, divided into different logical sections. The window shows the programs and the links between them. The "Top level" programs appear on the left side of the hierarchy tree.

Top level programs

The top-level programs appear on the left side of the hierarchy tree. Top level programs of the three first sections are always active, and are executed in the following order, during the run time cycle (scan):

- (Read inputs)
- Execute the top level programs of the BEGIN section
- Execute the top level programs of the SEQUENTIAL section
- Execute the top level programs of the END section
- (Refresh outputs)

The programs of the "**Begin**" or "**End**" sections describe cyclic operations. They are not dependent on Time. The programs of the "Sequential" section describe sequential operations, where the Time variable explicitly appears to distinguish basic operations. The main programs of the "**Begin**" section are systematically executed at the beginning of each run time cycle. The main programs of the "**End**" section are systematically executed at the end of each run time cycle. The main programs of the "Sequential" section are executed on the basis of the SFC or FC rules and must be written in SFC or FC language. The programs of the cyclic sections cannot be described in the SFC or FC language. Any program of any section may own one or more sub-programs.

Functions and function blocks

The programs of the "**Functions**" section can be called by any program of any section in the project. A function is an algorithm that processes one output value from several input values. A function algorithm only works with volatile intermediate variables, erased from one call to the other. This implies that a function should never call a function block. A program of the "**Functions**" section cannot be described in the **SFC** or **FC** language.

Unlike functions, "**Function blocks**" associate an algorithm working on input values with hidden static data, which are copied (instanced) by the system on each different use of the function block. The programs of the "**Function Blocks**" section can be called by any program of any section in the project. They cannot be programmed in **SFC** or **FC** language.

Sub-programs

Sub-programs are functions dedicated to one (SFC, FC or other) father program. Asub-program can be executed (called) by its parent program only. Each program of each section may have one or more sub-programs. Any language apart from **SFC** and **FC** can be used to describe a sub-program.

Child SFC and FC programs

A **child SFC program** is a parallel program that can be started or killed by its parent program. The parent program and child program must both be described in **SFC** language.

When a parent program starts a child **SFC** program, it puts a **SFC token** into each initial step of the child program. When a parent program kills a child **SFC** program, it clears all the tokens existing in the steps of the child.

Any FC program of the sequential section may control other **FC** sub-programs. An **FC** father program is blocked (waits) during execution of an **FC** sub-program. It is not possible that simultaneous operations are done in father **FC** program and one of its **FC** sub-programs.

Links between programs and sub-programs

Sub-programs and child programs are linked to their parent program by a line in the hierarchy tree. An arrow ends a link between an SFC program and an SFC child program. Note that such a link represents **parallel** operations.

Programming languages

Each program is described in only one **language**. This language, selected when the program is created, cannot be changed afterwards. However, **FBD** diagrams may include parts in **LD**, and **LD** diagrams may include function block calls. Available graphic languages are **SFC** (Sequential Function Chart), **FC** (Flow Chart) **FBD** (Functional Block Diagram) and **LD** (Ladder Diagram). Available literal languages are **ST** (Structured Text) and **IL** (Instruction List). SFC and FC languages are reserved for main and child programs of the sequential section. The language of each program is shown as an icon beside the program name in the Program Management window. Below are the icons used to represent the languages:


SFC (Sequential Function Chart)


FC (Flow Chart)


FBD (Functional Block Diagram)


LD (Ladder Diagram (entered with Quick LD editor)


ST (Structured Text)


IL (Instruction List)

3.2 Working with programs

The "**File**" menu groups all the commands used to create, update or modify programs. It also launches appropriate editors to enter the contents of application programs.

Creating a new program

The "New" function of the "**File**" menu allows the creation of top level, child or subprograms into each program section. The first piece of information to be entered is the name of the new program according to the following naming rules:

- the maximum length of a name is 8 characters
- the first character must be a letter
- the following characters must be letters, digits or '_' character
- the naming of a program is case insensitive

Next, select the editing language chosen to describe the new program:

SFC - Sequential Function Chart
FC - Flow Chart
FBD - Functional Block Diagram (may include parts in LD)
LD - Ladder Diagram entered with Quick LD editor
ST - Structured Text
IL - Instruction List

Finally, select an execution style for the program:

Begin - top level of the "Begin" section
Sequential - top level of the "Sequential" section
End - top level of the "End" section
Function - in the "Functions" section
Function block - in the "Function Blocks" section
Child of - SFC or FC child or sub-program of an existing program.

By selecting one of the first five choices, the program is put at the top level of a Begin, End, Sequential, Functions or Function Blocks section. The selection of the latter indicates that the new program is an SFC child program or an FC subprogram or a sub-program. Remember that a top-level sequential program must be described in the SFC or FC language, and that the SFC and FC languages cannot be used for cyclic programs and their sub-programs.

Entering comments for each program

ISaGRAF allows you to attach a description text to each program of the project. This comment text is displayed with smaller character font beside the name of the program. Use the "**File** / **Program comment text**" command to enter or change the comment attached to the selected program.

Editing the contents of a program

This command allows the modification of a program's contents. The editor used to enter a program depends on the language chosen for that program. Program editing is carried out in individual windows, so that it is possible to edit more than one program through parallel windows. Pressing the ENTER key allows

the editing of the highlighted program. The user can also double click with the mouse on thename of the program to edit it.

Editing the "diary" file

A diary file is attached to each program. This is a text file, which contains all the notes about the modifications made to the program during its lifetime. The diary file can be edited, freely modified or printed at any time. When leaving the editor used to modify the source code of a program, a window is automatically opened to enter notes for the diary list. Such notes are inserted with the correct date and time into the diary file.

The dictionary of variables

The "**File** / **Dictionary**" command runs the dictionary editor, where are declared the variables of the project. Variables may be global (known by any program in the project) or local to the selected program. The dictionary editor may also be used to declare defined words, which are semantic aliases, used to replace a name or an expression in the source code of a program.

Parameters of a function, sub-program or function block

The "File / Parameters" command allows the user to define the call and return parameters of the selected sub-program, function or function block. This command has no effect if a main program of the "Begin" or "End" section, or an SFC program is selected in the Program Management window.

Sub-programs, functions or function blocks may have up to 32 parameters (input or output). A function or sub-program always has one (and only one) return parameter, which must have the same name as the function, in order to conform to ST language writing conventions.

The list in the upper left side of the window shows the parameters, in the order of the calling model: first the calling parameters, last the return parameters. The lower part of the window shows the detailed description of the parameter currently selected in the list. Any of the ISaGRAF data types may be used for a parameter.

The return parameters must be located after calling parameters in the list. Naming parameters must conform to the following rules:

- the length of the name cannot exceed 16 characters
- the first character must be a letter
- the following characters must be letters, digits or underscore character
- naming is case insensitive

The "**Insert**" command is used to insert a new parameter before the selected parameter. The "Delete" command is used to erase the selected parameter. The "**Arrange**" command automatically rearranges (sorts) the parameters, so that the return parameters are put at the end of the list.

Moving a program in the hierarchy tree

The "**Rename/move**" command of the "**File**" menu is used to change the name

of a program, or to move it into another section of the hierarchy tree. However the description language of an existing program cannot be changed. When running this command, the same window as the one used for creating programs is opened, and all fields are set up with the attributes of the selected program. The name of a program can be modified, and another section or parent program selected to move it into the hierarchy tree.

The "**Arrange programs**" command of the "**File**" menu is used to give an explicit order between a list of programs with same level and father. If the selected program is at the top level, the command is used to arrange the top-level programs of the selected section. If the selected program is at a lower level, the command arranges only the SFC children and sub-programs which have the same father as the selected one. When the "**Arrange programs**" dialog box is opened, select the program you want to move, and press the "**Up**" or "**Down**" button to move it in the list.

Copying programs

To make a copy of a program, select the source program from the list of programs, and run the "**File / Copy**" command. When running this command, the same window as that used for creating programs is opened, with all fields set up with the attributes of the selected program. Enter the name of the destination program and its location in the sections of the hierarchy tree. If the destination program does not exist, it is created at the specified location. If the destination program already exists, it is overwritten. All the local declarations and defined words are copied with the program. The description language of the destination program must be the same as the one used for the source program. Press the "**OK**" button to copy the program.

The "**Copy to other project**" command of the "**File**" menu copies the selected program into another project, with the same name. The child SFC programs and sub-programs of the selected program can be copied with it. The names of the selected program and its children must not be used in the target project. Programs cannot be overwritten by this command. All the attached local declarations and defined words are copied with the programs.

Deleting programs

To delete a program, first select it from the list of programs, and then run the "File / Delete" command. A program owning child or sub-programs cannot be deleted. In order to delete a program with child or sub-programs, the child or sub-programs must be deleted first. All the local declarations and defined words are deleted with the program.

Importing function or function block from library

The "**Tools / Import from library**" command is used to copy a function or a function block written in IEC language described in the library to the "**Functions**" or "**Function blocks**" section of the open project. Local variables and defined words attached to the imported function are copied with it. When a function has been correctly imported from the library, it can be placed in another section or another location in the hierarchy tree, using the "**File /**

Rename/Move" command. In order to avoid naming clashes, the imported function or function block must be renamed when imported in the project area. Don't forget to rename also the return parameter in the case of a function.

Exporting function or function block to library

The "**Tools / Export to library**" command is used to send a program of the "**Functions**" or "**Function blocks**" section (in the open project) to the appropriate library. Local variables and defined words attached, to the exported function or block, are copied with it. The exported function or block will have to be re-compiled (verified) from the ISaGRAF Library Manager, to ensure that it can be used in a library environment. Functions and function blocks of the library cannot use global variables.

3.3 Running the code generation tools

The commands of the "Make" menu are used to run the code generator, and to enter options and additional data used when producing the application code. Refer to the chapter "Using the code generator" in this document for further information about these tools.

Make the application code

The "**Make**" command starts the project code generation. The options for target code generation must be set correctly before running this command. Before generating the target code, any program that is still not verified is checked to detect the syntax errors. ISaGRAF includes an incremental compiler, which does not recompile programs, which have already been compiled.

Verify the selected program

The "**Verify**" command allows the user to verify the syntax of the program currently selected in the list. When a program is verified, with no error detected, it is not re-verified during the code generation until its contents or dependent defined words or variables change.

Simulating a modification

The "Touch" command simulates a modification of each program so that all of them will be compiled again during the next code generation.

Application run-time options

This command opens a dialog box where are entered the main run-time parameters for the execution of the application. This includes the cycle timing programming, run time error management, the starting mode and the hardware implementation of retained variables. Refer to the chapter "Using the Code Generator" in this document for more explanations about this command.

Compiler options

This command is used to set-up the options used by the ISaGRAF Code Generator to produce and optimise target code. Refer to the chapter "Using the Code Generator" in this document for more explanations about this command.

Defining resources

A "resource" is a user defined data (for example a file) which has to be merged with the target code so it can be downloaded with it. Refer to the section "Using the Code Generator" in this document for more explanations about the format of the resource definition file.

3.4 Other ISaGRAF tools

The "**Project**" menu groups the commands that run ISaGRAF tools for the selected project. Refer to the corresponding chapters in this document for more information about these tools.

Compiler options

This command is used to set-up the options used by the ISaGRAF Code Generator to produce and optimise target code. Refer to the chapter "Using the Code Generator" in this document for more explanations about this command.

Wiring I/O variables

The "IO connection" command runs the ISaGRAF I/O variable connection editor. This tool is used to establish the relationship between I/O variables declared in the project dictionary and corresponding I/O hardware.

Running the cross reference editor

The "**Cross references**" command allows the user to calculate, to view or to print the cross references of the project. The cross-references show the user all the occurrences of each variable in the source code of the programs, in the entire project. This function is very useful to detect an access to a variable or any global resource, or to list all the occurrences of a global variable in the source code.

Entering the project descriptor

The "Project descriptor" command is used to edit the project text descriptor. This document fully identifies the project from the others on the project list. The project descriptor can also be used to record any remarks during the project lifetime. The project descriptor is the one displayed in the Project Manager window.

Printing a complete document

The "**Print project document**" command allows the user to build and print a complete document about the selected project. This document can group any component (program, variable, parameters...) of the selected project. To build a specific (non-complete) document, the user only has to define its table of contents.

History of modifications

This command opens a dialog box where is displayed the history of modifications for the project. Refer to the chapter "Managing projects" in this document for more explanations about this command.

3.5 Adding commands to the Tools menu

ISaGRAF provides a method to insert other commands in the "**Tools**" menu. User defined commands to be added are described in **"\ISAWIN\COM\ISA.MNU**" text file.

You can add up to 10 commands. Comments may be inserted on any line, beginning with ";" character. Each command is described on two lines of text, according to the following syntax:

```
M=menu_string
C=command_line
```

The menu string is the text to be displayed in the "**Tools**" menu. The command line is any MS-DOS or Windows executable, and can be completed with arguments. In command line, you can use **"%A"** characters to replace the name of the open project, and **"%P"** characters to replace the name of the selected program. The following example runs "Notepad" to edit the selected program (to be used with ST and IL programs):

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

3.6 Simulating and debugging the application

The command of the "Debug" menu are used to run the ISaGRAF graphic debugger, either in simulation mode or in real connected mode.

Simulation

The "Simulate" command opens the debugger in simulation mode. In this mode, another window appears, called the simulator. This command is very useful to test any application when the target machine is unavailable. Starting the simulator closes the Program Management window. The Program Management window is then re-opened in debug mode after both debugger and simulation windows are opened. The simulator cannot be started if the target code has not been generated.

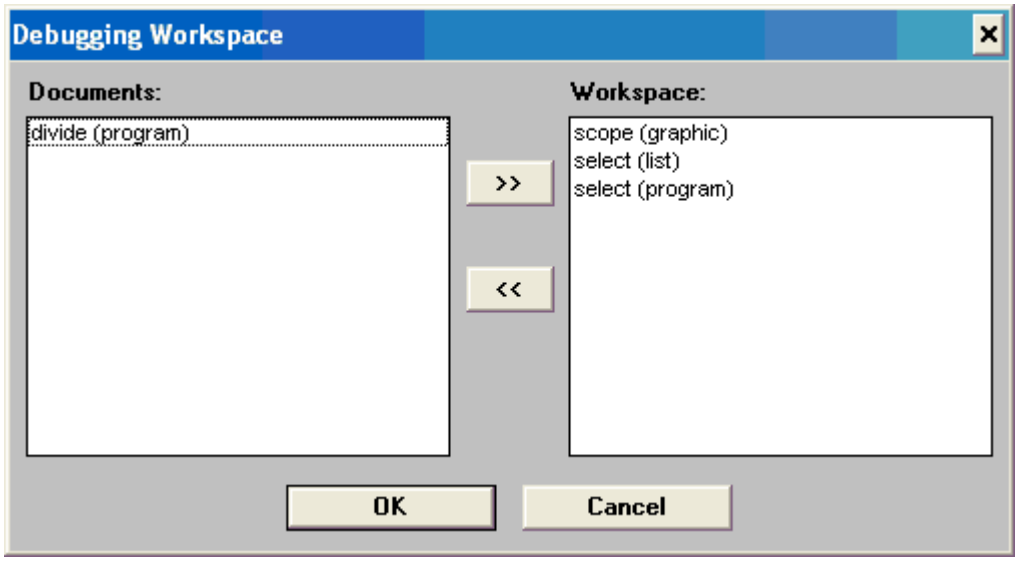
The simulator cannot be started when child windows (editors, code generation, I/O connection...) are opened. Each of them must be closed before running this command. This command is also available from menus of ISaGRAF editors.

Real debugging

The "**Debug**" command opens the debugger main window, and closes the Program Management window. The Program Management window is then re-opened in debug mode as soon as communication is established between the debugger and the target application. The debugger cannot be started if the target code has not been generated. The debugger cannot be started when child windows (editors, code generation, I/O connection...) are opened. Each of them must be closed before running this command. This command is also available from menus of ISaGRAF editors.

Preparing the debug workspace

The "**Debug / Workspace**" command enables you to define a list of documents for initial workspace. Such documents can be programs, SpotLight graphics, and lists of variables. Graphics and lists of time diagrams from previous ISaGRAF versions are also listed with project documents. Documents defined in the initial workspace are automatically opened when simulation or On Line monitoring is launched.



The dialog box shows the existing documents of the project on the left, and documents selected for the initial workspace on the right. Use ">>" and "<<" push buttons to move documents from one list to the other. Each project has its own list of documents for initial workspace.

Link set-up

The "**Link set-up**" command It enables the user to define the parameters of the link used for communication between the debugger on the host PC and the target ISaGRAF system.

The "**Slave number**" identifies the target ISaGRAF system or task. It can be

from **1** to **255**. Refer to the target supplier manual for the slave number of the target system used.

The "**Communication port**" identifies the hardware media between ISaGRAF workbench and target. It can be either the name of a serial port, or "**Ethernet**", reserved TCP-IP communication using the "Winsock" Version 1.1.

The "**Time out**" is the time left to the target system for its communication operations between the end of a debugger question and the beginning of its response. This time is set as a number in milliseconds. The "Retries" field is the number of automatic trials the debugger executes for a communication operation before detecting a communication error.

Serial link set-up

When a serial port (COM1..4) is selected, the "**Set-up**" button is used to access to other serial link communication parameters.

The transmission baud rate, parity and format may be selected. When the "hardware" choice is selected for "**Flow Control**", the ISaGRAF Workbench controls the CTS and DSR lines to enable hardware handshaking during exchanges.

Ethernet link set-up

When "Ethernet" is selected as a communication port, the "**Set-up**" button is used to enter the "**Internet Address**" and "**Internet port**" number, for TCP-IP communication.

These fields use the standard formats defined by the Socket interface. The Workbench uses the WINSOCK.DLL Version 1.1 library for TCP-IP communications. This file must be correctly installed on the hard disk. "**1100**" is the default port number used if not specified when running the ISaGRAF target.

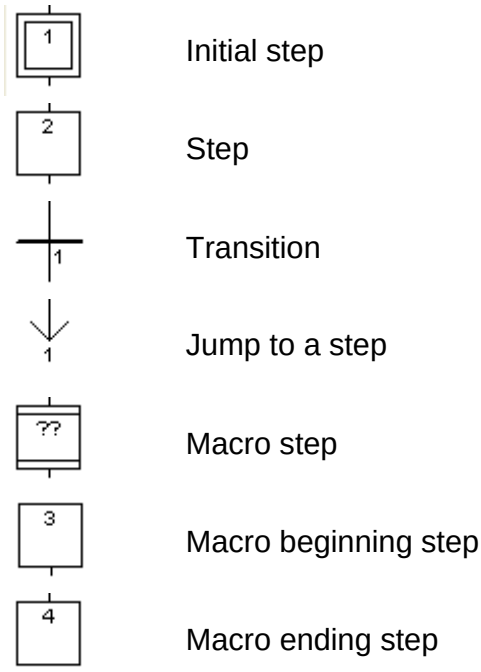
4. Using the SFC editor

The SFC language is used to describe operations of a sequential process. It uses a simple graphic representation for the different steps of a process, and conditions that enable the change of active steps. An SFC program is entered by using the ISaGRAF graphic SFC editor. SFC is the core of the IEC 1131-3 standard. The other languages usually describe the actions within the steps and the logical conditions for the transitions. The ISaGRAF graphic SFC editor allows the user to enter complete SFC programs. It combines graphic and text editing capabilities, thus allowing the entry of both the SFC chart, and the corresponding actions and conditions.

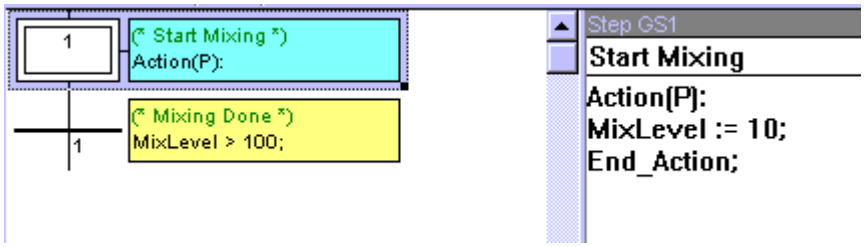
4.1 SFC language main topics

The SFC language is used to represent sequential processes. It divides the process cycle into a number of well-defined successive **steps** (self-contained situations), separated by **transitions**. Refer to the ISaGRAF Languages Reference Manual for more details on the SFC language.

SFC components are joined by **oriented lines**. The default orientation of a line is up to down. These are the basic graphic components used to build an SFC chart:



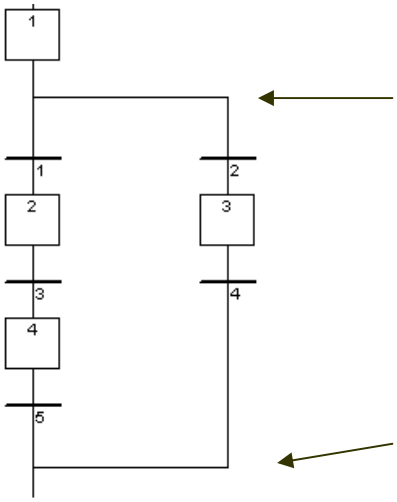
The SFC programming is usually separated into two different levels: The **Level 1** shows the graphic chart, reference numbers of the steps and the transitions, and comments attached to the steps and the transitions. The **Level 2** is the **ST** or **IL** programming of the actions within the steps, or the conditions attached to the transitions. Actions or conditions may refer to sub-programs written in other languages (FBD, LD, ST or IL). Below is an example of level 1 and level 2 programming:



The level 2 programming of a step is entered in a text editor. It can include action blocks programmed in ST or IL. The level 2 programming of a transition can be entered either in IL or ST text languages, or with Quick LD editor.

Divergences and convergences

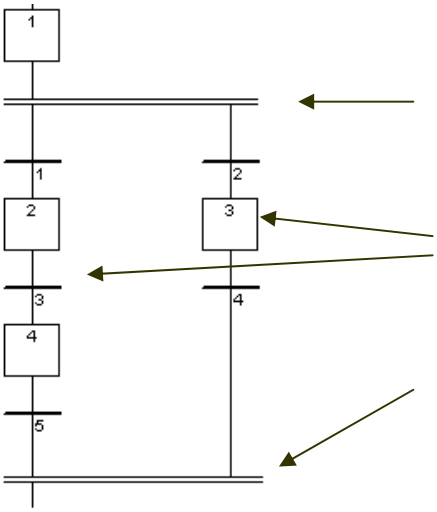
Divergences and convergences are used to represent **multiple links** between steps and transitions. Simple divergences or convergences represent different **inclusive** possibilities between different sub parts of the process:



Single divergence (OR)
Warning: following transitions are not implicitely exclusive

Single convergence (OR)

Double divergences represent parallel processes:



Double divergence (AND)

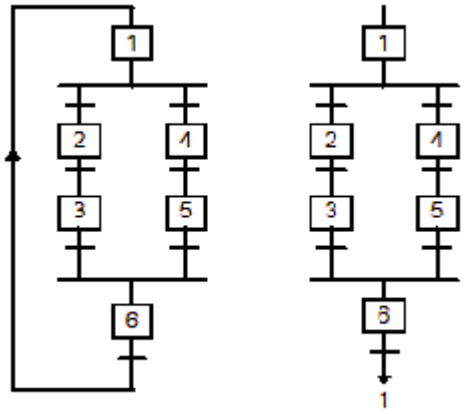
These are parallel processes

Double convergence (AND)

Jump to a step

The SFC editor only allows the user to draw links in the up to down direction. A jump to a step can be used to represent a link to an upper part of the chart.

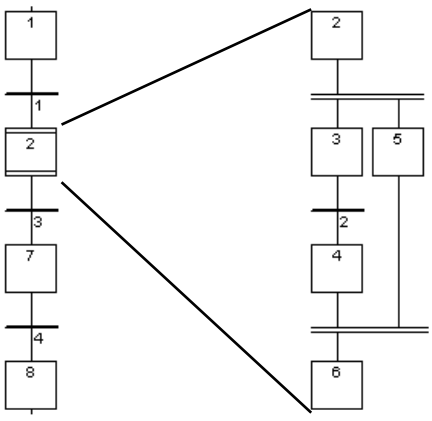
Following charts are equivalent:



Jump to a transition is forbidden, and must be explicitly represented as a double(AND) convergence.

Macro steps

A macro step is a **unique** representation of a **stand-alone** group of steps and transitions. A macro step begins with a **beginning step** and terminates with an **ending step**.



The detailed representation of a macro step must be described in the same SFC program. The macro-step symbol must have the same **reference number** as the macro beginning step. A macro step description may contain another macro step.

4.2 Entering an SFC chart

To draw an SFC chart, the user simply has to introduce the significant components of the chart. All the single lines joining two elements (horizontally or vertically) are drawn automatically by the SFC editor. To place an SFC component on the chart, the user has to move the selection to appropriate location and select the type of the component in the

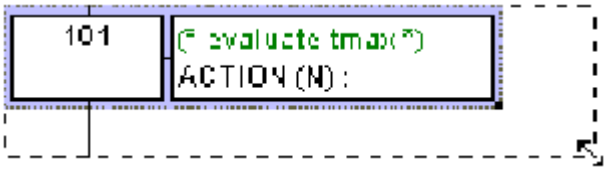
editor toolbar. The symbol is inserted at the current position. The following keyboard sequences can also be used:

F2:	Insert an initial step
F3:	Insert a single step
F4:	Insert a transition
F5:	Insert a jump to a step
F6: F7:	Insert an OR divergence or convergence / Add branches
↑F6: ↑F7:	Insert an AND divergence or convergence / Add branches
F8:	Insert a macro step
F9: ↑F9:	Insert begin or end step for the body of a macro step

(The "↑" symbol indicates a combination with SHIFT key)

The **editing grid** shows **matrix cells**. An editor option allows the user to show or hide the grid during chart input. The grid is very useful for initial layout of SFC chart, or selecting sub-parts of the chart. Use the "**Options / Layout**" command to display or hide the grid.

The ISaGRAF SFC editor always shows the current position in the matrix. The focused cell is marked in grey. The small square on its bottom right corner can be used to freely resize the cells. The X/Y ratio of the cells can also be changed this way.



Creating a divergence or convergence

Divergences and convergences are always drawn **from the left to the right**. To draw a divergence or a convergence, its **left** branches has to be placed on the chart area. The type of drawing (simple or double) is set by selecting one of these buttons in the toolbar.

F6: F7:	Insert an OR divergence or convergence / Add branches
↑F6: ↑F7:	Insert an AND divergence or convergence / Add branches

Adding branches to divergences

The **start** and **stop** position of each **auxiliary** branch is placed on the divergence or convergence line using these buttons in the toolbar. The left corner of the divergence or convergence must be present before inserting new branches. The right corners have the

same style (simple or double) as the main left corner. Right corners cannot be placed if the main left corner has not been added.

Insert an OR divergence or convergence / Add branches

Insert an AND divergence or convergence / Add branches

Inserting a macro step

This button is used to insert a macro step in the main chart. The body of the macro step must be entered elsewhere in the same SFC program.

Body of a macro step

Macro steps must be described in the same SFC program as the main chart. A macro step must start with a **beginning step** and stop with an **ending step**. The sub-chart described as the macro implementation must be **self-contained**. The macro beginning step must have the same **reference** as the macro-step symbol of the main branch.

4.3 Working on an existing SFC chart

You can use either the mouse or keyboards arrows to select a rectangle area in the chart. The whole selected area is marked in grey. The commands of the "**Edit**" menu can then used:

Cut / copy / delete / paste commands

The following commands are available from the "**Edit**" menu when the "**arrow**" button is selected in the editor toolbar:

Cut ..... Move selected rectangle from the screen to the SFC clipboard

Copy..... Copy selected rectangle from the screen to the SFC clipboard

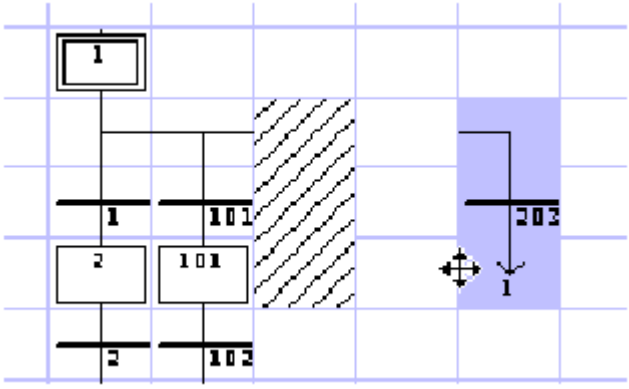
Delete..... Clear (delete) selected rectangle

Paste..... Insert contents SFC clipboard at the current position

The "**Edit / Paste**" copies SFC clipboard to the screen. Copy / Paste commands work on both SFC chart and step/transition level 2 programming. It is also possible to copy a chart in a program and paste it in another SFC program. Elements are inserted before the currently selected position.

Move elements

When SFC elements are selected in the SFC chart, you can move them to another location of the chart by dragging the selection with the mouse. While you drag the selection, the initial location of selected elements is hatched.



The destination area for moved elements must be empty. No insertion is possible while moving SFC symbols.

Renumbering steps and transitions

Each step or transition is identified by a logical number in the SFC chart. The "**Edit/Renumber**" command allows the user to automatically set up numerically sequential reference numbers for any of the steps and the transitions of the currently edited SFC program. When a step number is changed, all the jumps to this step are automatically updated with the new reference number. (This also applies to macro steps and beginning steps)

Direct access to a step or transition

The "**Edit / Go to**" command allows the user to access an existing step or transition. The scrolling position is automatically adapted so that the step or transition is visible.

Find and replace texts

The "**Edit / Find Replace**" command can be used to find or replace text strings in the complete program (all steps and transitions). The Find/Replace dialog box is used to enter a searched text and directly open the level 2 programming section where text occurs.

4.4 Entering the level 2 programming

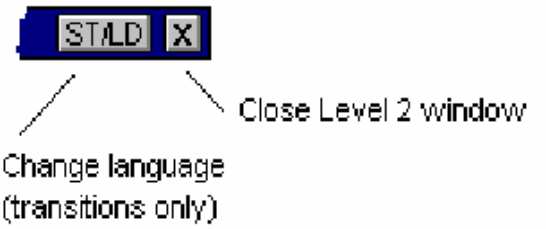
To enter the Level 2 text, the user must double click on the step or transition symbol. The level 2 programming is displayed on the right of the SFC window. The separation line between SFC chart and level 2 programming can be freely moved. You can open one or two level 2 areas at the same time. The following commands are available from keyboard, mouse or the "Edit" menu:

	Keyboard	***Mouse***	***"Edit" menu***
Open in last default window	Enter	Double Click	Edit level 2
Open in separate window	Ctrl+Enter	Ctrl + DoubleClick	Edit Level 2 in separate window

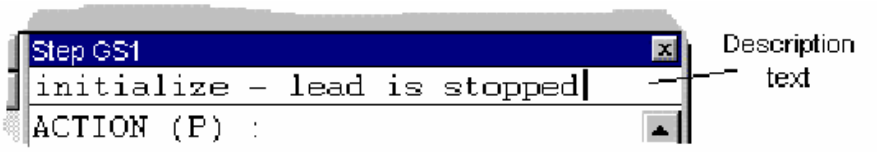
When two level 2 windows are visible, the separation between them can be freely moved. The button on the right of the level 2 title bar is used to close a level 2 window.

The default language for Level 2 programming is ST (Structured Text). For

transitions, level 2 programming can also be entered with Quick LD editor. Use the "ST/LD" button in level 2 title bar to change the active language. This command is valid only if the level 2 programming window is empty.



A single line edit box appears at the top of the level 2 window. It is used to enter a short description text. This text will be displayed as an IEC comment in drawing of SFC symbols. It is very useful as it is used by other commands such as "Go To..." and also in the SFC printout to document SFC steps and transitions.



The "**Options / Refresh**" command can be used at any time when level 2 windowsare open to refresh the main SFC chart with modified level 2 programs.

Inserting a variable name

When programming in text language, press this button to select a variable declared in the project dictionary and insert its name at the current position of the caret.

When programming in Quick LD, press this button to select the variable to be attached to the selected contact or block I/O parameter.

Inserting a Pulse action block in step

When programming the level 2 of a step, press this button to insert the template of a Pulse action block at the current position of the caret. Below is the format of a Pulse action block:

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

Pulse actions are instructions, which are executed only once when the step becomes active. Refer to the ISaGRAF language reference for further details on SFC programming.

Inserting a Non stored action block in step

When programming the level 2 of a step, press this button to insert the template of a Non stored action block at the current position of the caret. Below is the format of a Non stored action block:

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

Non stored actions are instructions which are executed on every PLC cycle when the step is active. Refer to the ISaGRAF language reference for further details on SFC programming.

New P0 and P1 action qualifiers

ISaGRAF supports new **P0** and **P1** action qualifiers. When programming the level 2 of a step, press these buttons to insert the template of a P0 or P1 action block at the current position of the caret. Below is the format of such blocks:

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

P1 actions are instructions which are executed only once when the step becomes active (same as Pulse). P0 actions are instructions, which are executed only once when the step becomes inactive. Refer to the ISaGRAF language reference for further details on SFC programming.

Boolean actions

Other text semantics are available to directly act on a boolean variable according to the step activity. Such actions consist of attaching the step activity signal to an internal or output boolean variable. This is the syntax of the basic boolean actions:

<boolean_variable> (N);	assigns the step activity signal to the variable
<boolean_variable>;	same effect (N attribute is optional)
/ <boolean_variable>;	assigns the negation of the step activity signal to the variable

Other features are available to set or reset a boolean variable, when the step becomes active. This is the syntax of set and reset boolean actions:

<boolean_variable> (S);	sets the variable to TRUE when the step activity signal becomes TRUE
<boolean_variable> (R);	resets the variable to FALSE when the step activity signal becomes TRUE

SFC actions

Other text semantics are available to control the execution of a child SFC program. An SFC action is a child SFC sequence, started or killed according to the condition of the step activity signal. An SFC action can have the N (Non stored), S (Set), or R (Reset) qualifier. This is the syntax of the basic SFC actions:

<child_program> (N);	starts the child sequence when the step becomes active, and kills the child sequence when the step becomes inactive
<child_program>;	same effect as the preceding syntax (N attribute is optional)
<child_program> (S);	starts the child sequence when the step becomes active - nothing is done when the step becomes inactive
<child_program> (R);	kills the child sequence when the step becomes active - nothing is done when the step becomes inactive

The SFC sequence specified as an action must be an existing child SFC program of

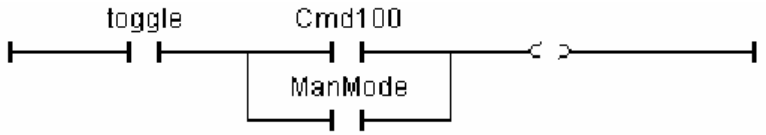
the currently edited program, created with the ISaGRAF program manager.

Transitions written in ST

The level 2 of a transition is a boolean expression. To program it in ST language, just enter the boolean condition according to the ST syntax. Optionally, a semicolon may be added at the end of the expression.

Transitions written in Quick Ladder

Quick LD editor is available to program the level 2 condition of a transition. In this case, the diagram is made of just one rung, with only one coil, which represents the transition. The name of the transition is not repeated with the coil symbol. Below is an example of transition condition programmed in Quick LD.



When programming in Quick LD, use the keyboard arrows to move the selection in the programming logical grid, and then use the following shortcuts to insert symbols:

F2	insert a contact after the selected symbol / initiate the rung
F3	insert a contact before the selected symbol
F4	insert a contact in parallel with the selected symbol
F6	insert a block after the selected symbol
F7	insert a block before the selected symbol
F8	insert a block in parallel with the selected symbol

You can also click on the function key bar at the bottom of the level 2 window instead of hitting function keys.

Hit RETURN when the selection is on a contact or a block I/O parameter to select a variable or enter a constant value. Hit RETURN when the selection is on a function block to select the type of the function block. You can also double click on a symbol for the same effect.

Hit SPACE bar when a contact is selected to change the type of contact (direct, negated or with pulse detection). Refer to the chapter "Using the Quick LD editor" in this document for more details about Quick LD capabilities.

4.5 Using the SFC gallery

The ISaGRAF SFC editor manages an SFC gallery: it is a collection of SFC structures that can be inserted in any SFC chart. Elements of the SFC gallery can optionally embed the level 2 programming of steps and transitions. Use the following commands of the "Tools" menu:

Copy to SFC	gallery copy selected elements to SFC gallery
Paste from SFC	gallery paste an SFC gallery element at the current location

When copying to SFC gallery (i.e. creating a new SFC gallery element), you can optionally ask to embed level 2 programming of selected SFC symbols.

5. Using the Flow Chart editor

The ISaGRAF Flow Chart graphic editor allows the user to enter complete FC (Flow Chart) programs, with actions and tests (decisions) programmed in either ST, IL or Quick LD language. Flow Chart is a decision diagram, which can also be used to describe sequential operations as it enables some advanced features such as non-blocking backward jumps.

5.1 Basics of the FC language

Flow Chart (FC) is a graphic language used to describe sequential operations. A Flow Chart diagram is composed of Actions and Tests. Between Actions and tests are oriented links representing data flow. Below are graphic components of the Flow Chart language:

Beginning of FC chart:

A "begin" symbol must appear at the beginning of a Flow Chart program. It is unique and cannot be omitted. It represents the initial state of the chart when it is activated.

Ending of FC chart:

An "end" symbol must appear at the end of a Flow Chart program. It is unique and cannot be omitted. It is possible that no connection is drawn to the "End" symbol (always looping chart), but "End" symbol is still drawn anyway at the bottom of the chart. It represents the final state of the chart, when its execution has been completed.

FC flow links:

A flow link is a line that represents a flow between two points of the diagram. A link is always terminated by an arrow. Two links cannot start from the same source connection point.

FC actions:

An action symbol represents actions to be performed. An action is identified by a number and a name. Two different objects of the same chart cannot have the same name or logical number. Programming language for an action can be ST, LD or IL. An action is always connected with links, one arriving to it, one starting from it.

FC tests:

A test represents a boolean condition. A test is identified by a number and a name. According to the evaluation of attached ST, LD or IL expression, the flow is directed to "YES" or "NO" path. When programmed in ST text, the expression may optionally be followed by a semicolon. When programmed in LD, the unique coil represents the condition value.

FC sub-program:

The system enables the description of a hierarchised structure of FC programs. FC programs are organised in a hierarchy tree. Each FC program can call other FC programs. Such a program is called a child program of the FC program, which calls it. FC programs, which call FC sub-programs, are called father

program. FC programs are linked together into a main hierarchy tree, using a "father - child" relation. A sub-program symbol in a Flow Chart represents a call to a Flow Chart sub-program. Execution of the calling FC program is suspended till the sub-program execution is complete.

FC I/O specific action:

An I/O specific action symbol represents actions to be performed. As other actions, an I/O specific action is identified by a number and a name. The same semantic is used on standard actions and I/O specific actions. The aim of I/O specific actions is only to make the chart more readable and to give focus on non-portable parts of the chart. Using I/O specific actions is an optional feature. I/O specific blocks have exactly the same behaviour as standard actions.

FC connectors:

Connectors are used to represent a link between two points of the diagram without drawing it. A connector is represented as a circle and is connected to the source of the flow. The drawing of the connector is completed, on the appropriate side (depending on the direction of the data flow), by the identification of the target point (generally the name of the target symbol). A connector always targets a defined Flow Chart symbol. The destination symbol is identified by its logical number.

FC comments:

A comment block contains text that has no sense for the semantic of the chart. It can be inserted anywhere on a free space of the Flow Chart document window, and is used to document the program.

5.2 Entering a Flow Chart

To enter a chart, you have to place elements (actions, decision tests, connectors...) in the graphic area, and draw flow links between them.

Inserting objects

To insert an object in the diagram, select the corresponding button in the toolbar and click in the graphic area, where you want to insert it. You can either put the element on an empty area, or insert it in a flow by clicking on a flow link. Insertion on a link is allowed for top to bottom vertical links only. You can insert the following basic elements:


action programmed in ST, IL or Quick LD


I/O specific action (highlights a particular non-portable action)


test (decision) programmed in ST, IL or Quick LD


connector


call to an FC sub-program


comment (description text)

The ISaGRAF Flow Chart editor also proposes you a list of classical Flow Chart structures. Such structures can only be inserted on an existing flow link. They cannot be put in an empty area:

 If / Then / Else (binary selection)

 Repeat until (waits for a condition)

 While (loops while a condition is true)

Selecting objects

Selecting graphic objects is needed for most of the editing commands. The ISaGRAF FC graphic editor enables the selection of one or more objects existing in the diagram area. To select objects, the "select" (button with an arrow) choice must be checked in the editor toolbar. To select one object, the user only has to click on its symbol.

To select a list of objects, drag the mouse in the diagram to draw a rectangle area. All graphic objects in the selection rectangle are marked as "selected".

A selected object is drawn in dark blue colour, with little black squares around its graphic symbol. It is also possible to add or remove one object to a multiple selection, by clicking on its symbol with Shift or Ctrl key pressed.

By making a new selection, selection of all objects previously selected is removed. To remove the existing selection, simply click with the mouse in an empty area, outside of the rectangle which borders the selected objects.

For single selection, it is possible to use keyboard arrows to move selection from one object to the other in the chart. Flow links can also be selected.

Inserting comments

Comments may be inserted anywhere in an empty part of the diagram. Comments have no influence on the program execution. They allow a higher readability of the diagram. To insert a comment block, select the corresponding button in the toolbar, and click in the diagram where comment must be put. Double click on a comment to enter / change its text. No special leading or trailing characters such as "(*" and "*)" are needed when entering the text of a comment block. A comment block may be resized by dragging the corners of its border when it is selected.

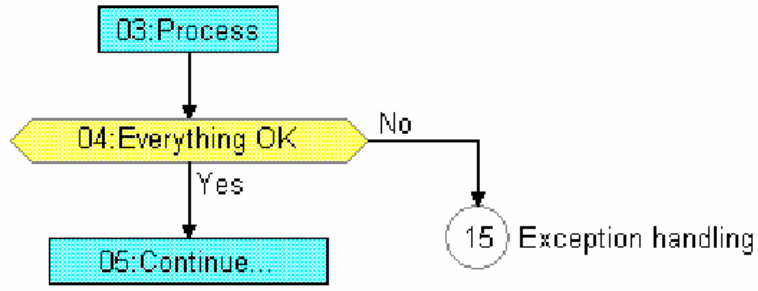
Drawing flow links

Select this button in the toolbar to draw a flow link between existing elements. A link must always be drawn in the direction of the flow. First select a non-connected output point of an FC element, and drag the mouse to the destination point to insert the link. The destination point can either be the top (input point) of a non-connected FC element, or any location on an existing link. Convergence points between links are shown with small grey circles in the Flow Chart. Convergence points can also be selected and moved in order to arrange the diagram.

Using connectors

The ISaGRAF Flow Chart editor enables the use of graphic connectors, as a replacement of a visible flow link. Connectors can be very useful to avoid very long links and increase chart readability. A connector cannot be used to establish a link with another FC program.

A connector is put in the chart as other FC objects. It is represented by a circle containing the numerical reference of targeted element (destination of the flow link). The short description text of the target element is displayed close to the connector circle.



Moving objects

To move objects in the chart, you have to select them, and drag the mouse to move them within the chart. You can either move a single element or a multiple selection. Elements cannot be overlapped when moving them. Moving elements cannot be used to connect them to an existing link.

When a single element (action, test...) is moved, the ISaGRAF Flow Chart editor automatically moves with the selected element all objects placed below and connected to it. This feature does not operate in the case of a multiple selection.

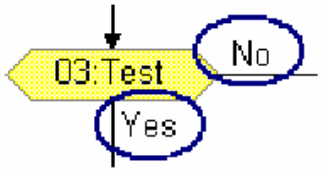
Resizing objects

Any graphic element of a flow apart from "Begin", "End" symbols and connectors can be resized freely. To resize an element, you first have to select it. Then drag with the mouse the small squares drawn on its border to change its size.

When an element is connected to a flow link, resizing it horizontally acts on both left and right borders, so that the element is still correctly centred on the link when resized.

Swapping the outputs of a test

You can swap locations of YES / NO outputs on a test (decision). To do that, simply double click on either "Yes" or "No" marks displayed close to the test symbol.



5.3 Working on an existing chart

The commands of the "**Edit**" menu are used to change or complete an existing diagram. Most of these commands act on the elements currently selected in the diagram.

Correcting a chart

The **DEL** key can be used to remove the selected elements. Pending links are deleted with selected elements. Use "**Edit / Undo**" command to restore elements after a **DEL** command. The **DEL** command can also be applied to a group of elements selected in the diagram. The "**Cut**", "**Copy**", "**Paste**" commands of the **Edit**" menu are used to move or copy selected elements.

Find and replace

The "**Edit / Find Replace**" commands can be used to find or replace text strings in he complete program (all actions and tests programmed in ST, IL or Quick LD). he Find/Replace dialog box is used to enter a text to be searched and to directly pen the programming section where the text is found.

Direct access to an element

The "**Edit Go to**" command allows the user to access a graphic element existing in he chart. The scrolling position is automatically adapted so that the element is isible. The element, when reached, is selected.

Renumbering elements

The "**Edit / Renumber**" command is used to renumber elements of the Flow Chart. ny FC element put in the chart is identified with a unique reference number.Reference numbers are allocated by the editor each time new elements are inserted. The "**Renumber**" allows you to re-adjust element numbering according to heir location in the chart. Growing numbering is performed from top to bottom and from left to right.

5.4 Entering level 2 programs

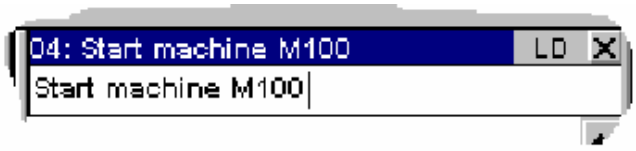
To enter the level 2 program, the user must double click on the action or test symbol. The level 2 programming is displayed on the right of the FC window. The separation line between FC chart and level 2 programming can be freely moved. You can open one or two level 2 areas at the same time. The following commands are available from keyboard, mouse or the "Edit" menu:

	Keyboard	***Mouse***	***"Edit" menu***
Open in last default window	Enter	Double Click	Edit level 2
Open in separate window	Ctrl+Enter	Ctrl + DoubleClick	Edit Level 2 in separatewindow

When two level 2 windows are visible, the separation between them can be freely moved. The button on the right of the level 2 title bar is used to close a level 2 window.

The default language for Level 2 programming is **ST** (Structured Text). The programming language can also be **IL** or Quick **LD**. The name of the selected language is displayed in a small box in the level 2 title bar. Run the "**Options / Set Level 2 language**" command from menus or click on that box to change the active language. This command is valid only if the level 2 programming window is empty.

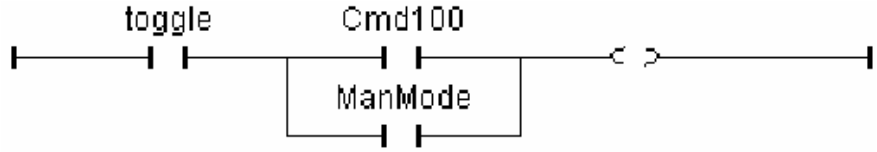
A single line edit box appears at the top of the level 2 window. It is used to enter a short description text. This text will be displayed as an IEC comment in the drawing of FC symbols. It is very useful as it is used by other commands such as "Go To..." and also in the FC printout to document FC actions and tests.



The "**Options / Refresh**" command can be used at any time when level 2 windows are open to refresh the main FC chart with modified level 2 programs.

5.5 Programming level 2 with Quick LD

Quick LD editor is available for level 2 programming. In the case of a decision test, the LD diagram is made of just one rung, with only one coil, which represents the decision. The name of the test is not repeated with the coil symbol. Below is an example of a test programmed in Quick LD.



When programming in Quick LD, use the keyboard arrows to move the selection in the programming logical grid, and then use the following shortcuts to insert symbols:

F2:..... insert a contact after the selected symbol / initiate the rung

F3:..... insert a contact before the selected symbol

F4:..... insert a contact in parallel with the selected symbol

F5:..... add a coil in parallel with the selected one (not for tests)

F6:..... insert a block after the selected symbol

F7:..... insert a block before the selected symbol

F8:..... insert a block in parallel with the selected symbol

F9:..... add a jump symbol in parallel with the selected coil (not for tests)

A jump leads to a rung name. The name of a rung can be entered by hitting ENTER when selection is on the rung head. The ISaGRAF editor keeps the memory of the rung labels you already entered, whether it has been specified for a rung name or a jump operation. The "Jump/Label"' dialog box gives you the possibility either to enter a new label, or to select an existing one. If you enter a new name, it will automatically be added to the list.

The "**Remove**" button is used to remove the selected name from the list. It does not remove the label on the rung you selected in the diagram. To do this, just press **OK** when the edit box is empty.

You can also press buttons in the LD toolbar instead of hitting function keys.

Hit ENTER when the selection is on a contact or a block I/O parameter to select a variable or enter a constant value. Hit ENTER when the selection is on a function block to select the type of the function block. You can also double click on a symbol for the same effect.

Hit Control + SPACE bar when a contact is selected to change the type of contact or coil (direct, negated). Refer to the chapter "Using the Quick LD editor" in this document for more details about Quick LD capabilities.

5.6 Display options

The "**Options / Layout**" command opens a dialog box where are grouped all the parameters and options concerning the editor workspace and the drawing of the diagram. Use the check boxes in the "Workspace" group to display or hide editor toolbars and status bar. Option of the "Document" group allow you to show or hide points of the editing grid and to display chart either in black and white or with colours.

Use the "Zoom" button of the toolbar to change current zoom ratio. This command is also available when working on a Quick LD program attached to an action or a test.

Use the "Grid" button of the toolbar to show or hide points of the editing grid. This command is also available when working on a Quick LD program attached to an action or a test.

Use the "**Options / Font**" command to select the name of the character font to be used in all ISaGRAF documents. When called from an ST or IL block, you can specify size of the font. When selecting font for a graphic view (FC or Quick LD), font style and size are not relevant and do not need to be specified. ISaGRAF graphic editors always calculate the font size according to the current zoom ratio.

6. Using the Quick LD editor

The LD language enables graphic representation of boolean expressions. Boolean AND, OR, NOT operators are explicitly represented by the diagram topology. Boolean input variables are attached to graphic contacts. Boolean output variables are attached to graphic coils. The ISaGRAF Quick LD editor provides easy LD diagram entering using either keyboard or mouse. Elements are automatically linked and arranged on rungs by the Quick LD editor. No connection is drawn manually by the user. The Quick LD editor also arranges rungs in the diagram so that the space filled by the diagram is always optimised.

6.1 Basics of the LD language

An LD program is expressed as a list of **rungs** where contacts and coils are arranged. Below are the basic components of an LD diagram:

Rung head (left power rail)

Each rung begins with a left power rail, which represents the initial "TRUE" state. SaGRAF Quick LD editor automatically creates the left power rail when the first contact of the rung is placed by the user. Each rung may have a logical name, which can be used as a label for jump instructions.

Contacts

A contact modifies the boolean data flow, according to the state of a boolean variable. The name of the variable is displayed upon the contact symbol. The following types of contacts are supported by ISaGRAF Quick LD editor:

 direct contact

 negated contact

 contact with positive (rising) edge detection

 contact with negative (falling) edge detection

Coils

A coil represents an action. The rung state (state of the link on the left of the coil) is used to force a boolean variable. The name of the variable is displayed upon the coil symbol. The following types of coils are supported by ISaGRAF Quick LD editor:

 direct coil

 negated coil

 "set" action coil

 "reset" action coil

 coil with positive (rising) edge detection

 coil with negative (falling) edge detection

Function blocks

A block in an LD diagram can represent a function, a function block, a sub-program or an operator. Its first input and output parameters are always connected to the rung. Other input and output parameters are literally written outside of the block rectangle.

Rung end (right power rail)

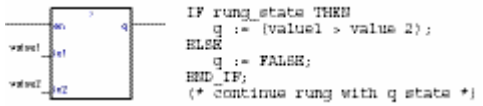
A rung ends with a right power rail. Using the Quick LD editor, the right power rail is automatically inserted when a coil is placed by the user.

Jump symbol

A jump symbol always refers to a rung label, i.e. the name of a rung defined somewhere in the same LD diagram. It is placed at the end of a rung. When the rung state is TRUE, the execution of the diagram directly jumps to this target rung. Note that backward jumps are dangerous as they may lead to a blocking of the PLC

Return symbol

A return symbol is placed at the end of a rung. It indicates that the execution of the program must be stopped if the rung state is TRUE.



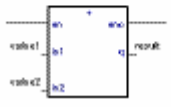
```
IF rung_state THEN
   q := (value1 > value 2);
ELSE
   q := FALSE;
END_IF;
(* continue rung with q state *)
```

The "ENO" output

On some operators, functions or function blocks, the first output does not have boolean data type. As the first output must always be connected to the rung, another output is automatically inserted at the first position, called "ENO". The ENO output always takes the same state as the first input of the block. Below is an example with AVERAGE function block, and the equivalent code expressed in ST:

```
AVERAGE(rung_state, Signal, 100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
(* continue rung with eno state *)
```

On some cases, both **EN** and **ENO** are required. Below is an example with an arithmetic operator, and the equivalent code expressed in ST:



```
IF rung_state THEN
   result := (value1 + value2);
END_IF;
eno := rung_state;
(* continue rung with eno state *)
```

Limitations of Quick LD editor

The ISaGRAF Quick LD editor does not allow to continue a rung (insert other contacts or coils) on the right of a coil. If several outputs have to be made on the same rung, the corresponding coils must be drawn in parallel.

6.2 Entering an LD diagram

All the editing commands of the Quick LD editor may be achieved either with the keyboard or with the mouse.

The editing grid

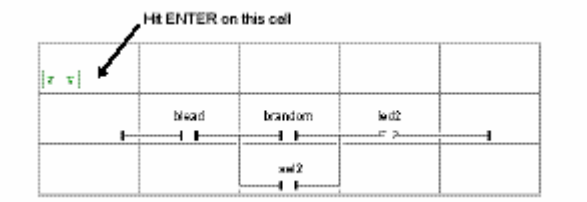
The LD diagram is entered in a logical matrix. Each cell of the matrix may contain up to one LD symbol. Use the arrows of the keyboard, or click on a cell to move the current selection. The selected cell is marked in reverse. For some cut/copy/paste operations, it is possible to select several cells. To do that with the mouse, just drag the mouse cursor in the diagram. With keyboard, use arrow keys with SHIFT key pressed.

Starting a new rung

To add a new rung to a diagram, move the selection after the last existing rung and insert a contact (hit F2 or press the corresponding button in the LD toolbar). A new rung with one contact and one coil is created.

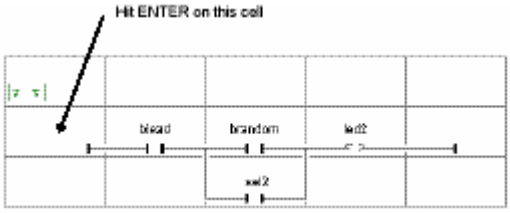
Entering the rung comment

Each rung may be documented with up to two lines of text. To enter a rung comment text, move the selection on the cell upon the rung and hit ENTER key, or double click on this cell with the mouse:



Entering the rung label

Each rung may be identified by a name. This name can be used as a target label for jump operations. To enter or change the label of a rung, move the selection on rung head and hit ENTER key, or double click on this cell with the mouse:



The ISaGRAF Quick LD editor keeps the memory of the rung labels you already entered, whether it has been specified for a rung name or a jump operation. The "Jump/Label"' dialog box gives you the possibility either to enter a new label, or to select an existing one.

If you enter a new name, it will automatically be added to the list. The "Remove" button is used to remove the selected name from the list. It does not remove the label on the rung you selected in the diagram. To do this, just press OK when the edit box is empty.

Inserting symbols on a rung

The insertion of symbols (contacts, coils, blocks...) on an existing rung is always made according to the current selection. You have to select a valid cell position within the rung and hit one of the following function keys to insert:

F2..... a contact before the selected symbol (on the left)

F3..... a contact after the selected symbol (on the right)

F4.....a contact in parallel with the selected symbol

F6.....a block before the selected symbol (on the left)

F7.....a block after the selected symbol (on the right)

F8.....a block in parallel with the selected symbol

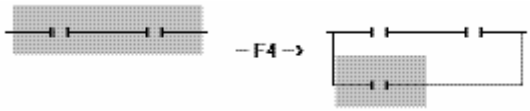
The following commands are valid when the selection is on the rung output (coil):

F5..... add a coil in parallel with the selected one

F9..... add a "Jump" symbol in parallel with the selected one

Shift+F9 ... add a "Return" symbol in parallel with the selected one

For parallel insertion (F4/F8), if several contacts of a rung are selected together, the symbol is inserted in parallel with the group of selected elements. Below is an example:



To insert symbols in the diagram, you can also use the commands of the "Insert" menu. With the mouse, you can click on the LD toolbar, on the type of symbol you want to insert:

Entering symbols

To associate a variable symbol to a contact or a coil, select it and hit ENTER. With the mouse, double click on the contact or coil. A variable selection box appears. Refer to chapter "More about program editors" in this document for further information about how to use this box. To associate a function, function block or operator to a block, hit ENTER when the selection is on the inside its rectangle. To associate a variable symbol to an input or output block parameter the selection must be on the corresponding location, outside the rectangle of the block. Dialog boxes including variable or block selection lists are normally used for text input. If the "Manual keyboard input" mode is checked in the "Options" menu, variable symbols and block names are entered directly in a single text edit box. Enter new text and hit "Enter" key to validate it, or hit "Escape" key to exit modification and close the text editing box. The text edit box used in "manual keyboard input" mode cannot be closed with the mouse.

Changing the type of contacts and coils

The "Edit / Change coil/contact type" changes the type of the selected contact or coil. A contact may be direct, negated, with positive or negative edge detection. A coil may be direct, negated, set or reset, with positive or negative edge detection. Hitting the SPACE bar has the same effect.

Inserting a rung in a diagram

The "Edit / Insert rung" command insert a new rung in the diagram, before the selected one. The rung is initiated with one contact and one coil.

6.3 Working on an existing diagram

The commands of the "Edit" menu are used to change or complete an existing diagram. Most of these commands act on the elements currently selected in the diagram.

Correcting a diagram

The DEL key can be used to remove the selected elements. It is not possible to remove a coil, a jump or return symbol when it is the only output of a rung. Use "Edit / Undo" command to restore elements after a DEL command. The DEL command can also be applied to a group of elements selected in the diagram. The DEL command can be used when selection is on the rung comment text to reset it. The DEL command, used when the selection is on the rung head, removes the entire rung.

Copying symbols

The "Cut", "Copy", "Paste" commands of the "Edit" menu are used to move or copy selected elements. These commands do not act on rung comments. The "Edit / Paste special" command gives you the choice to insert the pasted elements:

- before the selected element (on the left)
- after the selected element (on the right)
- in parallel with the selected element

Managing entire rungs

All editing commands (delete, copy, cut...) act on the entire rung if the selection is on the rung header (left power rail). This provides an easy way to arrange rungs in the diagram, just by moving the selection in the first column. It is also possible to extend the selection vertically so that it includes several rung headers. In this case edition commands may be applied to a list of entire rungs.

Find and replace

The "Edit / Find" and "Edit / Replace" menu commands are used to find and replace texts in the diagram. Only complete names can be found. Search acts on contacts, coils, block names, block parameters and run labels. It cannot be used to find a string in a rung comment. The Replace command cannot be used to change the type of a block. The research can be made upward or downward, starting at position of the current selection. It "loops" when the limits of the diagram are reached. The following shortcuts are also available for quick research of variable

ALT+F2 finds the next element with the same variable name as the element currently selected. This feature can also be applied to function blocks and rung labels.

ALT+F5 finds the next coil with the same variable name as the element currently selected.This feature is mainly used in debug mode, to quickly find out the rungs which forces a suspicious variable.

6.4 Display options

The commands of the "Options" menu are used to customise the drawing of the LD diagram on the screen, and to hide or display some types of information.

Rung comments

Use the "Options / Rung comments" command to hide or display the rung comments in the whole diagram. Hiding the rung comments can be required to have a more condensed view on a huge diagram, as each comment consumes one row in the editing matrix. This option does not affect the contents of the existing rung comments, and can be swapped at any time.

Names and aliases

Each variable, when associated to a contact, a coil or a block I/O parameter is identified by its symbolic name. The ISaGRAF Quick LD editor also introduces the notion or "**alias**" for each variable. The alias of the variable is the variable comment text, truncated before the first '**:**' character, and limited to 16 characters. Below are examples:

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

Aliases have no effect on the execution of the LD diagram and should be considered

as comments for the syntactic point of view. A variable alias is automatically extracted from the variable comment when the name is selected in the variable list. It cannot be changed manually. Use the "**Options / Contacts and coils**" commands to select a display mode for variable identification. The following modes are available:

- display only the variable names
- display only the variable aliases
- display both names and aliases

Quick LD editor does not automatically updates LD documents when variable aliases are changed in the dictionary. Use the "**Options / Contacts and coils / Update aliases**" command to update all aliases in edited diagram. You can also set the "**Always update on Open**" option from "**Options / Contacts and coils**" to ask ISaGRAF to automatically update all used aliases each time a Quick LD program is open. Warning: Setting this option may significantly increase the time spent to open a program.

Drawing options

The "Options / Layout" command opens a dialog box where are grouped all the parameters and options concerning the editor workspace and the drawing of the graphic LD diagram. Use the check boxes in the "Workspace" group to display or hide editor tool bar, status bar and LD toolbar. Options of the "Document" group allow you to show or hide points of the editing grid, and to enable/disable the use of colours for the drawing.

Options of the "Zoom" group allow you to select a main zoom ratio. You can also use the "zoom" button in the editor toolbar to swap between default zoom ratios.

You can also customise the X/Y aspect ratio of cells in the editing grid. This last option can be used to reduce the default cell width, if you commonly use short names for variables. You can also use the "width" button in the editor toolbar to change the X/Y aspect ratio without entering the

Layout dialog box. Use the "Options / Font" command to select the name of the character font to be used in all ISaGRAF graphic documents. When selecting font, font style and size are not relevant and do not need to be specified. ISaGRAF graphic editors always calculate the font size according to selected zoom ratio.

7. Using the FBD/LD editor

The ISaGRAF FBD/LD graphic editor allows the user to enter complete FBD programs, which may includes parts in LD. It combines graphic and text-editing capabilities, so both diagrams and corresponding inputs and outputs can be entered. As this editor is more dedicated to FBD language, pure LD diagrams should rather be entered using the ISaGRAF Quick LQ editor.

7.1 Basics of the FBD/LD languages

The **FBD** language is a graphic representation of many different types of equations. **Operators** are represented by rectangular function boxes. Function inputs are connected to the left side of the box. Function outputs are connected to the right side. Diagram inputs and outputs (**variables**) are connected to the function boxes with **logical links**. An output of a function box may be connected to the input of another box.

The **LD** language enables graphic representation of boolean expressions. Boolean **AND**, **OR**, **NOT** operators are explicitly represented by the diagram topology. Boolean input variables are attached to graphic **contacts**. Boolean output variables are attached to graphic **coils**. Contacts and coils are connected together and to left and right power rails by **horizontal lines**. Each line segment has a boolean state of **FALSE** or **TRUE**. The boolean state is the same for all the segments directly linked together. Any horizontal line connected to the left **vertical power rail** has the **TRUE** state.

LD and FBD diagrams are always interpreted from the left to the right, and from the top to the bottom. Refer to the ISaGRAF Language reference Manual for more details about LD and FBD languages. These are the basic graphic components of the LD and FBD languages, such as supported by the FBD/LD editor:

Left power rail

Rungs must be connected on the left to a left power rail, which represents the initial "TRUE" state. ISaGRAF FBD editor also allows connecting any boolean symbol to a left power rail.

Right power rail

Coils may be connected on the right to a right power rail. This is an optional feature when using the ISaGRAF FBD/LD editor. If a coil is not connected on the right, it includes a right power rail in its own drawing.

LD vertical "OR" connection

LD vertical connection accepts several connections on the left and several connections on the right. Each connection on the right is equal to the OR combination of the connections on the left.

Contacts

A contact modifies the boolean data flow, according to the state of a boolean variable. The name of the variable is displayed upon the contact symbol. The following types of contacts are supported by ISaGRAF FBD/LD editor:

- ..... direct contact
- ..... negated contact
- ..... contact with positive (rising) edge detection
- ..... contact with negative (falling) edge detection

Coils

A coil represents an action. It must be connected on the left to a boolean symbol such as a contact. The name of the variable is displayed upon the coil symbol. The following types of coils are supported by ISaGRAF FBD/LD editor:

- ..... direct coil
- ..... negated coil
- ..... "set" action coil
- ..... "reset" action coil

Function blocks

A block in an FBD diagram can represent a function, a function block, a subprogram or an operator. Inputs and outputs must be connected to variables, contacts or coils, or other block inputs or outputs. Formal parameter names are displayed inside of the block rectangle.

Labels

Labels can be placed everywhere in the diagram. Labels are used as targets for jump instructions, to change the execution order in the diagram. Labels are not connected to other elements. It is highly recommended to place labels on the left of the diagram, in order to increase the diagram readability.

Jumps

A jump symbol always refers to a label placed elsewhere in the diagram. Its left connection must be linked to a boolean point. When the left connection is TRUE, the execution of the diagram directly jumps to this target label. Note that backward jumps are dangerous as they may lead to a blocking of the PLC loop in some cases.

Return symbol

A return symbol is connected to a boolean point. It indicates that the execution of the program must be stopped if the rung state is TRUE.

Variables

Variables in the diagram are represented inside small rectangles, connected on the left or on the right to other elements of the diagram.

Connection links

Connection links are drawn between elements put in the diagram. Links are always drawn from an output to an input point (in the direction of the data flow).

Connection links with boolean negation

Some boolean links are represented with a small circle on their right extremity. This represent a boolean negation of the information transported by the link.

User defined corners

User defined points may be defined on links. They allow the user to manually control the routing of a link. If no corner is placed, the ISaGRAF FBD/LD editor uses a default routing algorithm.

7.2 Entering an FBD diagram

To enter a diagram, you have to place elements (blocks, variables, contacts, coils...) in the graphic area, and draw links between them.

Inserting objects

To insert an object in the diagram, select the corresponding button in the toolbar and click in the graphic area, where you want to insert it.

Selecting objects

Selecting graphic objects is needed for most of the editing commands. The ISaGRAF LD/FBD graphic editor enables the selection of one or more existing objects in the diagram area. To select objects, the "select" (button with an arrow) choice must be checked in the editor toolbar. To select one object, the user only has to click on its symbol. To select a list of objects, drag the mouse in the diagram and select a rectangle area. All the graphic objects that intersect the selection rectangle are marked as "selected". A selected object is drawn with little black squares around its graphic symbol. By making a new selection, all previously selected objects are unselected. To remove the existing selection, simply click with the mouse on an empty area, outside of the rectangle which borders the selected objects.

Inserting comments

Comments may be inserted anywhere in the diagram. Comments have no influence on the program execution. They allow a higher readability of the diagram. To insert a comment block, select this button in the toolbar, and drag the mouse to select the rectangle area where comment must be drawn. Then enter the text of the comment. No special leading or trailing characters such as "(*" and "*)" are needed when entering the text of a comment block. A comment block may be resized by dragging the corners of its border when it is selected.

Moving objects

To move objects in the diagram, you have to select them, and drag the mouse to move the selected area in the diagram. To move connected objects, the user simply has to move the graphic symbols put on the diagram. The ISaGRAF LD/FBD editor will automatically redraw the connection lines between the objects that were moved, based on their new location.

Drawing links

Select one of these buttons in the toolbar to draw a link between connection points of existing elements. If you draw a link from a connection point to an empty location in the diagram, a user-defined corner automatically terminates it, so that you can continue drawing another segment.

Changing link drawing

The "Tools / Move line" command is used when a link is selected in the diagram to change its automatic routing. This command has no effect when the link is connected to a user-defined corner. When a link is drawn as three segments, this command changes the position of the second segment. Below are examples:

Changing the type of a link

You can easily change the type of link (with or without Boolean negation) by double clicking with the mouse on its right extremity.

Drawing LD rungs

To draw a new LD rung, first insert the left power rail. Then place a coil: it will be automatically linked to the power rail. Other contacts and vertical OR connections may be directly inserted on the rung line, without drawing any new connection link. When a new LD contact or coil is inserted in an empty space of the editing area, the new horizontal rung line is automatically drawn from the new inserted element to the existing power rails on the left and on the right. This line is not automatically drawn if the new contact or coil is not placed between power rails. The new inserted contact or coil can then be freely moved on the drawn rung. The horizontal lines created by the editor while inserting an LD contact or coil symbol can be selected and deleted. You can insert a new LD contact or coil symbol on the horizontal line of an existing rung. The editor automatically cuts the rungs and connects it to the left and right connection points of the new inserted contact or coil.

Multiple connections

A multiple connection can be created on the right of any output point. It means that the information is broadcasted to several other points in the diagram. The same state is propagated on each extremity on the right. The number of lines drawn at the right of an output connection point is not limited. Two connection lines cannot have their right extremity connected on the same input point, except for the following LDsymbols:

..... right power rail

 ..... multiple connection on the left (OR) operator

These LD symbols can have an unlimited number of inputs.

7.3 Working on an existing diagram

The commands of the "Edit" menu are used to change or complete an existing diagram. Most of these commands act on the elements currently selected in the diagram.

Correcting a diagram

The DEL key can be used to remove the selected elements. Pending links are deleted with selected elements. Use "**Edit / Undo**" command to restore elements after a DEL command. The DEL command can also be applied to a group of elements selected in the diagram. The "**Cut**", "**Copy**", "**Paste**" commands of the "**Edit**" menu are used to move or copy selected elements.

Find and replace

The "Edit / Find" and "Edit / Replace" menu commands are used to find and replace texts in the diagram. Only complete names can be found. Research acts on contacts, coils, block names, variables and labels. It cannot be used to find a string in a comment text. The Replace command cannot be used to change the name of a block. The research can be made upward or downward, starting at the current selection position. It "loops" when the limits of the diagram are reached.

Displaying the execution order

When an FBD diagram includes backward loops, the execution order cannot follow the single left to right / top to bottom method. In order to avoid confusion, use the "Tools / Show execution order" command or press Control + F1 keys to display the execution order that will be used at compiling time. Tags numbered from 1 to N are displayed close to symbols that lead to an action (coils, set variables and function blocks).

Entering symbols and texts

Double click with the mouse on an element to enter the associated symbol or text. This applies to variables, contacts and coils, comment texts and labels. When used on a contact or coil, this also allows to change its type (direct, negated...).

Dialog boxes including variable or block selection lists are normally used for text input. If the "Manual keyboard input" mode is checked in the "Options" menu, variable symbols and block names are entered directly in a single text edit box. Enter new text and hit "Enter" key to validate it, or hit "Escape" key to exit modification and close the text editing box. The text edit box used in "manual keyboard input" mode cannot be closed with the mouse.

If the "Auto input" mode is checked in the "Options" menu, the variable symbol must be entered immediately each time a new contact or coil is inserted. The symbol must always be entered immediately when a variable or label is inserted.

Selecting function block type

Double click with the mouse on a block is used to change its type. The block type is selected from the list of available operators, functions and function blocks. This command also allows changing the number of input points in the case of a commutative operator. (e.g. AND, OR, ADD, MUL...)

Getting free space

When you press the right button of the mouse in the FBD drawing area, a popup menu is displayed. It contains the following commands that can be used to insert or remove free space at the location of the mouse cursor:

Insert rows:.....	This command inserts horizontal free space, made of 4 rows according to the grid step, starting at the position of the mouse cursor where popup menu is open.
Delete rows:	This command removes unused horizontal space (rows) starting at the position of the mouse cursor where popup menu is open. This command cannot be used to remove FBD elements.

When popup menu is open, a grey line in the FBD drawing area indicates where empty space will be inserted or removed.

7.4 Display options

The commands of the "Options" menu are used to customise the drawing of the FBD diagram on the screen.

Layout customisation

The "Options / Layout" command opens a dialog box where are grouped all the parameters and options concerning the editor workspace and the drawing of the graphic diagram. Use the check boxes in the "Workspace" group to display or hide editor toolbars and status bar. Option of the "Document" group allows you to show or hide points of the editing grid.

Options of the "Zoom" group allow you to select a main zoom ratio. You can also use the "zoom" button in the editor toolbar to swap between default zoom ratios.

Use the "Options / Font" command to select the name of the character font to be used in all ISaGRAF graphic documents. When selecting font, font style and size are not relevant and do not need to be specified. ISaGRAF graphic editors always calculate the font size according to selected zoom ratio.

7.5 Styles and modification tracking

The ISaGRAF LD/FBD editor enables you to assign a graphic style to any component of an LD/FBD diagram. A style is mainly defined as a special diagram colouring. But ISaGRAF also uses styles to enable modification tracking in diagrams for version control purpose.

Note that styles are not visible during simulation or on-line debug, as colours (red and blue) are used in that mode to highlight TRUE / FALSE states of spied variables.

Predefined styles

The following styles are pre-defined:

Normal.....	Default drawing (black). For modification tracking, "normal" style indicates that elements having that style are part of the original diagram. "Normal" style elements are normally scanned during execution.
Modified	Elements marked as "modified" are painted in pink. For modification tracking, the "modified" style is used to highlight elements that have been added or changed after the original release of the diagram. "Modified" style elements are normally scanned during execution.
Deleted	Elements marked as "deleted" are painted in grey, with dashed lines. Such elements are not taken into account for the execution of the diagram. This style is used to keep a track of elements removed after the original release when version control is required.
Custom.....	In addition to predefined style, ISaGRAF LD/FBD editor allows you to select any colour to be applied to a part of the diagram. Such elements are considered as having a "Custom" style. The use of "Custom" style has no effect on the diagram execution at run time.

Use the commands of "Style" sub-menu in "Edit" menu to manually apply a style to selected elements.

Modification tracking

The use of styles and the availability of the "Deleted" style allow automatic modification tracking in an existing diagram. Use the "Mark modifications" Command in "Edit/Style" menu to enable or disable modification tracking. When the "Mark modifications" option is set, all elements changed in or added to the diagram are automatically set with "Modified" style. When an element is deleted, using "Delete" or "Cut" commands, they are not visually removed from the diagram, but simply marked with "Deleted" style". This enables the user to automatically keep a trace of all modifications entered in the diagram. Use the "Edit/Style/Remove all deleted items" to actually remove all elements marked with "Deleted" style from the LD/FBD diagram. This command does not take care of the current selection, and always applies to the entire diagram.

To "restore" one element marked with the "Deleted" style, select the desired element and apply to it the "Normal" style, the "Modified" style or any "Custom" style. Such operation may lead to invalid connections (more than one link connected to the same input point) that will be detected during next program verification.

8. Using the text editor

This chapter only describes features and commands of the ISaGRAF text editor, particularly when used to enter the source code of ST and IL programs.

8.1 Editing commands

The commands of the "Edit" menu are used to work on the edited text. Most of these commands act on the characters currently selected in the diagram, or perform an action at the current location of the caret.

Cut and paste

The DEL key can be used to remove the selected text. Use "Edit / Undo" command to restore elements after a DEL command. The "Cut", "Copy", "Paste" commands of the "Edit" menu are used to move or copy text in the program, or to insert pieces of texts copied in the clipboard by other applications.

Find and replace

The "Edit / Find" and "Edit / Replace" menu commands are used to find and replace texts in the program. Any character string can be found. Research can be performed upward or backward, starting at the current location of the caret. It does not "loops" when the limits of the program are reached.

Go to line

The "Edit / Go to line" command is used to move the caret to a specific line number. This can be very useful to have access to a line with an error detected by the ISaGRAF compiler in an ST or IL program, and referenced by a line number.

Insert symbol from dictionary

Use the "Edit / Insert variable" command to insert at the caret position the symbol of a variable or object declared in the project dictionary. Symbol is selected through the common variable selection box described in chapter "More about program editors" in this document.

Insert file

The "Edit / Insert file" command inserts the whole contents of a file at the current location of the caret. Note that only pure ASCII text files can be handled by this command.

8.2 Options

The commands of the "Options" menu are used to display or hide editor toolbars, and select the character font. The selected character font will be used for any text editing in all ISaGRAF Workbench.

When used to enter the source code of an ST / IL program, the "Options / Show keywords" command is used to show or hide a toolbox that groups the most common keywords of ST or IL language. Click on a button in the toolbar to insert the corresponding keyword or operator at the current location of the caret.

9. More about program editors

This chapter contains useful information about editing features which are common to all the ISaGRAF program editors. This mainly concerns links with other ISaGRAF tools and common ISaGRAF dialog boxes.

9.1 Calling other ISaGRAF tools

Verify (compile) the program

The "File / Verify" command runs the ISaGRAF code generator to verify the programming syntax of the currently edited program. In case of SFC language, both level 1 and 2 are checked. When syntax verification is complete, the code generator window must be closed to continue work on the program. If there is only one program in the application (the edited one) the application code is generated if no syntax error is detected. The "Options / Compiling options" command is used to set compiling and optimising parameters. Refer to chapter "Using the code generator" in this document for further information about compiling and code generation.

Simulate or debug the application

The "File / Simulate" and "File / Debug" commands run the ISaGRAF graphic debugger either in simulation or real connected mode, and re-opens the edited SFC program in debug mode. Used in debug mode, no modification can be entered in the program.

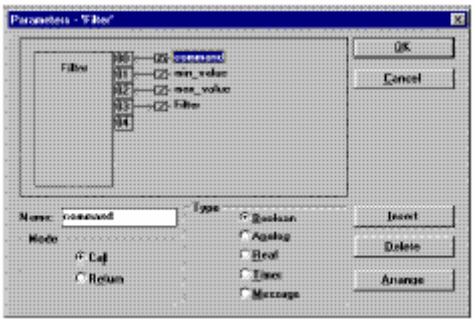
Editing the dictionary of variables

The "File / Dictionary" command is used to edit the dictionary of variables for the current application and the current program. It also contains the entry points to edit the user-defined words. The local declarations or defined words relate to the currently edited program.

9.2 Parameters of the program

When the edited program is a function, a function block or a sub-program, the "File / Parameters" command is used to define its calling and return parameters. This command has no effect if the edited program is an SFC or top level program from section Begin or End.

Sub-programs, functions or function blocks may have up to 32 parameters (input or output). A function or sub-program always has one (and only one) return parameter, which must have the same name as the function, in order to conform to ST language writing conventions. The following dialog box is used to describe the parameters of the sub program:



The list in the upper left side of the window shows the parameters, in the order of the calling model: first the calling parameters, last the return parameters. The lower part of the window shows the detailed description of the parameter currently selected in the list. Any of the ISaGRAF data types may be used for a parameter. The return parameters must be located after calling parameters in the list. Naming parameters must conform to the following rules: • the length of the name cannot exceed 16 characters

• the first character must be a letter

• the following characters must be letters, digits or underscore character

• naming is case insensitive

The "**Insert**" command is used to insert a new parameter before the selected parameter. The "**Delete**" command is used to erase the selected parameter. The "**Arrange**" command automatically rearranges (sorts) the parameters, so that the return parameters are put at the end of the list.

9.3 Other commands of the "File" menu

The following commands are available in the "File" menu of all program editors:

Open another program

The "File / Open" command allows the user to close the currently edited program and start editing another program of the current project with the same language. This function cannot be used to edit a program written in another language. The new selected program replaces the current one in the editing window.

Printing the program

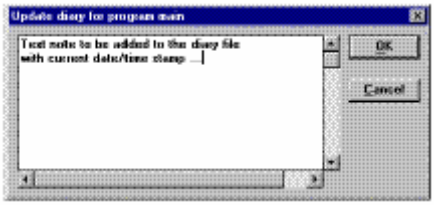
The "File / Print" command outputs the edited program on printer. This command automatically runs the ISaGRAF document generator to printout the edited program and attached local variables.

For some graphic programs (SFC, FBD and Quick LD) You can also use the "Edit / Copy drawing" command to copy in the clipboard the drawing of the chart in metafile format, so that it can be pasted in other applications such as word processors. For SFC programs, only the level 1 information (chart, numbering and level 1 comments) appears on the copied metafile.

9.4 Updating the program diary

The diary file attached to the edited program may be manually entered using the "File / Diary" command. The diary file is automatically updated with syntax checking output messages each time the program is compiled. Compiling outputs are completed with the compiling date / time stamp.

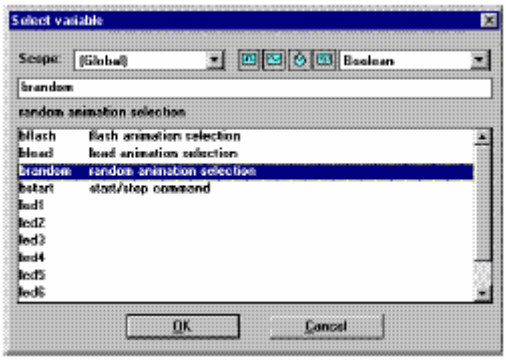
If the "**Update diary**" mode is selected in the "**Options**" menu of program editors, the following dialog box is opened each time the program is saved on disk.



If OK button is pressed, the entered text note is then stored at the end of the diary file with current date / time stamp. This feature is very useful for maintenance of complete programs, as it provides useful help about the program life cycle.

9.5 Selecting a variable from dictionary

When editing a text program (ST or IL) the "Edit / Insert variable" allows the selection of a declared variable name to be inserted at the current position of the caret. When editing LD or FBD programs, variable selection is required for the description of contacts, coils, block I/O parameters or FBD variable boxes. In both cases, the following dialog box is open to select a declared variable:



The "Scope" selection box is used to select between global and local variables. The selection box on the right allows the selection of the data type. Small icons beside the type selection box are buttons that can be used as shortcuts to select most current data types:

..... Boolean

..... Integer / Real

..... Timer

..... Message

To select a variable, click on its name in the list. Its name and comment are then displayed on the top of the list. Then press the "OK" button to confirm its selection. It is also possible to directly enter a variable name in the edit control without using the list.

9.6 The output window

The following commands are available in the Tools menu of all language editors. They are used to display information in a small text list at the bottom of the editing window, and use it for program browsing.

"Show compiler output"	display in the output window the error messages from the last compiling of the edited program.
"Find in…"	find occurrences of a text in the whole edited program, and list them in the output window. For SFC and FC languages, this command searches in all level 2 programs.
"Hide output window"	close the output list window

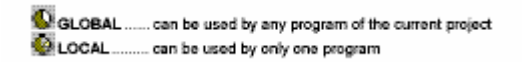
When error messages or occurrences are displayed in the output window, double click on a line to directly move selection to the corresponding location. For SFC and FC languages, this command opens corresponding level 2 programming window.

10. Using the dictionary editor

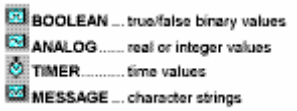
The ISaGRAF dictionary is an editing tool for the declaration of the internal variables, I/O variables, function block instances, and "defined words" of the application. The dictionary groups together the declared variables and function block instances of the application, and the words defined as constant strings. Variables, function blocks and defined words must be declared in the dictionary before using them in source code. Variables and defined words can be used with any of the automation languages: SFC, FBD, LD, ST and IL. Function blocks used in FBD language do not have to be declared, because the ISaGRAF FBD and Quick LD editors automatically declare the instances of the used blocks.

Variables

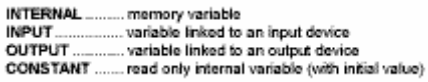
The variables are sorted according to their range and type. Only variables of the same type and the same range can be entered on the same input grid. These are basic ranges for variables:



These are basic types of variables:



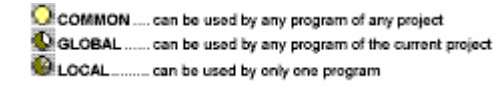
A variable is identified by a name, a comment, an attribute, a network address and other specific fields. Here are the basic variable attributes:



Note: Timers are always internal variables. Input and Output variables always have the GLOBAL range.

Defined words

A defined word is an alias that can be used in any language to replace a text string. The replaced text can be a variable name, a constant expression or a complex expression. Defined words are sorted according to their range. Only defined words of the same type and the same range can be entered on the same input grid. Here are basic ranges:



A defined word is identified by a name, a well-defined block of ST text equivalence and a free comment.

Function block instances

The instances of the function blocks used in the ST and IL languages must be declared in the dictionary. Because a function block has internal "hidden" data, each copy of a function block must be identified. The following example shows the function block "R_TRIG" (rising edge detection) defined in the library, used for edge detection on different variables. Each copy of the block must be identified by a unique name. Naming the type of block and definition of its parameters is made by using the library manager:

```
Block name:     R_TRIG
Parameters:     Input=CLK
                Output=Q
```

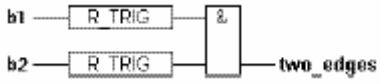
Naming the instances is made by using the dictionary editor:

```
instance name:  TRIG_B1     Block name:  R_TRIG
instance name:  TRIG_B2     Block name:  R_TRIG
```

The declared instances may be used in ST programs:

```
TRIG_B1 (b1);
edge_b1 := TRIG_B1.Q;    (* b1 variable edge detection *)
TRIG_B2 (b2);
edge_b2 := TRIG_B2.Q;    (* b2 variable edge detection *)
```

Declared function block instances may be GLOBAL (known by any program in the project), or LOCAL to one program. Function blocks used in FBD or LD languages do not have to be declared, because the ISaGRAF FBD editor automatically declares the instances of the used blocks.



(* the function blocks always have the name of the block defined in the library. The ISaGRAF FBD and Quick LD editors automatically declare an instance each time a block is inserted in the diagram *)

Function block instances automatically declared by the FBD and Quick LD editors are always LOCAL to the edited program.

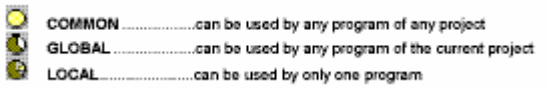
Network addresses

Network addresses are optional. A variable with a non-zero network address can be spied by an external system (for example a process visualisation system) at run time. More generally, the network address provides an identifying mechanism for each run time communication system that cannot handle symbolic names. A network address may be entered for each variable, during its complete description, when the variable is created or modified.

10.1 The dictionary main window

The dictionary editing window shows a list of variables with same type and range.The type and range of edited variables is always displayed in the title bar.

The editing window shows only main fields of variable description: name, attribute and network address, and text comment. The full description of the selected variable is always displayed in the status bar. Use the following buttons in the toolbar to select the range of variable to be edited:



Use the "Tab" control displayed with title bar to select the type of object to be edited:

Use the text-input field on the left of the toolbar to search for a variable prefix name. In this case, research is processed on the entire list, from the beginning, based on the current selection. The "Edit / Find" command is also available to search a text string in variable names and comments, and to move the selection to this variable. Search is always case insensitive.

10.2 Managing variables

The available "Files" menu commands work on the entire selected class of variables, function block instances or defined words. Use the "Other" command to select the type and range of objects to be edited.

Printing variables

Use the "Files / Print" command to print the currently edited list of variables or defined words, on a standard Windows™ printer device. Printing is made using the ISaGRAF document generator. The printout includes the complete description of each variable or defined word of the currently edited type.

Creating new variables

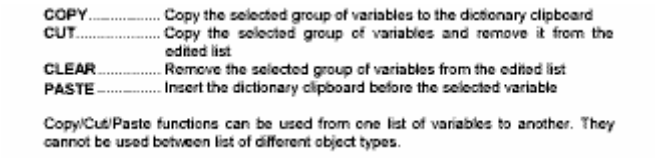
The "Edit / New" command allows the user to create new variables, function block instances or defined words for the selected range and type. New variables are inserted just before the variable currently pointed to by the selection bar. When this command is run, an input box is opened to enter the variable description. When the description is complete, pressing the "Store" button puts it onto the list. The input box is automatically re-opened, so the user can enter other variables with the same "Edit" command. Pressing the "Cancel" button of the dialog box breaks the variable creation process.

Modifying existing variables

The "Edit" command of the "Edit" menu allows the user to modify the description of the variable currently pointed at by the selection bar. When this command is run, an input box is opened to modify the variable description. When description is complete, pressing the "Store" button enables modification. The user also can press "Next" and "Previous" buttons to extend the modification command to adjoining variables. Pressing the "Cancel" button closes the dialog box without storing any modification.

Cut and paste

The ISaGRAF dictionary editing tool enables multiple-line selection. Many commands are available to work on the currently edited list of variables. Below are available "Edit" menu commands:



Sorting variables

The "Tools / Sort" command sorts the variables or defined words of the currently edited list. The sorting order is given by the attributes of the variables:

- first the internal variables
- then the input variables
- finally the output variables

Variables with the same attribute are sorted into alphabetical order. Defined words are always sorted into alphabetical order.

Setting network addresses

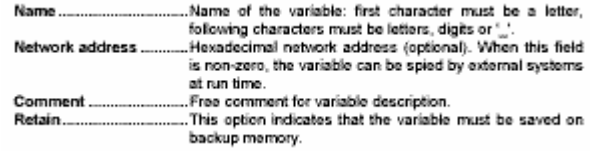
Network addresses are optional. A variable with a non-zero network address can be spied by an external system (for example a process visualisation system) at run time. A network address may be entered for each variable, during its complete description, when the variable is created or modified. The "Tools / Renumber addresses" command allows the user to set up network addresses of an entire group of variables. When this command is run, it acts on the group of variables currently selected on the list. Entering a hexadecimal basis address (address for the first variable of the group) results in network addresses of the variables of the group being set with consecutive addresses. Entering a null basis address resets to zero the network address of all the selected variables.

Importing boolean "true/false" strings

When editing defined words, the "Tools / Import true/false definitions" allows the user to automatically define as language keywords the strings attached to boolean variables to represent TRUE and FALSE states. Such strings are normally defined for debug formatting. They have to be specified as defined words if they are to be used in programs. This command searches for boolean true/false strings in the declarations with the same range as the one currently selected for the editing of the defined words.

10.3 Description of objects

A complete description must be entered for each variable, function block instance, or defined word. Description fields are different for each type of object. The following fields are common for any type of variables:



These are other description fields for a boolean variable:

Attribute ..... Specifies an internal, constant, input or output variable.
"False" string ..... String used for false value at debug time.
"True" string ..... String used for true value at debug time.
Set to true at init ..... The initial value is TRUE if this option is checked, otherwise the initial value is FALSE.

These are other description fields for an integer or real variable:

Attribute ..... Specifies an internal, constant, input or output variable.
Format ..... Specifies an integer or real (floating) variable. Display format used during debug can be selected.
Unit string ..... String used to identify the physical unit at debug time.
Conversion ..... Name of the conversion table or conversion function attached to the variable (for input or output variables only)
Initial value ..... Initial value of the variable (must have the same format as the variable). If not specified, the initial value is 0.

These are other description fields for a timer variable:

Attribute ..... Specifies an internal or constant variable.
Initial value ..... Initial value of the variable (time value). If not specified, the initial value is time#0s.

These are other description fields for a message variable:

Attribute ..... Specifies an internal, constant, input or output variable.
Maximum Length ..... Specifies the maximum number of characters that can be stored in the message.
Initial value ..... Initial value of the variable (length cannot exceed the capacity of the message). If not specified, the initial value is the empty string.

These are the description fields for a defined word:

Name ..... Name used in ST source files: first character must be a letter, following characters must be letters, digits or '_'.
Define ..... String according to ST syntax that replaces the defined word during compiling. Example: Name = PI - Equivalence = 3.14159
Comment ..... Free comment for defined equivalence

These are the description fields for a function block instance:

Name ..... Name of the instance, used in ST source files: first character must be a letter, following characters must be letters, digits or '_'.
Type ..... Name of the corresponding function block in the library.
Comment ..... Free comment for the function block instance.

10.4 Quick declaration

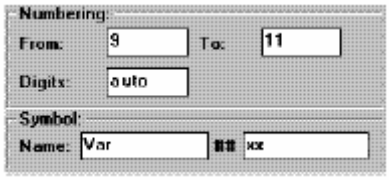
The "Tools / Quick declaration" command enables you to declare several variables at the same time. Variables created by quick declaration are named using a numbering convention. For that, you have to define:

- the index (number) of the first and the last variables,
- the text to be added before and after the number in variable symbols
- the number of digits used to express the number in variable symbols.

Additionally, you can specify basic attributes of created variables (internal, input or output...), plus some properties depending on the variable type ("Retain" attribute, integer or real format, message string maximum length).

You always need to define a text to be inserted before variable number, as a variable symbol cannot start with a digit. When the "number of digits" is set to "Auto", ISaGRAF formats the variable number on the minimum needed number of digits. When number of digits is specified, ISaGRAF formats all numbers to the specified length by adding leading '0' characters. Setting a fixed number of digits for variable numbers can be very useful to prevent bad lexicographic sorting. Below are some examples.

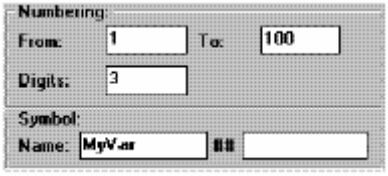
Example This setting for quick declaration:



will create the three following variables:

Var9xx Var10xx Var11xx

Example This setting for quick declaration:



will create 100 variables with names from MyVar001 to MyVar100

10.5 Modbus SCADA addressing map

ISaGRAF "network addresses" are often used to establish a link between ISaGRAF system and a SCADA based on Modbus communication. In that case, the SCADA is a Modbus master and ISaGRAF target acts as a Modbus slave. Network addresses are used to create a virtual Modbus map for all ISaGRAF variables that must be controlled from the SCADA. The "Tools / Modbus SCADA addressing map" is a powerful to quickly create a Modbus virtual map with variables of the application.

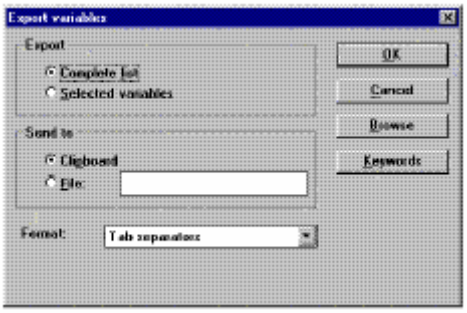
The mapping tools shows two lists. The upper one is a segment (4096 locations) of the Modbus map, showing mapped variables (the ones having a network address). The lower list shows unmapped variables (without network address defined). The "0" address cannot be used to map a variable. Use the "Map" and "Remove" commands of the "Edit" menu to move a variable from one list to another, and thus build the map. Same actions can be performed by double clicking on a variable symbol in a list, to send it to the other list. At any moment, you can use the "Segment" drop down list to view another segment of the map. The commands of the "Options" menu can be used at any moment to display addresses either in decimal or in hexadecimal. The "Edit / Find" commands is used to search for a declared variable, whether it is already mapped or not.

10.6 Exchanging information with other applications

The ISaGRAF dictionary editing tool offers import/export functions in order to exchange information with other applications, such as word processors, spreadsheets, data base managers... These commands are grouped in the "Edit" menu. The "Export text" command builds a pure ASCII text description of the fields describing a set of edited objects, and stores this text either in the Windows clipboard or in a file. Such information is typically used by another application. The "Import text" command imports variable declaration description fields, described in pure ASCII text format, stored either in the Windows clipboard or in a file, and updates the currently edited list with imported fields. Such information is typically produced by another application.

- ***Exporting data***

The following dialog box appears when the "Export text" command is run. It enables the user to control the export mechanism.

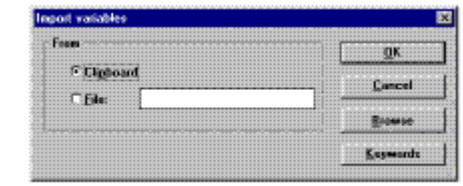


Checking the "Complete list" choice indicates that the complete edited list has to be exported. The current selection is ignored in this case. Checking the "Selected variables" choice indicates that only highlighted variables will be exported. If the "Clipboard" option is checked, the exported information is stored, in pure ASCII text format, in the Windows clipboard. The text is then available for "paste" commands in other applications. If the "File" option is checked, the exported text is stored in an ASCII file. The complete pathname of this file has to be entered. The "Browse" command may be used to find an existing path name.

Then the user chooses a format for the exported text. The available formats are described in further sections. Pressing the "OK" button runs the export function. Pressing the "Cancel" button closes the dialog box and escapes from the export command. All the fields of the selected objects are stored in the exported text, in the standard declaration order. The first line of the exported text contains the name of the fields. Each object is described on one line of text. The "end of line" separator is the Standard MS-DOS sequence "0d-0a". The names used to identify the fields in the first exported line may be changed, by pressing the "Keyword" button. This command is described in further sections.

- ***Importing data***

The following dialog box appears when the "Import text" command is run. It enables the user to control the import mechanism.



If the "Clipboard" option is checked, the imported information is taken from the Windows clipboard, in pure ASCII text format. If the "File" option is checked, the exported text is read in an ASCII file. The complete pathname of this file has to be entered. The "Browse" command may be used to find an existing pathname. The import function automatically recognises the format (separators) used in the imported text. The available formats are described in further sections. Pressing the "OK" button runs the import function. Pressing the "Cancel" button closes the dialog box and escapes from the import command. The names used to identify the fields in the first imported line may be changed, by pressing the "Keyword" button. This command is described in further sections. The first line of the text must contain the name of the fields, according to the order used in the following lines. Each object must be described on one line of text. The "end of line" separator is the standard MS-DOS sequence "0d-0a". Fields can appear in any order. If some fields are missing, they are automatically filled in the imported object description by default values. If an imported object already exists in the edited list, the user has to confirm that it will be overwritten. The object description is then updated with imported fields. If some fields are missing, they are not updated in the object description.

Available text formats

Below is the list of available formats for export command. The import command automatically recognises these formats.

- tab separators

Description: Fields are separated by tab characters.

Example:

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

- comma separators

Description: Fields are separated by commas.

Example:

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

- semicolon separators

Description: Fields are separated by semicolons.

Example:

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

- commas and quotes

Description: Fields are separated by commas.
Each field is written between quotes.

Example:

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

Keywords

The names used to identify the fields in the first imported or exported line may be changed, by pressing the "Keyword" button. This command opens the following dialog box:

The window shows the list of object fields, and the associated keywords. To modify a keyword, the user must select a field in the list and press the "Modify" button.

Pressing the "Default" button restores the original list of keywords. Naming the keywords must conform to the following rules:

- the name cannot exceed 16 characters
- the first character must be a letter
- the following characters can be letters, digits or '_' character
- the same name cannot be used for different keywords

Below are the standard keywords found in ISaGRAF:

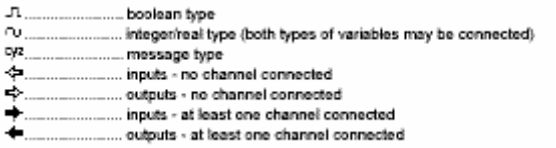
Object name	Name
Text comment	Comment
Network address	Address
Attributes (internal, input, output)	Attribute
Boolean 'False' string	False
Boolean 'True' string	True
Analog format (real or integer)	Format
Analog unit string	Unit
Analog conversion name	Conversion
Message maximum length	MaxLength
Function block library type	Library
Defined word equivalence	Equivalence
Internal attribute	Internal
Input attribute	Input
Output attribute	Output
Constant attribute	Constant
Real analog format	Real
Integer analog format	Integer

11. Using I/O connection editor

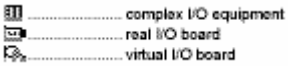
The aim of the I/O connection operation is to establish a logical link between the I/O variables of the application and the physical channels of the boards existing on the target machine. To make this link the user has to identify and set-up all the boards of the target machine, and place I/O variables on corresponding I/O channels. The list on the left shows the rack of the target machine, with board slots. A slot may be free, or used by one I/O board or complex equipment. Each slot is identified by an order number. The rack may contain up to 255 boards. The list on the right shows the board's parameters and the variables connected on the selected board. A board may have up to 128 I/O channels. The total number of single I/O boards (including single equipments and boards of complex equipments) cannot exceed 255.

Icons

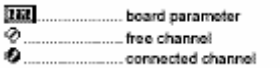
The icons displayed on the front face indicate the type and attributes of variables that may be connected to the board channels. The ISaGRAF system does not allow the connection of variables of different types on the same board. This is the meaning of the used icons:



Below are the icons used to show the type of I/O device installed on a slot:



Below are the icons used to draw a parameter or a channel:



Moving boards in list

Use these buttons in the toolbar or "Edit / Move board up/down" menu commands to move the selected I/O board one line up or down in the main list. The "Edit / Insert slot" command inserts an empty slot at the current position.

11.1 Defining I/O boards

The "Edit" menu contains basic commands to define the selected board (set-up its parameters), and to connect I/O variables to its channels.

Selecting I/O board type

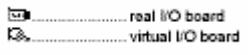
Before connecting I/O variables to a board, the board identification must be entered. A library of pre-defined boards is available on the ISaGRAF workbench. This library may have been compiled by one or more I/O device suppliers. The "Edit / Set Board/Equipment" command is used to set-up board identification. This command can be used to select either a single board, or complex I/O equipment from the ISaGRAF library. It is also possible to double click on a slot to set the corresponding board or equipment. All the channels of a single board have the same type (boolean, integer/real or message) and direction (input or output). Real and integer variables are not distinguished during I/O connection. A complex I/O equipment represents an I/O device with channels of different types or directions. A complex I/O equipment is represented as a list of single I/O boards. It uses only one slot in the rack list.

Removing a board

The "Edit / Clear slot" command is used to remove the currently selected board or I/O equipment. If variables are already connected to the corresponding channels, they are automatically disconnected when clearing the slot.

Real boards and virtual boards

The "Edit / Real/virtual board" command sets the validity of the selected board or complex I/O equipment. The following icons are displayed in the rack list to show the validity of a board:



In Real Mode, I/O variables are directly linked to the corresponding I/O devices. Input or output operations in the application program tie directly to corresponding input or output conditions of the actual field I/O devices. In Virtual Mode, I/O variables are processed exactly as internal variables. They can be read or updated by the debugger, so that the user can simulate the I/O processing, but no real world connection is made.

Technical notes

The "Tools / Technical note" command displays the on-line user's guide of the selected board or complex equipment. The board technical note is written by the hardware supplier of the I/O board. It contains all the information about I/O board management. It also describes the meaning of its parameters.

Removing connected variables

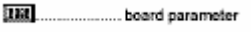
The "Tools / Free board channels" command disconnects all the I/O variables already connected on the selected board.

Defining comments for free channels

The "Tools / Free board channels" command disconnects all the I/O variables already connected on the selected board.

11.2 Setting board parameters

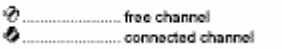
To set the value of a board parameter, the user has to double click on its name in the list on the right. It is also possible to select (highlight) it and choose the "Set channel/parameter" command of the "Edit" menu. Parameters are listed at the beginning of the list. The following icon is used to represent them in the list:



The meaning and input format of the parameter are designed by the supplier of the corresponding I/O board or equipment. Use the "Tools / Technical note" command or refer to your hardware manual for more information about board parameters.

11.3 Connecting I/O channels

To set the connection of a channel, the user has to double click on its location in the list on the right. It is also possible to select (highlight) it and run the "Edit / Set channel/parameter" command. The following icons are used to represent channels in the list.

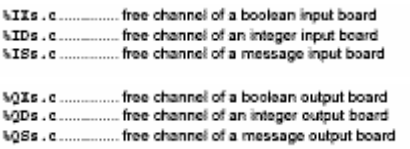


The list contains all the variables which match with the selected board type and direction. Only variables which are not yet connected are listed here. The "Connect" button connects the variable selected in the list to the selected channel. The "Free" button removes (disconnects) the variable from the selected channel. "Next" and "Previous" buttons are used to select another channel of the board. The location of the selected channel is always displayed in the title of the dialog box.

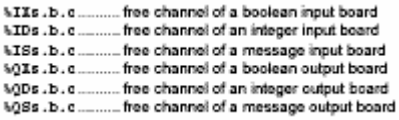
11.4 Directly represented variables

Free channels are the ones which are not linked to a declared I/O variable. ISaGRAF enables the use of directly represented variables in the source of the programs to represent a free channel. The identifier of a directly represented variable always begins with "%" character.

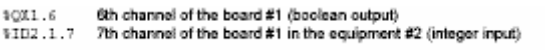
Below are the naming conventions of a directly represented variable for a channel of a single board. "s" is the slot number of the board. "c" is the number of the channel.



Below are the naming conventions of a directly represented variable for a channel of a complex equipment. "s" is the slot number of the equipment. "b" is the index of the single board within the complex equipment. "c" is the number of the channel.



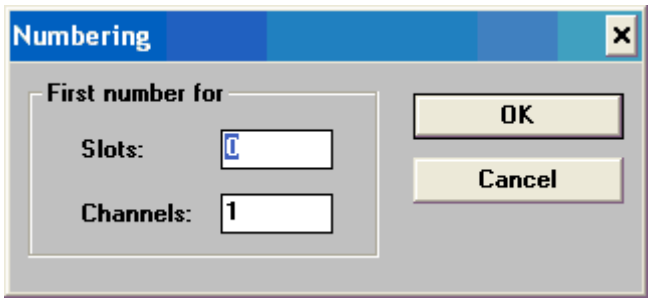
Below are examples:



A directly represented variable cannot have the "real" data type.

11.5 Numbering

Use the "Options / Numbering" command to set numbering conventions. You can specify the number used for the first slot and the number used for the first channel of each board in the following dialog box:



As default, slot numbering starts at index "0", and channel numbering starts at index "1".

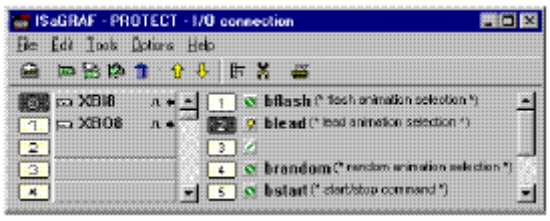
Warning: Be very careful while changing numbering conventions as it has effect on symbols used for directly represented variables and may lead to compiling errors if directly represented I/O variables are used in existing programs.

11.6 Setting individual protections

The ISaGRAF workbench provides a complete data protection system based on hierarchised passwords. I/O connection can be globally protected by a password. Additionally, ISaGRAF enables you to set individual protection to any I/O channel. This assumes that:

- passwords are already defined in the password definition system (use the "Project / Set password" command of the Project Management window) so that protection levels are available for individual protection.
- you use protection levels with higher priority for individual protection compared to global I/O protection.

When an I/O channel has individual protection, a small icon is draw close to its name in the I/O connection window:



Use the "Set protection" and "Remove protection" commands of the "Edit" menu to set or remove an individual protection for selected channel. Both commands ask you to enter a valid password so that a protection level can be attached to the channel. Then, each time you want to change connection to a channel having individual protection you must enter a password with sufficient priority level.

Warning: If a channel is protected with a level, and the corresponding password is removed from protection system, and if no higher level password is defined, connection to the channel cannot be changed anymore unless a new password with sufficient level is defined.

12. Creating conversion tables

The ISaGRAF workbench allows the user to create conversion tables. A conversion table is a set of points used to define an analog conversion. A conversion table can be attached to an analog input or output variable. A table creates a proportional relationship between electrical values (read on input sensor or sent to the output device) and physical values (used in application programming). Conversion tables are edited through a dialog box run by the "Tools / conversion" command in the ISaGRAF dictionary window

A defined conversion table can be used to filter values of any input or output analog variable of the selected project. Attaching a conversion table to a variable is made using commands of the ISaGRAF dictionary, the variable declaration editor. An input or output analog variable must then be selected and its parameters edited. A variable cannot be attached to a conversion table that is not already defined.

12.1 Main commands

The "Conversion tables" dialog box shows the list of defined conversion tables, and contains push buttons for main commands, to edit an existing table (define its points), to create a new table, and also to rename or delete a table. Press OK to quit the "Conversion tables" dialog box and save them on disk.

Creating a new table

The "New" command allows the user to create a new conversion table. Up to 127 conversion tables can be created for each project. Only used tables (the ones attached to analog variables) are inserted in the application executable code. Naming a table must conform to the following rules:

- the name cannot exceed 16 characters
- the first character must be a letter
- the following characters can be letters, digits or '_' character
- the table name is case insensitive

Changing the contents of a table

The "Edit " command is used to enter the points of a table selected from the list. It is also possible to double click on the name of the table. The "Edit" command is automatically called when a new table is created. At least two points must be entered for each table.

12.2 Entering points of a table

The "Edit" dialog box allows the user to define the points of a conversion table. The box shows on the left side the list of points already defined. The lower right box shows the defined table as a graphic curve. The points are entered by using the box commands. The user must comply with the number rules for the definition of points, described at the end of this chapter. The box on the left always contains the list of existing points for the currently edited table. The column on the left shows the electrical (external) value of the points. The column on the right shows the physical (internal) values. The user has to select a point on the list in order to modify its values or to clear (remove) it. The last choice of the list ("... ...") is used to define a new point. The box on the lower right shows the currently edited table as a graphical curve. No axes or co-ordinates are shown, as this is a proportional representation of the curve. This representation is useful as a quick check that the curve is properly defined.

- ***Defining a new point***

  When defining a new point, select the last entry ("... ...") on the list of points. This is also the default mode when starting to define a new conversion table. The user has to enter the electrical (external) and the physical (internal) values of each point. Values are stored as simple precision floating point numbers. Remember that at least two points have to be entered to define a curve. When both values are entered, pressing the "Store" button adds the point to the table. A maximum of 32 points can be defined for each conversion table.

- ***Modifying a point***

  To modify the values of an existing point, first select it from the list. The new electrical (external) and the physical (internal) values of the point can then be entered. Values are stored as simple precision floating point numbers. When both values are entered, pressing the "Store" button updates the point in the table.

- ***Clearing a point***

  An existing point is cleared by selecting it from the list and pressing the "Clear" button. Remember that at least two points must be entered to define a table.

12.3 Rules and limits

The rules shown below must be followed when defining a conversion table. The table can be used to convert both input and output analog variables:

- Two points cannot be defined with the same electrical value
- The curve must be continuously increasing or decreasing
- Two points cannot be defined with the same physical value

The following limits apply when defining conversion tables for a project:

- No more than 127 conversion tables can be defined in the same project
- No more than 32 points can be defined for the same conversion table.

13. Using the code generator

The code generation window is automatically opened by the "Verify" and "Make" commands of the other ISaGRAF Workbench windows. The code generation window is not automatically closed when the requested code generation operation ends, so that the user still has access to all the code generation commands and options from the window menu.

13.1 Main commands

The "Files" menu contains the commands for program syntax checking and code generation.

- ***Make application code***

The "Make" command constructs the entire code of the project. Before generating anything, this command checks the syntax of the declarations and programs. Any error that cannot be detected during single program compiling is detected during code generation. This applies to tables of conversion, I/O variable connections and links with the libraries. The code generation halts the compiling of a program when errors are detected. This program must be corrected before continuing the code generation. Programs which have already been checked (with no error detected) and that have not been modified since their last "Verify" operation are not recompiled. Variable declaration verification and application coherence checking are always processed. During program checking, the "Make" operation can be aborted by hitting the ESCAPE key.

Note: If the declaration of a local variable of a program has been modified, this program is verified. If a global variable has been modified, all the programs are verified.

Program syntax checking

The "Verify program" command allows the user to verify only one program. The selected program is compiled even if it has not been modified since its last verification. The "Verify dictionary" command allows the user to verify the declarations of all the variables of the project.

The "Verify all programs" checks the syntax of all the programs of the project, even if some of them have not been modified. This command does not stop when an error is detected in a program. It can be used to produce a complete listing of all the errors remaining in programs of the project. This command may be aborted by hitting the ESCAPE key.

Simulating a modification

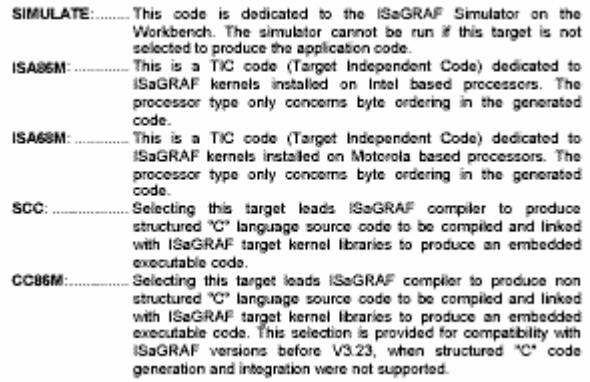
The "Touch" command simulates a modification of all the project's programs, so that they are all verified during the next "Make" operation. The "Open" command is used to open the last verified program. This command is very useful to directly access a program where syntax errors have been detected.

13.2 Compiler options

The "Compiler options" command is used to set-up main parameters used by the ISaGRAF Code Generator to build and optimise the target code. The aim of this command is to select the type of code which has to be generated, according to corresponding ISaGRAF targets, and to set-up the optimiser parameters according to the expected compiling time and application run-time requirements. The "Upload" button opens a second dialog box with other options that enable the embedding of zipped source code to downloaded code, in order to enable the "Upload" feature. Refer to "Upload" documentation for further explanations.

Selecting targets

The upper list shows the list of available target codes that can be produced. The ">>" sign is used to indicate the selected target(s). The ISaGRAF Code Generator can produce up to 3 different codes in the same compiling operation. Use the "Select" and "Unselect" buttons to set the list of required target codes, according to your target hardware. Below are the standard ISaGRAF targets:



SIMULATE: This code is dedicated to the ISaGRAF Simulator on the Workbench. The simulator cannot be run if this target is not selected to produce the application code.

ISA86M: This is a TIC code (Target Independent Code) dedicated to ISaGRAF kernels installed on Intel based processors. The processor type only concerns byte ordering in the generated code.

ISA68M: This is a TIC code (Target Independent Code) dedicated to ISaGRAF kernels installed on Motorola based processors. The processor type only concerns byte ordering in the generated code.

SCC: Selecting this target leads ISaGRAF compiler to produce structured "C" language source code to be compiled and linked with ISaGRAF target kernel libraries to produce an embedded executable code.

CC86M: Selecting this target leads ISaGRAF compiler to produce non structured "C" language source code to be compiled and linked with ISaGRAF target kernel libraries to produce an embedded executable code. This selection is provided for compatibility with ISaGRAF versions before V3.23, when structured "C" code generation and integration were not supported.

Refer to your hardware manual to know the type of ISaGRAF target kernel installed on your PLC. Other target types (machine code, C source code...) may be supported in future releases of the ISaGRAF Workbench.

SFC processing

Check the "Use embedded SFC engine" box to enable the use of the ISaGRAF SFC engine. This mode should be preferred as it leads to higher run time performances. However, the target engine may be missing on some particular implementations of the ISaGRAF target, of more commonly on customised targets based on ISaGRAF code post-processing. In this case you may have to remove this option and let ISaGRAF compiler translate SFC charts into low level instructions. Refer to your hardware documentation for more information about the use of this option.

Optimiser options

Below are the parameters, used by the ISaGRAF Code Generator to optimise the target code, that can be set from the "Compiler options" dialog box. The "Default" button is used to remove all optimising options, in order to reduce the compiling time.

When the "Run two optimiser passes" option is set, the ISaGRAF Code Optimiser is run twice. Optimisations made during the second pass are generally less significant than the ones made in the first pass.

- When the "Evaluate constant expressions" option is set, constant expressions are evaluated by the compiler. For example, the numerical expression "2 + 3" is replaced by "5" in the target code. When this option is not set, constant expressions are calculated at run-time.
- When the "Suppress unused labels" option is set, the Optimiser simplifies the system of jumps and labels of the programs, in order to suppress unused target labels or null jumps.
- When the "Optimise variable copying" option is set, the use of temporary variables (used to store intermediate results) is optimised. This option is commonly used with the "Optimise expressions" option. When this option is set, the Optimiser re-uses the result of expressions and sub-expressions which are used more than once in the program.
- When the "Suppress unused code" option is set, the Optimiser suppresses the code which is not significant. For example, if the following statements are programmed: "var := 1; var := X;", the corresponding generated code is only: "var := X;".
- When the "Optimise arithmetic operations" option is set, the Optimiser simplifies arithmetic operations according to special operands. For example, the expression "A + 0" will be replaced by the "A". When the "Optimise boolean operations" option is set, the Optimiser simplifies boolean operations according to special operands. For example, the boolean expression "A & A" will be replaced by "A".
- When the "Build binary decision diagrams" option is set, the Optimiser replaces the boolean equations (mixing AND, OR, XOR and NOT operators), by a reduced list of conditional jump operations. The translation is operated only if the expected execution time of the jump sequence is less than the one expected for the original expression.

The following table summarises the expected optimisation and requested compiling time corresponding to each parameter:

	gain (performances)	compiling time
Run 2 passes	xxxx	(*)
Optimise constant expressions	xxxxxxxx	xxxx
Suppress unused labels	xxxx	xxxxxxxx
Optimise variable copying	xxxx	xxxxxxxx
Optimise expressions	xxxx	xxxxxxxx
Suppress unused code	xxxx	xxxxxxxx
Optimise arithmetic operations	xxxxxxxx	xxxx
Optimise boolean operations	xxxxxxxx	xxxx
Build binary decision diagrams	xxxxxxxxxx	xxxxxxxxxx

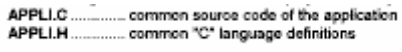
(*) The compiling time is also multiplied by 2.

13.3 Producing C source code

The ISaGRAF workbench enables the production of source code in "C" language. In this case, the whole contents of the application, including SFC chart description, data base definition and sequences of code are generated in "C" source code format. There are two possibilities, proposed as two styles of generated code:

CC86M	(C source code - V3.04) produces non-structured "C" source code. This style should be selected if your target software is based on ISaGRAF release previous to 3.23.
SCC	(structured C source code) produces a structured "C" source code. This style should be preferred if your target software is based on ISaGRAF release 3.23 or later.

The following two files are created in the project directory:



In the case structured "C" source code generation, a ".C" source file and a ".H" definition file are created for each program of the application, in addition to common "APPLI.C" and "APPLI.H" files. These files must be compiled and linked to the ISaGRAF target libraries in order to produce the final executable code. Refer to the "ISaGRAF I/O development toolkit User's Guide" for further information about recommended implementation techniques.

Note: In the case structured "C" source code generation, a ".C" source file and a ".H" definition file are created for each program of the application, in addition to common "APPLI.C" and "APPLI.H" files. These files must be compiled and linked to the ISaGRAF target libraries in order to produce the final executable code. Refer to the "ISaGRAF I/O development toolkit User's Guide" for further information about recommended implementation techniques.

13.4 Viewing information

The "Edit" menu contains the commands for viewing the different text files built during code generation or syntax checking operations on the code generator window. The code generation window is a text area that contains messages duringcode generation or syntax checking operations. All information is stored on the disk so it can be examined using the "Edit" menu commands

Editing commands

The "Clear Screen" command is used to clear the window text area. The window is automatically cleared before each code generation or syntax checking operation. The "Copy" command is used to copy the displayed text in the clipboard of Windows, so it can be used by other applications such as ISaGRAF text editors.

Viewing compiler output messages

The "Execution messages" command shows all the messages displayed during the last "Make" or "Verify" operation on the window text area. This applies to all the error messages.

Other choices of the "Edit" menu allow the user to monitor auxiliary text files created during syntax verification and code generation. These files are not usually used for a common ISaGRAF project.

13.5 Defining resources

The "Resources" command of the "Options" menu allows the user to define resources. A resource is any user-defined data (network configuration, hardware setting...) of any format (file, list of values) which has to be merged with the generated code, in order to be downloaded with it in the target PLC. Such data is not directly operated by the ISaGRAF kernel, and is commonly dedicated to other software installed on the target PLC. Refer to your hardware manual for further information about available resources.

The resource definition file

The resources are defined in a "Resource definition file" stored with other files of the ISaGRAF project. This is a pure ASCII text file, processed by the ISaGRAF Resource Compiler. This compiler is automatically run when the application code is built. This section explains the syntax of this file. The resource definition file uses lexical rules of the ST language. Comments, beginning with "(*" and ending with "*)" characters can be inserted anywhere in the text. Strings are delimited by single apostrophes. Refer to the second part of this manual for more explanations about the lexical formats used to enter numerical values.

Language reference

Below is the list of keywords and statements used in a resource definition file.

ULONGDATA

Meaning: Specifies a resource which is a list of integer values. Values are stored in target code as unsigned 32 bit integers. Values are stored in the order specified in the resource definition file. Values must be separated by comas. The name of the resource cannot exceed 15 characters.

Syntax:
```
ULONGDATA '<resource_name>'
BEGIN
  ...target_selection...
  ...list of values...
END
```

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

VARLIST

Meaning: Specifies a resource which is a list of variable addresses. Variables are identified by their name in the resource definition file. Variable addresses are stored in target code as unsigned 16 bit integers. Addresses are stored in the order specified in the resource definition file. Variables must be separated by comas. The name of the resource cannot exceed 15 characters.

Syntax:
```
VARLIST '<resource_name>'
BEGIN
  ...target_selection...
  ...list of variable names...
END
```

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

BINARYFILE

Meaning: Specifies a Binary File resource. The source data is stored in an MS-DOS file. The target resource definition is completed with a target pathname. End of line characters are not converted by the ISaGRAF Resource Compiler. The name of the resource cannot exceed 15 characters.

Syntax:
```
BINARYFILE '<resource_name>'
BEGIN
  ...target selection...
  FROM '<source_pathname>'
  TO '<destination_pathname>'
END
```

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

TEXTFILE

Meaning: Specifies a Text File resource. The source data is stored in an ASCII file. The target resource definition is completed with a target pathname. End of line characters are converted by the ISaGRAF Resource Compiler according to the target host system conventions. The name of the resource cannot exceed 18 characters.

Syntax:
```
TEXTFILE '<resource_name>'
BEGIN
  ...target selection...
  FROM '<source_pathname>'
  TO '<destination_pathname>'
END
```

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

TARGET

Meaning: Specifies the name of a target code that has to include the resource. Refer to the previous section (compiler options) for further information about handled targets. The "Target" statement can appear more than once in the same resource block, in order to select several targets. This statement cannot be used if the "AnyTarget" statement is specified.

Syntax:
```
TARGET '<target_name>'
```

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

ANYTARGET

Meaning: Specifies that the resource must be merged to all the target codes built by the Code Generator. The ISaGRAF Code Generator can produce several target codes during the same "Make" command. This statement cannot be used if one or several "Target" statements are specified.

Syntax:
```
ANYTARGET
```

Example:
```
ULongData 'MYDATA'

Begin
  AnyTarget
  ...
End
```

FROM

Meaning: Specifies the source pathname (on the PC where the ISaGRAF Workbench is installed) of a **BinaryFile** or **TextFile** resource. The characters used to isolate the components of the pathname (drive, directory, prefix, suffix) must conform to the MS-DOS system conventions.

Syntax:
```
FROM '<target pathname>'
```

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

TO

Meaning: Specifies the destination pathname (on the target system) of a **BinaryFile** or **TextFile** resource. The characters used to isolate the components of the pathname (drive, directory, prefix, suffix) must conform to the target host system conventions.

Syntax:
```
TO '<target pathname>'
```

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

Example

Below is a complete example of a resource definition file:

```
(* 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
```

Resource compiling

If resources have been entered in resource definition file, a dialog box appears at the end of ISaGRAF code generation. Press the "Start compile" button to run resource compiler. Output messages and errors will be displayed in the main control. Press "Exit" to avoid resource compiling. In this case, resources will not be added to the ISaGRAF code.

Implementation

The number of resources, the size of data rows and files are not limited by ISaGRAF. Resources are stored at the end of the generated code, with a resource directory. Below is the format (using C notations) of the resource directory format:

```
_RESOURCE:
{
  long nbres;                  /* number of defined resources */
  {
    char name[16];             /* resource name */
    long type;                 /* resource data type */
    long size;                 /* exact size of data block */
    uint32 data;
    uint32 path_offset;        /* points to a string */
  } /*nb of records */
}
```

Below are the possible values of the "type" field:

- 1 = binary file
- 2 = text file
- 3 = ulong data (path_offset field is not used in this case)
- 4 = variable list (path_offset field is not used in this case)

For text files, end of line characters are translated by the resource compiler, according to the target system conventions. All pointers are 32 bit offsets from the address of the corresponding structure. All resource names and pathnames are NULL terminated strings. Pathnames and data follow the resource directory.

14. Cross References

The ISaGRAF workbench includes a cross-reference editor which provides user with a total view of the declared variables in the project's programs, and where they are used. The aim of the cross reference is to list all the variables declared in the project, and to localise, at the source of each program the parts of source code where those variables are used. The cross-references are very useful for a global view of one variable life cycle. They help localise side effects, and reduce the time to understand the project during the maintenance. The cross-references may also be used for a global view of the complete dictionary of a project, so unused variables are easily found and the complexity of the project measured. The list on the left shows the declared objects of the project (programs, variables and defined words), and the library elements (functions and function blocks) referenced in the project. The list on the right shows the occurrences in the programs of the object currently selected in the first list. The description of an occurrence includes the program name, the number of the FC or SFC step, transition or test, plus line number for text languages or co-ordinates for LD or FBD diagrams. For quick LD diagrams, the description is completed with the number of the rung. If the variable is used as an output (on a coil) the rung number is followed by a star ("*") character. Set the "Show unused variables" option from the "Options" menu to display also in main list variables that are not used in the application programs.

Object type selection

Because a project can group a huge number of declared objects, the combo box in the editor toolbar is used to select the type of objects which must be listed in the window. This allows the user to have access to selected information. Each time the cross-references are re-calculated, the selection is reset to "All objects" in order to present the complete list.

Re-calculate cross-references

The "File / Re-calculate" command can be used at any time to update the cross references according to the modifications entered in other ISaGRAF editing

windows.

Export cross-references

The "Tools / Export" command is used to write the complete listing of the crossreferences in an ASCII text file. This file can then be opened with other applications such as Windows NotePad or word processors.

Dictionary errors

The "Edit / Dictionary errors" command displays in a dialog box the list of errors detected when the project dictionary was loaded.

Statistics

The "Tools / Statistics" command displays in a dialog box the number of objects and variables declared in the project, according to variable types and attributes. particular application of this command is to know the number of I/O variables declared in the project, in order to ensure that it can be compiled, if a limited version of the ISaGRAF Workbench is used.

Search in object list

The "Edit / Search" command allows the user to directly select an object in the editor list. The searched object cannot be found if it is not actually listed (when using a selected display). It is recommended, before searching for an object, to activate the "All" selection in the toolbar.

Open program

The list on the right contains the occurrences of the selected object in the source files and I/O connection of the open project. The "Edit / Open program" command enables the user to directly open a program where the object appears. It is also possible to double click the mouse on an occurrence (in the occurrence list) to open the corresponding program.

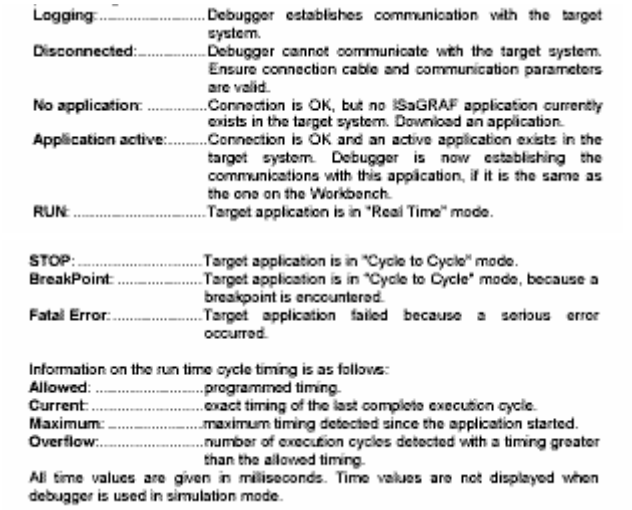
15. Using the graphic debugger

ISaGRAF includes a complete graphic and symbolic debugger. The "Debug" command of the program management window runs the debugger to control the application downloaded in the target PLC. In this mode, the debugger communicates with the target system via hardware link. The "Simulate" command of the program management window simultaneously runs the debugger and a complete target simulator. This enables the user to test his application when the target's I/O system is not yet complete. The debugger window contains the commands to control the entire application. When the debugger starts, and if the application in the target PLC is the same as the one on the workbench, it automatically opens the program management window, in debug mode. Commands of this window may be used to open other ISaGRAF windows (graphic and text editors, dictionary, lists of variables, I/O connection...). All windows opened during a debug session operate in "debug mode", meaning that the editing command is disabled. Displayed program components (steps, transitions, variables...) are shown with their current run time status or value. Double clicking on an object changes its status or value in the target application. When running the debugger in simulation mode, communication with the ISaGRAF target system is stopped. The debugger only communicates with the simulator window. Because the target system does not exist in this mode, the "download", "stop" or "activate" commands are not available on the debuggermenu.

15.1 The debugger window

The debugger window only contains information about the complete application status. It is linked to other ISaGRAF windows creating a complete interactive debug system. Detected run time errors are displayed in the bottom area of the debugger window. Commands from the "Options" menu are used to hide, show or clear the list of errors.

The control panel (area under the debugger menu) shows the global status of the target application, and information about the execution cycle timing. The list of possible target status is as follows:



15.2 Controlling the application

The "File" and "Control" menus contain the commands for the installation and the control of the currently edited ISaGRAF application on the ISaGRAF target system.

Note: Some of these commands are not available during simulation, because the application processed by the simulator is automatically installed by the ISaGRAF Workbench.

Stop the target application

The "File / Stop application" command stops the execution of the application currently active in the ISaGRAF target system.

Activate the target application

The "File / Start application" command runs the application existing in the target system. When an application is downloaded, it is automatically started, so that the "Start" command does not have to be used. The "Start" command is typically used after a "Stop" command.

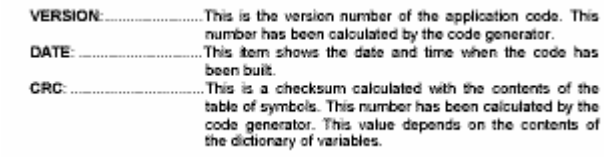
Note: the target application must be stopped (inactive) before it is possible to download a new application.

Download the application

The "File / Download" command is used to download the application code in the target system. Select the type of code to be downloaded, according to the target system processor and the application options.

Display version number

The "File / Get version number" command is used to display complete identification of both Workbench and target applications. The Workbench application is the one currently open on the ISaGRAF Workbench. The target application is the one executed in the target ISaGRAF PLC. The following items are displayed:



Note: The "Get version number" command is also available during simulation. In real debug mode, this command cannot be used if the target PLC is not connected.

On line modification

The "File / Update application" command enables the user to achieve "on line modification" of the running target application. This command is detailed in further sections of this chapter. It is not available when the debugger is used in simulation mode.

Real Time mode

The "Control / Real time" command is not available when no application is active. It sets the target application in normal "real time" mode: Normal mode: the execution cycles are triggered by the programmed cycle timing.

Cycle to Cycle mode

The "Control / Cycle to cycle" command is not available when no application is active. It sets the target application in normal "cycle to cycle" mode: In this mode, cycles are executed one by one, according to the "Execute one cycle" commands made by the user from the debugger menu.

Execute one cycle

When target is in cycle to cycle mode, the "Control / Execute one cycle" command runs the execution of one cycle.

The cycle timing

The "Control / Change cycle timing" command enables the user to modify the programmed cycle timing. This time is titled as "Allowed" in the debugger control bar window. The "Cycle to cycle" mode should be set before modifying the cycle timing. The cycle timing is entered as an integer number in milliseconds.

Remove all breakpoints

The "Control / Clear all breakpoints" command removes all the breakpoints currently installed (encountered or still active) in the whole application. Existing breakpoints are not automatically removed when the debugger window is closed.

Unlock I/O variables

The "Control / Unlock all IO variables" command unlocks all the I/O variables currently locked in the application. When an I/O variable is locked, no input or output status change is made to the corresponding I/O device. Variables attached to the I/O can still be written by the application or by the debugger. Currently locked I/O variables are not automatically unlocked when the debugger window is closed.

15.3 Options

The "Options" menu contains the options to control the information displayed in the debugger window.

The communication parameters

The communication timing parameters can be adjusted when the debugger is active. Only communication time-outs can be set here. Other communication parameters (baud rate, parity...) must be set from the "Debug" menu of the Program Management window. The "Communication time-out" is the time left for the target system to begin the answer to one workbench request. The "Cyclic refresh duration" is the time period required for the "read" requests to be sent by the debugger in order to refresh data in the opened windows. All the time values are displayed and entered as integer numbers in milliseconds. The communication timing parameters cannot be set when the debugger is used insimulation mode.

Display options

The "Show cycle timing" option enables the user to hide or show the cycle timing values in the debugger control bar. When this option is set, all the cycle timing components (allowed, current, maximum, overflows) are displayed and refreshed. Disabling this option reduces the debugger communication burden. When the "Show errors" option is set, detected run time errors are listed in the bottom area of the debugger window. When this option is disabled, the error list is closed. Removing this option reduces the debugger display and communication burden. The "Options / Clear errors" command clears the list of run-time errorscurrently displayed in the debugger window. The "Options / minimise window" command reduces the size of the debugger window so that it is shown as a small, always on top, panel containing only the application status and graphic buttons for most commonly used commands.

15.4 "Write" commands

The ISaGRAF symbolic debugger offers many commands to change the value or status of the application components. Selecting the component to be changed is done by double clicking on its name or its drawing in an editing window, when the debugger window is opened.

Variables

A variable status is changed by double clicking on its name in one of the following windows:

- Dictionary
- Lists of variables or time diagrams
- LD or FBD Programs

- I/O connection

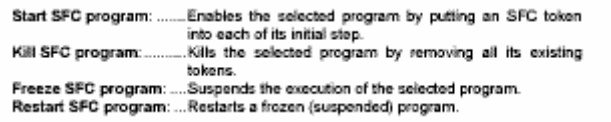
The following commands are offered in the debug dialog box:

- Write the variable to a new value
- Lock the variable (for I/O variables only)
- Unlock the variable (for locked I/O variables only)
- Start or stop a timer variable (set automatic refresh mode)

Symbolic values used to represent boolean FALSE and TRUE values are the strings defined for that specific boolean variable in the dictionary. The analog value specified for a "Write" command must be entered in an integer or real format, according to the variable definition in the dictionary. The string specified for a "Write" command for a message cannot be longer than the message capacity attached to that specific variable in the dictionary.

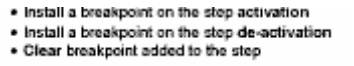
SFC objects

To observe a control operation on an SFC program while debugging the application, commands of the "File" menu are used in the Program Management window. The SFC program must be selected from the list of programs. The following commands are available:



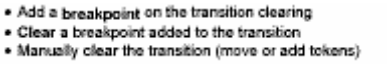
For child programs, these commands correspond to the "GSTART", "GKILL", "GFREEZE" and "GRST" functions in the programming language.

A control operation can be seen in an SFC step when debugging the application by double clicking on its graphic representation in the SFC editing window. The following commands are available in the debug dialog box:



Note: Activation and de-activation breakpoints cannot be added to the same step.

A control operation can be seen in an SFC transition when debugging the application by double clicking on its graphic representation in the SFC editing window. The following commands are proposed in a debug dialog box:



Conditional clearing: a token is created on the steps following the transition. The tokens existing in the preceding steps are removed. Unconditional clearing: a Conditional clearing: a token is created on the steps following the transition. The tokens existing in the preceding steps are removed. Unconditional clearing: a

15.5 On line modification

The "On line modification" feature enables the user to modify the application while the process is running. This is sometimes necessary for chemical processes where any interruption may jeopardise production or safety. This function should be used very carefully. ISaGRAF may not be able to detect all possible conflicts generated by user defined operations as a result of these on-line changes.

Code sequences

As ISaGRAF offers many possibilities for access to variables, programs or I/O boards from the debugger, the "On line modification" function described here applies only to the code sequences modification. A sequence of code is a complete set of ST, IL, LD or FBD instructions executed in a row. In a "beginning of cycle" or "end of cycle" program, a code sequence is the entire list of instructions written in the program. In an SFC program, a code sequence is the Level 2 programming of one step or transition. The "On line modification" consists of replacing one or more code sequences, without stopping the PLC execution cycle. As the control of the SFC tokens is very critical, it is not possible to modify an SFC structure, to add, renumber or remove a step, a transition or an SFC program.

Variables

As the variable database is a very critical part of the application, it can be accessed at any time by other processes (on multitasking PLC). It is also possible to modify variable values from the debugger. Therefore, ISaGRAF does not allow the user to add, rename or remove a variable on line. Anyway, it is possible to modify the way a variable is used in the application. It is also possible to reserve "unused" internal or I/O variables in the first version of the application, so that future modifications can make use of them.

They are different styles of variables in ISaGRAF target database. Limitations act on all of them:

- Declared variables

They are the ones declared using the ISaGRAF dictionary. They cannot be changed and cannot be renamed for on line change. It is recommended that some extra variables are declared and initialised in the application even if not used today. Such extra variables will enable future modifications to work on without changing the application data checksum.

- Instances of function blocks

Each instance of "C" or IEC written function block corresponds to data stored in ISaGRAF target real-time database. When function block instances are added or removed, On Line change is no more possible. So it is better to work in ST with FB instances declared in dictionary, rather than adding blocks (that will correspond to new automatically declared instances) in Quick LD or FBD diagrams. Also, any modification in the definition of available function blocks in the ISaGRAF library will lead to an impossible On Line change.

- Steps

Each SFC step corresponds to a piece of data where are stored SFC step dynamic attributes (its activity time and flag). Adding or removing SFC steps change the application database and is denied for On Line change.

- Hidden variables allocated by compilers

The ISaGRAF Compiler generates "hidden" temporary variables to solve complex expressions. In some case, the change of an expression may lead to a different set of non-visible temporary variables, and that leads to an impossible On Line change. To avoid this situation, you can add the following entries in ISA.INI file, in order to force a minimum number of temporary variables to be allocated for each program, even if not used for the compiling of the first application version. Values given here are examples:

```
[DEBUG]
MNTVboo=8       ; for booleans
MNTVana=4       ; for integers and reals
MNTVtmr=4       ; for timers
MNTVmsg=2       ; for messages
```

When such a setting is written in ISA.INI, the compiler outputs a warning message if a new compiling of the application leads to a greater number of allocated temporary variables.

Inputs and outputs

As the ISaGRAF I/O system is very open, required modifications should be implemented by the OEM, using specific features of the corresponding hardware. The ISaGRAF system does not allow the user to add, connect or remove an I/O variable, or to modify the description of an I/O board on line. Operations such as modifying board parameters and locking I/O channels are available using standard OEM features and the "OPERATE" function.

Run time operations

Modifying a running application consists of the following operations:

- modify the application source code on the workbench
- generate the new application code
- download the new application code using **"update"** command instead of **"download"**
- switch from the old application to the new one, in between PLC execution cycles using the **"Realise update"** command.

This procedure guarantees that the target PLC always has a complete and reliable running application, and enables the user to control the timing of the sample operations in a very safe and efficient way. It also enables the user to modify the project as often as possible. Regardless to the process itself, the "On line modification" is essentially the same as a normal "stop, start and download" set of commands. The only differences are that no variable state is lost, and the switching time is very short (usually 1 or 2 cycle duration). During the switch, no variable is modified, and all internal, input or output variables keeps the same value before and after the application modification. During the switch, no action is performed, and SFC tokens are not moved.

Memory requirements

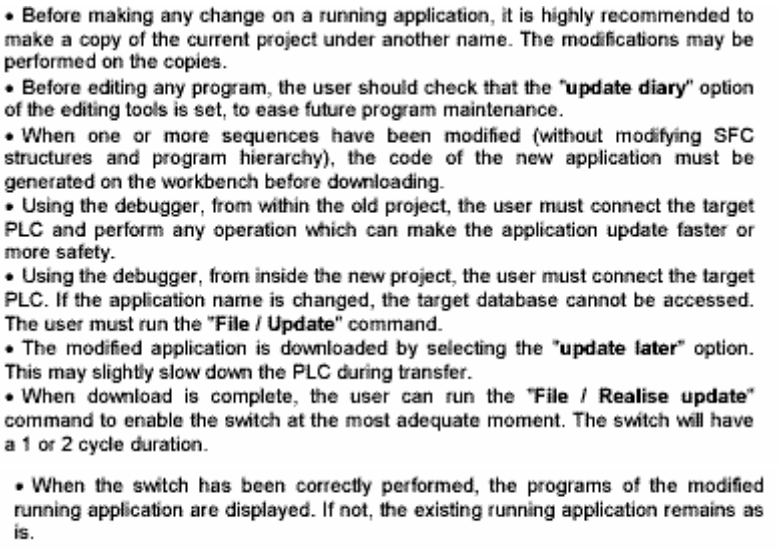
In order to support the "On line modification" capability, the target PLC must have free memory space to enable the storage of the modified version of the application code. Both versions of the application code have to be stored in PLC memory during the switch operation.

Limitations

As described before, only modifications to code sequences are allowed. Variable definition, application parameters and I/O connections cannot be modified. When downloading a modified version of the application, ISaGRAF makes a comparison between the modified application and the running one, in order to detect any unsafe change. If the switch seems dangerous or impossible, a download error is generated. One of the safeguards performed by ISaGRAF is to compare the symbol table checksum, so that any variable, program, or SFC element name change is detected. If a step is active when the switch occurs, its non-stored (N) actions are lost. The new step activation actions are not executed. Actions executed at the deactivation of the step are the ones carried over in the new application code. If a transition is valid when the switch occurs, its receptivity equation is updated. The new downloaded application code is not backed up on the PLC. The backup is the version which was previously downloaded with standard download commands.

Operations

To update the code of a running application, the following operations have to be performed:



15.6 DDE exchanges

The ISaGRAF debugger includes a DDE (Dynamic Data Exchange) server. An advice loop can be installed between the ISaGRAF debugger and other applications, in order to dynamically display the current value of variables in non- ISaGRAF applications.

Only "advise" and "poke" transactions are supported by the ISaGRAF debugger DDE server. You can use "request" transaction only for variables already spied in an advice loop. Other DDE services such as "execute" are not available. When an advice loop is established on a variable, the value of this variable is updated in the client application each time it changes. Variables of any type can be spied. The identification of the dynamic link includes the following names:

Service name: .... "ISaGRAF"

Topic name: ..... Name of the ISaGRAF project

Item name: ..... Name of the variable

If the variable is local to a program, its name must be followed by the name of its father program, written between parentheses, with the following syntax:

variable_name(program_name)

The ISaGRAF debugger DDE server is dedicated to the ISaGRAF application currently spied by the debugger. Up to 256 variables can be spied by the ISaGRAF server. The DDE server may be used when the ISaGRAF debugger runs in either connected or simulation mode. The refresh duration is the one established for communication between the debugger and the ISaGRAF target system or simulator.

16. Spying Lists of variables

The "Spy lists " command in the "Spy" menu of the Debugger window enables the user to build non-contiguous lists of variables which are refreshed with their current values. Lists are built when debugging the application. The lists can be stored on the disk and opened again during other debug sessions. A list may contain up to 32 variables. Variables of different types may be mixed in the same list. Global and local variables can be inserted in a list. A list of variables is dedicated to one particular project. Lists of variables are very useful for the functional testing of an application. They allow the user to watch the changes of a limited part of the controlled process, independent of the corresponding source code in the application programs. Lists of variables are also useful while debugging ST and IL text programs. The user can easily group in a list the set of variables used in a program, in order to control or monitor the execution of the programmed instructions. For each variable of the list, ISaGRAF displays its name, its current value and its comment text. Columns can be resized by dragging separation lines with mouse in the list title bar.

Saving lists on hard disk

The commands of the "File" menu are used to create, open and save the lists of variables. The number of lists for one project is not limited by ISaGRAF. While naming the lists of variables to be saved on disk, the rules shown below have to be followed:

- name cannot exceed 8 characters
- the first character must be a **letter**
- the following characters can be **letters**, **digits** or underscore character
- naming of lists is case insensitive

The list editor cannot display more that one list of variables at a time in the same window. However, the list editor can be run more than once, in order to spy different lists simultaneously.

Inserting variables in the list

The "Edit / Insert" command inserts another variable in the list. The variable name is selected in the list of objects defined in the project dictionary. This way the user does not have to manually enter the identifier. The variable is inserted before the variable currently selected in the list. The list cannot contain more than 32 variables. The same variable cannot appear more than once in the same list.

Changing the selected variable

The "Edit / Modify" command replaces the selected variable by another variable. You can also use the "Cut" command to remove the selected variable from the list.

Dump display

At any time, you can swap viewing mode between list and "Dump" view. Press the "zoom" button in toolbar or use "Options / Dump" command to swap viewing mode.

In "Dump" mode, only one variable value is displayed. Its value is displayed in numerical/symbolic format at the top of the window, and is also displayed in binary "dump" format. This mode allows you to spy hexadecimal value of each byte in the variable value.

"Dump" display is very useful for spying and understanding message strings containing non-printable characters.

17. Debugging ST and IL programs

During simulation or On Line debugging of ST and IL program, no modification can be entered in the program text.

For IL programs, instructions are formatted in a list view. Current value of a variable used in an instruction is displayed on the same line. You can double click on an instruction to change the value of the corresponding variable.

For ST programs, a Spy List window is embedded in the editor window. You can resize views by dragging with the mouse the separation line between them.

For each variable of the list, ISaGRAF displays its name, its current value and its comment text. Columns can be resized by dragging separation lines with mouse in the list title bar.

Saving list on hard disk

The "File / Save list" command save the lists of variables on the disk, under the same name as the edited program. This list will be automatically re-loaded each time ST or IL program is open in debug mode. This list can also be freely open and modified using the Spy List tool run by the "Spy / Spy list" command of the debugger window.

Inserting variables in the list

The "Edit / Insert variable" command inserts another variable in the list. The variable name is selected in the list of objects defined in the project dictionary. This way the user does not have to manually enter the identifier. The variable is inserted before the variable currently selected in the list. The list cannot contain more than 32 variables. The same variable cannot appear more than once in the same list.

When the name of a variable is highlighted in ST text, press this button in the toolbar or run the "Edit / Spy selection" command to directly send the variable to embedded spy list.

Changing the selected variable

The "Edit / Change variable" command replaces the selected variable by another Variable. You can also use the "Cut variable" command to remove the selected Variable from the list.

18. Debugging with SpotLight

ISaGRAF SpotLight tool allows the user to define watch lists that can be displayed either as graphic pictures or as lists during debug. Graphic items must be linked to the variables of the ISaGRAF project. The graphic picture is both defined and animated "on line".

To force the value of a variable, double click on the corresponding item from graphic or list layout, or hit ENTER when it is selected. You also can lock the document (deny any modification) using the "File / Lock" command. When a document is locked, you still can force variables by double clicking on their symbol.

18.1 Building the graphic layout

A chart is made of background pictures (bitmaps or metafiles), and a set of graphic items that will be animated during debug. To enter the chart, the following operations must be performed: Insert background

Background pictures

The background pictures are "bitmap" (.BMP) or “metafile” (.WMF) files. Numbers of pictures included in the graphic layout is not limited. Pictures can be moved or resized in graphic layout. They do not appear in list layout. Pictures are built with other tools. SpotLight does not include a painting tool. The "Options / Background colour" command is used to select a solid colour for empty space in graphic layout.

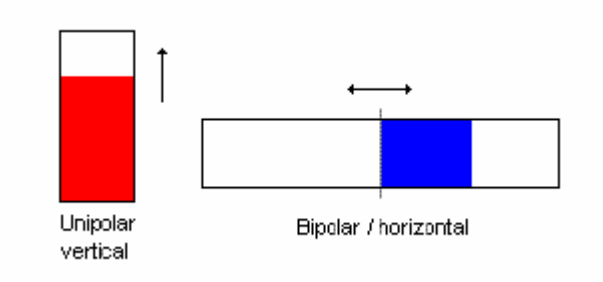
Note: Bitmaps consume a large amount of memory. It is highly recommended to correctly size the picture, and limit the unused space inside the bitmap rectangle.

Single text display

A “single text” item is a text written in a rectangle. The text displayed is the value of the attached variable. Thus, such item can be linked to message string variable. The rectangle where text is displayed can be either filled with colour or transparent. The character font used to display text is adjusted to fit the height of the rectangle when item is resized.

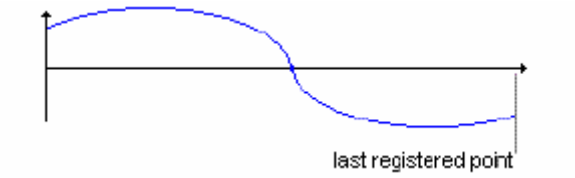
Unipolar and bipolar bar graphs

A bar graph is a rectangle with a coloured part that represents the numerical value of the attached variable. Optionally, the rest of the rectangle can be filled with colour. A bar graph can be either horizontal or vertical. Unipolar bar graphs can grow in any direction: to the top, to the bottom, to the left or to the right. Bipolar bar graphs can grow either in positive or negative direction, according to the value of attached variable. In case of a bipolar bar graph, the maximum allowed value is the same for both negative and positive scales.



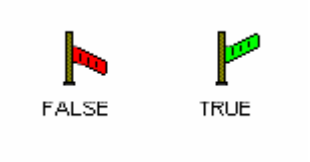
Curves

It is possible to insert a curve in a document. A curve shows the history of the attached variable. Although it is not a precise measurement tool, it can give useful debug information about synchronism between various variables. A curve stores the 200 last values of a variable. The number of samples is not changed when the curve item is resized in the graphic layout.



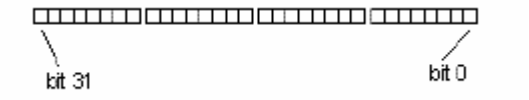
Boolean icons

A “boolean icon” item is used to display a binary state. One icon (.ICO) file is defined for FALSE or 0 value. Another icon is defined for all other non zero values. As SpotLight does not include an icon editor, icon files should be prepared with another tool.



Bit fields

A “bit field” item shows in a graphic panel the 32 bits of an integer value. The less significant bit is always displayed on the right. It is not recommended to use bit field for other data types such as real analog values, as the displayed information can lead to confusions.



Select, move or resize items

Selecting graphic objects is needed for most of the editing commands. SpotLight enables the selection of one or more existing objects in the chart area. To select objects, the "select" (button with an arrow) choice must be checked in the editor toolbar. To select one object, the user simply has to click on its symbol. To select a list of objects, drag the mouse in the drawing area to select a rectangle area. All the graphic objects that intersect the selection rectangle are marked as "selected". A selected object is drawn with little black squares around its graphic symbol.

By making a new selection, any previously selected objects are unselected. To remove the existing selection(s), simply click with the mouse on an empty area outside of the rectangle which borders the selected objects.

To move objects, you first have to select them. Then place the mouse cursor on the border of the selected item and drag it to other location.

To resize an object, you first have to select it. Then place the mouse cursor on one of the small rectangles displayed in the selection border, and drag it in appropriate direction to resize the object. Pictures can also be resize. In such case, the corresponding bitmap or metafile is stretched to fit the new specified item rectangle.

Group items / dissociate groups

You can group items together so that they are managed as one item. To make a group, select items in graphic layout and run the “Edit / Group” command. The “Edit / Dissociate” command is used to restore items of the selected group as separated ones.

A group may contain a picture. A group may also contain another group.When items are grouped, their style cannot be changed anymore. Items of the group are still displayed, but cannot be used (with double click) to modify the value of attached variables. A group appears at just one line in the list layout.

18.2 The list layout

At any time, you can swap between graphic and list layout, by pressing this button. You can also use the “Options / List - Graphic layout” command. In the list layout, items are shown in a classical list box. The height of each item is calculated according to its drawing style. Pictures (bitmaps and metafile) are not visible from the list layout. A selection is available in list layout, and should be used to set item style or change the value of a variable. Multiple selection and commands using it are not available in this mode.

You can re-order the items in the list using the “Edit / Move in list” commands. The item to be moved should be selected in the list.

18.3 Defining the item style

The graphic style and settings of an existing item can be modified, by double clicking on its symbol in the graphic area, or by running the “Edit / Set style” command when item is selected in graphic or list layout. The “Style” dialog box is also opened when a new item is added to the document. It groups the following pieces of information to be selected by the user:

- ***Graphic style and settings:***

  The display style (single text, bar graph, curve…) of an item can be changed dynamically. When foreground and background colours are used, they can be customised using the corresponding boxes. When style is “boolean icon”, the pathname of corresponding .ICO files has to be specified. Use “…” buttons close to these controls to browse icon files existing on the disk.

- ***Scale:***

  This is the maximum value that can be displayed in bar graphs and curves. For bipolar bar graphs and curves, the same absolute value is used for both positive and negative axis.

- ***Variable name:***

  When the "Name" field is the active field, pressing the "…" button close to edit control enables the user to find the names of the variables already declared in the project dictionary.

- ***Caption:***

  A caption can be displayed closed to the graphic item in graphic layout. You can customise the location of the caption text (top, bottom, left or right) and its contents. Caption can be any combination of the variable name and its value formatted as text. Caption customisation has no effect on list layout.

- ***Command variable:***

  If the "Command variable" option is set, the user can modify the value of the linked variable during debug by double clicking on the item graphic symbol.

18.4 Commands of the "File" menu

The "File" menu contains commands that allow the user to manage the complete document.

The "New" command of the "File" menu starts the editing of a new document. The number of documents defined for a project is not limited by ISaGRAF. Before editing the new chart, the previously opened chart is closed. The SpotLight cannot be used to edit several charts at once. However, multiple SpotLight windows can be opened simultaneously with each used to edit a different document.

The "Open" command of the "File" menu allows the user to close the currently edited document and to start editing another document of the current project. The new selected document replaces the current one in the editing window. When selecting the new document, the "Delete" button can be used to delete an existing file, in order to clean up the project directory. Icon and bitmap files referenced in a chart are not erased when the chart is deleted.

The "Save" command of the "File" menu stores the currently edited document on the disk. If it is a new untitled document, the user must give it a name before saving it. Naming a document must conform to the following rules:

- The length of the name cannot exceed **8** characters
- The first character must be a **letter**
- The following ones must be **letters**, **digits** or **underscore** characters
- Naming is case insensitive

The "Save as" command of the "File" menu allows the user to store the currently edited document under another name.

18.5 Note for ISaGRAF V3.2 users

Spotlight can read graphics and lists of time diagrams built with the tools of ISaGRAF V3.0 or V3.2. Such files appear in the "Open" dialog box, with the description of their origin. Files can be read and freely modified with SpotLight.

When opening an ISaGRAF V3.2 graphic, the document is automatically marked as "Locked". Remove the "Lock" option from the "File" menu if you want to make changes in the graphic.

When an ISaGRAF 3.2 graphic or list of time diagram is open, SpotLight always proposes to save it in native SpotLight format. The "Save As" dialog box is systematically open when closing such a document.

19. Uploading applications

ISaGRAF supports the uploading of the application stored in the target. The upload procedure communicates with the target to load the embedded zipped source code (EZS) and then restore the loaded project in the workbench environment. The project running on the connected target system can be uploaded if the target version is V3.22 or later, and if zipped source code have been embedded with the application. Embedding source code for upload is an optional feature.

19.1 Uploading a project

The "Upload" dialog box is run from the "Files" command of the ISaGRAF Project Manager. Upload does not refer to an existing project on the Workbench. The currently selected project in project management list has no relationship with upload mechanism. To upload the application running on the target you must:

1- ensure that the target is properly connected
2- set-up the communication parameters according to the connection link
3- press the "**Run**" button

Uploading embedded zipped source (EZS) and decompressing them may take few seconds. Messages in the dialog box will inform you when upload is complete, or in case of error.

The name used to create the ISaGRAF project is the one read in the target through communication. If this name is already used for an existing project in the workbench, you will be prompt to either overwrite it or select an unused name. You cannot cancel the registration of loaded sources as a project when upload is complete. The uploaded project is now ready and can be opened.

Possible errors

The following errors may occur when uploading a project. You are informed of the error in the "Upload" dialog box.

- communication cannot be established with the target
- connected target is an ISaGRAF system before version 3.22
- there is no application running in the target
- there is no EZS embedded in the target

19.2 Communication settings

Pressing the "Set-up" button enables the user to define the parameters of the link used for communication for upload between ISaGRAF workbench and the target ISaGRAF system. You have to ensure that the configured parameters match to the connected target before running upload.

19.3 Preparing a project for upload

You have to inform the ISaGRAF Code Generator that zipped source code must be embedded with the application code if you want to enable upload later. For this, press the "Upload" button in the "Compiler options" dialog box. Another dialog box enables you to check, as an option, the embedding of zipped source code. In this case, only minimum required source files will be embedded. Use other check boxes to embed also optional files.

Important note: Libraries are not downloaded with embedded source code. This includes functions and function blocks and I/O boards and equipment.

- ***Optional files***

  In addition to the minimum required source code, the following files can also be embedded. They are options as their selection leads to extra memory requirement on the target.

  Project descriptor: If not embedded, the project descriptor after upload will just indicate the upload date.

  Password protection: Upload function is not protected by a password. If you want the uploaded project protected, you have to embed its password protection system with source code.

  Comments for not connected I/O channels: ISaGRAF gives you the possibility to enter description text for non-connected I/O channels. Do not check this option is you work with connected I/Os only.

  History of modifications: This is the global history of modifications for the project. Diary files: Diary file of each program contains user written notes plus the history of compiler output messages referring to the program. Embedding diary files may consume a lot of memory in target.

  Lists of variables and time diagrams: These are the files created during debug, and containing lists of variable names for list or time diagram monitoring.

  Graphics, icons and bitmaps: This includes ISaGRAF graphics, plus all attached icon and bitmap files, if they are located in the project directory. Warning: embedding diary files may consume a lot of memory in target.

19.4 How zipped source are stored in the target

Embedded zipped source (EZS) is stored in generated code with resources. The generated resource is called "EZS". If source code embedding is selected, you cannot choose this name for another resource. Embedding source code does not imply any limitation in resource definition. The user written resource definition file is not affected by source embedding.

Please refer to the ISaGRAF documentation about the Code Generator for further details and information about resources.

19.5 Memory requirements on the target

Embedded zipped source (EZS) code requires extra memory to be stored with application code in the target. A general rough estimation is that minimum EZS (no extra option selected for source embedding) has one and a half the size of the executable code. This means that the embedding of EZS will multiply the size of downloaded code by 2.5.

Special limitation may appear on some target system based on segmented memory. As EZS are stored as resources in generated code, they must be stored in the same data segment as the application code.

19.6 About uploaded project

The uploaded project contains all the files and data required for re-compiling. Depending on the options selected during its previous compiling, it may also contain auxiliary files such as project descriptor and program diary files. You have to compile (make) the project before debugging or monitoring it. Warning: as ISaGRAF uses the compiling date stamp to compare applications, you will be informed when opening the debugger that workbench and target applications have different version codes.

Important note: Libraries are not downloaded with embedded source code. You have to ensure that the appropriate library functions and function blocks are installed with your ISaGRAF workbench before re-compiling the uploaded application.

19.7 Compatibility issues

Upload is supported by ISaGRAF target and workbench version 3.22 or later. Extensions have been made to the communication protocol to support upload. There is no restriction in embedding zipped source code (EZS) in a target based on ISaGRAF systems version 3.03 to 3.21, as EZS are stored in application code as standard resources. But embedded information cannot be uploaded in this case, as such target does not support required communication services.

20. Using the Diagnosis tool

The "Diagnosis Tool" is a subset under the ISaGRAF debugger tool. It enables the end user to work on a predefined set of variables, in order to examine and control the process. The ISaGRAF debugger is a very powerful tool, which includes high level functions. The Diagnosis Tool provides a safe way to control the target application for final running operations or maintenance. The ISaGRAF Diagnosis Tool is run directly from the ISaGRAF group in Program Manager, by double clicking on the following icon:

The list of existing projects is displayed in a dialog box. It enables the user to run the limited ISaGRAF debugger on an existing, already downloaded ISaGRAF application. Pressing the "OK" button starts the limited debugger on the selected project. Pressing the "Cancel" button closes the dialog box. The "Set-up" command is used to set-up the communication link between the ISaGRAF Workbench and the target PLC. Refer to the "Managing programs" chapter of this manual for more information about this command.

Note: The ISaGRAF Diagnosis Tool (limited debugger) cannot be used to download, stop or update the application running in the target PLC. No operation can be performed if the project selected in the Diagnosis Tool dialog box is not the same as the one installed and running in the PLC.

When the limited ISaGRAF debugger is run, and correctly connected to the target application, the following commands are available:

- Spy lists of variables
- Spy graphic documents with SpotLight

21. Using the ISaGRAF simulator

The ISaGRAF Kernel simulator is started with the debugger when the "Simulate" command of the "Debug" menu in the Program Management window is run. The kernel simulator is a complete ISaGRAF target system supporting ISaGRAF standard features and all the "C" functions and function blocks of the standard library delivered by CJ International. The I/O boards are graphically simulated in a window. Any type of I/O board can be simulated. The boards defined as "Virtual boards" during the I/O connection also appear in the simulation window.

21.1 Links with the debugger

The kernel simulator supports full communication with the ISaGRAF debugger, so any of the debug possibilities can be used during simulation. The kernel simulator always works on the current ISaGRAF application. During simulation, the debugger commands "Start", "Stop", "Download" or "Update" are no longer available. The simulator cannot be used if the "SIMULATE" target was not selected in compiler options before building the target code. Closing the simulator window implies that the debugger window (and any ISaGRAF window opened during the debug session) is also closed.

21.2 I/O simulation

I/O boards appear in the simulator window, titled by their name and slot number. Any of the ISaGRAF standard types of I/Os (boolean, analog or message) are handled. The channels of the input boards are displayed with special buttons and fields. The channels of the output boards are displayed with graphic status lights and data fields.

Boolean inputs: A boolean input is represented by a square green button. The number of the channel is displayed with the I/O button. The input value is TRUE when the button is pressed. Clicking on the button changes the corresponding I/O value. Use the right button of the mouse to set the input only when the button is pressed.

Boolean outputs: A boolean output is represented by a small circle. The number of the channel is displayed with the I/O. The output value is TRUE when the graphic symbol is highlighted.

Analog inputs: An analog input channel is a simple numerical field, where the value of the corresponding input can be entered. Clicking on the box displays the caret. A new value for the channel can then be entered. It is not necessary to use the ENTER key after input. Analog inputs can be entered in either decimal or hexadecimal base Use up/down buttons to increase or decrease the current value.

Analog outputs: An analog output channel is a numerical output field. The output value can be displayed as either a decimal or hexadecimal number. No action can be performed by the user on an output channel.

Message inputs: A message input channel is a simple text field, where the value of the corresponding input is entered. Clicking on the box displays the caret. A new value for the channel can then be entered. It is not necessary to use the ENTER key after input.

Message outputs: A message output channel is a text output field No action can be performed by the user on an output channel.

21.3 Library components

The ISaGRAF simulator fully supports the standard conversions, functions and function blocks, delivered by CJ International. Below is the list of supported objects:

Conversion functions:

bcd, scale

Functions:

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

Function blocks:

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

User defined conversions, "C" functions and function blocks are commonly not integrated with the ISaGRAF Simulator. Typically, such objects are designed to use software and hardware resources of the target system. Such resources are generally not available on the Windows system. The ISaGRAF Simulator provides the following standard behaviour for any user defined conversion, function or function block:

• When a new conversion is processed by the simulator, it is replaced by a "null" conversion. This means that the physical value of the analog variables is always equal to the electrical value (as entered or displayed on the Simulator panel).

• When a new "C" function or function block is run by the simulator, it does not process any operation. The result value is not set.

21.4 Options

The commands of the "Options" menu enable the user to control the display of I/Os in the simulator panel. The user can set or remove these options at any time during debug.

When the "Colour display" option is set, I/O channels are displayed as colour bitmaps. If colours cannot be distinguished on some LCD screens, the user should remove this option, to get pure black and white input and output graphics for I/O channels.

When the "Variable names" option is set, a sticker is displayed beside any I/O channel, with the name of the connected I/O variable. Removing this option enables the user to reduce the size of the simulator panel.

- When the "Hexadecimal values" option is set, any input or output analog channel is displayed or entered in hexadecimal format.
- When the "Always on top" option is set, the simulator window is always visible, even if the input focus is on another window.

21.5 Saving and restoring input states

Using the ISaGRAF simulator, input channels are forced through manual operations, acting on toggle buttons and edit controls of the simulation panel. You can at any time use the following commands of the "Tools" menu to save and restore the state of all input channels:

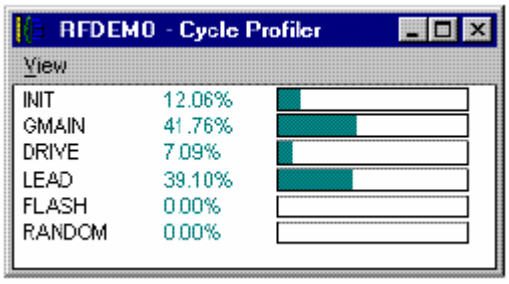
Load input scheme	Set values of input channels with values stored in a file that has been created on disk by "Save input scheme" command.
Save input scheme	Save state of input channels in a file so that they can be restored later using the "Load input scheme" command. File is stored in the project directory and thus is saved with other project files by the ISaGRAF archive utility.

Note: Only named input channels (the ones having a variable connected) are saved on disk.

21.6 The cycle profiler

The ISaGRAF Cycle Profiler is a powerful diagnostic tool that shows how cycle time is distributed between various programs, functions and function blocks of an application. This tool is very useful to have a quick diagnostic on the application performances, and leads the programmer to the parts of code which may need optimisations.

The Cycle Profiler is run by the "Tools / Cycle Profiler" command in the menus of the ISaGRAF Simulator window. It displays, for each program, function or function block, the percentage of the cycle time spent to execute it:



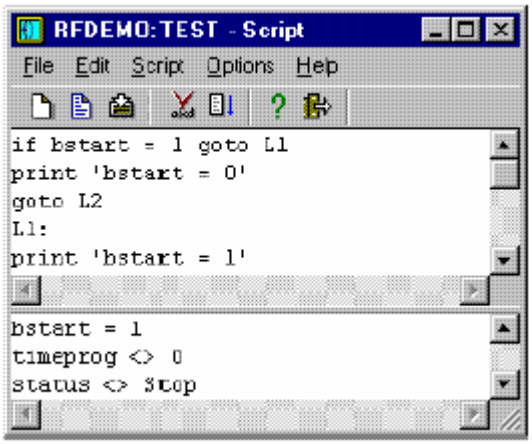
When the "View / Average" option is set, displayed information is an average of percentages calculated since the application has been started, or since the last time the "View / Reset" command has been run. If the "View / Average" option is not set, displayed information shows measurements done during the execution of the last cycle. You can also use this feature when the application is in "Cycle to Cycle" mode to have a set of measurements depending on the application context. Use the "View / Copy" command to copy program names and percentages to the Windows Clipboard in ASCII format. Then, data can be pasted into text documents or common Spread Sheets.

Important notes:

These are not precise measurements. Percentage calculation is based on TIC instructions counting, taking into account various instruction execution times. Calculation does not include the time spent in "C" functions and function blocks. The value displayed for a function or a function block is the sum of all the "calling times" from the application programs in the same cycle. Time calculation is based on TIC code and does not provide reliable information if the actual application code is generated in "C" language and built using a "C" compiler.

21.7 Simulation scripts

ISaGRAF simulator includes a tool to build and run simulation script. A script is described with an easy ST like text language, and is used to automate tests with the ISaGRAF simulator. The simulation script editor is run by the "Tools / Simulation scripts" command of the Simulator window. Below is the frame of the script editor:



The upper window is a text editor where script instructions are entered. It is used as other ISaGRAF text editors and includes high level features such as mouse selection of a variable symbol. You can use commands of the "Options" menu to set-up tab width and select a character font. The lower window shows all the messages output when the script is run. The separation line between windows can be freely dragged to resize windows. The output window can be hidden during script editing, but is automatically open each time a script is run.

- ***Editing scripts***

  Use the commands of the "File" menu to manage script files:

New ..... creates a new untitled script
Open..... loads an existing script from file
Save..... saves script text and contents of output window to disk, in project directory
Save as..... saves script under another name

Two files are created in the ISaGRAF project directory for each script:

<*scriptname*>.SCC .....text of the script (instructions)
<*scriptname*>.SCO.....contents of the output window

where <scriptname> is the name of the script. Both files are standard text files and can be open using any other text editor.

While editing a script, you can use the "Edit / Insert symbol" command to select a declared variable name to be inserted at the caret position.

Running scripts

Script must be checked and compiled before running it. If necessary, syntax checking is automatically performed on a "Run" command. Use the following commands of the "Script" menu:

Check .....check syntax and compile script

Run Script.....start execution of the script currently edited

In the case of a new untitled script, it must be saved (and a name must be entered for it) before it is checked. In case of a named script, script is automatically saved to disk before syntax checking. When script is running, its contents cannot be changed. A message is displayedwhen end of script is reached. You can also abort a running script using the following command of the "Script" menu:

Abort Script.....terminates the running script

Script execution is performed between target cycles. In the case of an infinite loop programmed in the cycle, ISaGRAF simulator ensures that this loop is always broken so that ISaGRAF cycles are still executed and other ISaGRAF applications are not blocked. The ISaGRAF script interpreter decides to break script execution if the same "label" is encountered more than once in the same target cycle. Script execution can also be normally broken by "Cycle" or "Wait" instructions.

Script description language

Script description language is a very simple text language similar to ST, but where each instruction is entered on a separate text line, and does not need to be terminated by a semicolon. Use the following button of the toolbar to know the list of available instructions and to insert a keyword at the caret position:

insert instruction (keyword and help as comments)

There are various types of instructions. First is the assignment (forcing) of a variable:

:= ..... assignment

Other instructions allow the output of messages to the output window:

Print..... outputs a text string or a variable value

PrintTime ..... outputs current time stamp

Other instructions are used to synchronise script instructions with ISaGRAF cycle:

Cycle..... let ISaGRAF simulator execute one cycle

Wait ..... waits during a specified time

Other instructions are used to control instruction flow in the script:

Labels..... can be placed anywhere in the script

Goto..... unconditional jump to a label

If goto..... conditional jump to a label

End..... terminates script

Script language is not case sensitive. Comments can be inserted at the end of any text line. Comments can either be written according to ST conventions (between "(*" and "*)" characters), or prefixed by a ";" character.

":=" Assignment

Meaning: Force the value of an ISaGRAF variable. It can be an internal variable, an input channel or an output channel.

Syntax:
```
<varname> := <constant_expression>
<varname> = <constant_expression>
```

Arguments: <varname> is a valid symbol of a declared application variable, or a directly represented I/O variable using "%" writing conventions.

<constant_expression> is a valid constant expression that matches the type of the specified variable. For booleans, "0" and "1" can be used instead of "FALSE" and "TRUE". For timers, the "T#" or "TIME#" prefix can be omitted.

Notes: Input variable forced by a script does not need to be locked. The drawing of the corresponding input channel is updated when input variable is forced by a script.

Warning: do not force input or output analog variable attached to a conversion, as script execution does not support conversion functions or tables.

Example:
```
MyBooVar := 1          (* same as TRUE *)
MyIntVar := 1234
MyRealVar := 1.2345
MyMsgVar := 'Hello'
MyTmrVar := t#12s
```

Print

Meaning: Writes a string or the value of a variable in the output window. Text is output as one new line at the end of text already written in output window.

Syntax:
```
Print '<text>'
Print <varname>
```

Arguments: <text> is any text string expressed between single quotes

<varname> is the valid symbol of a declared application variable, or a directly represented I/O variable using "%" writing conventions.

Notes: Output of variable values is always formatted according to IEC conventions.

Example:
```
Print 'Hello'
Print MyBooVar
```

Output:
```
Hello
MyBoovar = TRUE
```

PrintTime

Meaning: Writes the current time stamp in the output window. Text is output as one new line at the end of text already written in output window.

Syntax: `PrintTime`

Notes: Time stamp is formatted according to current setting of Windows System

Example:
```
Print 'Time now is:'
PrintTime
```

Output:
```
Time now is:
15:45:22
```

Cycle

Meaning: Suspends the execution of the script until the next ISaGRAF cycle is performed.

Syntax: `Cycle`

Notes: Script instructions are executed at the beginning of an ISaGRAF cycle. If the simulator is in "Cycle to Cycle" mode, the "cycle" instruction is immediately followed by a cycle. The following instructions of the script will be performed on the next "Execute one cycle" command from the debugger.

Example:
```
(* the ISaGRAF program copies A to B *)
A := 0
Cycle
Print B
A := 1
Print B (* no cycle performed / B not set to 1 *)
Cycle
Print B
```

Output:
```
B = 0
B = 0
B = 1
```

Wait

Meaning: Suspends the execution of the script until a delay is elapsed

Syntax: `Wait <delay>`

Arguments: `<delay>` delay expressed according to IEC conventions for time constant expression. The "T#" or "TIME#" prefix can be omitted. Delay value must between 10 milliseconds and 1 hour.

Notes: Accuracy of the "Wait" instruction is not precise as it depends on the host Windows system. Also, the delay should be considered with an accuracy of plus or minus one ISaGRAF cycle. When a "Wait" instruction is reached, ISaGRAF cycles are performed until the delay is elapsed and before continuing the script execution.

Arguments: `<delay>` delay expressed according to IEC conventions for time constant expression. The "T#" or "TIME#" prefix can be omitted. Delay value must between 10 milliseconds and 1 hour.

Notes: Accuracy of the "Wait" instruction is not precise as it depends on the host Windows system. Also, the delay should be considered with an accuracy of plus or minus one ISaGRAF cycle. When a "Wait" instruction is reached, ISaGRAF cycles are performed until the delay is elapsed and before continuing the script execution.

Labels

Meaning: Labels can be placed anywhere in the script. They are used as a destination by "Goto" instructions and allow flow control for script instructions.

Syntax: `<labelname>:`

Arguments: `<labelname>` unique name according to ISaGRAF variable naming conventions: limited to 16 characters, beginning with a letter, followed by letters, digits or underscore characters. When defined, label name should be followed by a ":" character.

Notes: No instruction should be placed on the line where a label is defined. Label name should not be the same as a declared ISaGRAF variable symbol

Example:

```
(* example of a script with an infinite loop *)
loop:
PrintTime
Wait 1s
Goto loop
```

Goto

Meaning: Unconditional jump to a label

Syntax: `Goto <labelname>`

Arguments: `<labelname>` is the name of a label defined in the script.

Notes: Backward jumps are allowed. In case of an infinite loop, script execution is automatically broken on each loop in order to preserve execution of ISaGRAF cycles.

Example:

```
Print 'Before Jump'
Goto MyLabel
Print 'Within Jump'  (*never performed *)
MyLabel:
Print 'After Jump'
```

Output:

```
Before Jump
After Jump
```

If Goto

Meaning: Conditional jump to a label. The condition is either a comparison between two ISaGRAF variables, or a comparison between a variable and a constant expression.

Syntax:
```
If <var1> test <var2> Goto <labelname>
If <var1> test <constant_expr> Goto <labelname>
```

Available comparison ***tests*** are:

=	true if both members have same value
<>	true if members have different values
<	true if first member is less than second
<=	true if first member is less than or equal to second member
>	true if first member is greater than second
>=	true if first member is greater than or equal to second member

Arguments: `<var1>` `<var2>` are valid symbols of declared application variables, or directly represented I/O variables using "%" writing conventions.

`<constant_expr>` is a valid constant expression that matches the type of specified variable. For booleans, "0" and "1" can be used instead of "FALSE" and "TRUE". For timers, the "T#" or "TIME#" prefix can be omitted.

`<labelname>` is the name of a label defined in the script.

Notes: Backward jumps are allowed. In case of an infinite loop, script execution is automatically broken on each loop in order to preserve execution of ISaGRAF cycles.

Example:
```
(* This script loops until MyVar is TRUE *)
Loop:
If MyVar = TRUE Goto TheEnd

 Print MyVar
 Goto Loop
 TheEnd:
```

End

Meaning: Terminates script

Syntax:
```
End
```

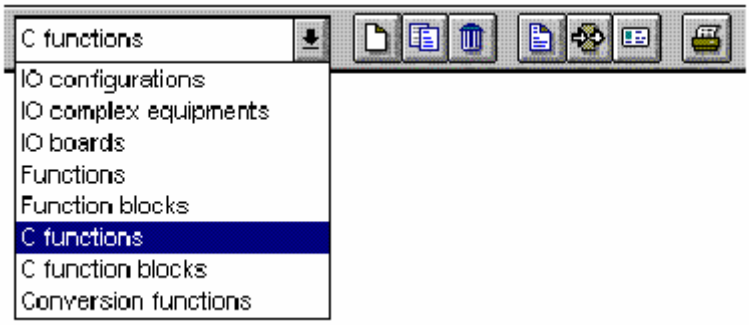
Notes: It is not mandatory to place an "End" instruction on the last line of the script

Example:
```
(* This script loops until MyVar is TRUE *)
Loop:
If MyVar = FALSE Goto Continue
End
Continue:
Print MyVar
Goto Loop
```

22. Using the Library Manager

The ISaGRAF libraries provide a standard interface between automation development and the software or hardware capabilities of the ISaGRAF target system. There is one library for each type of interface. The ISaGRAF Workbench Library Manager is dedicated to the hardware supplier, or to the software engineer. He uses the library manager to describe the ISaGRAF programming interface of the objects he creates.

The ISaGRAF Workbench Library Manager shows the elements of one of the ISaGRAF libraries. In the left area of the window is the list of the elements of the selected library. In the right area is the technical note (user manual) of the element currently selected on the element list. The menus of the library manager contain the commands to create, define or modify elements of the active library. The "File / Other library" command allows the selection of one of the ISaGRAF libraries. The combo box on the left of the toolbar can also be used to select a library:



22.1 Managing library elements

Use the commands of the "File" menu to create elements and work on existing ones in the open library

Creating a new element

The "New" command of the "File" menu creates a new element into the selected library. The name of the new element is entered, based on the following naming rules:

- the maximum length of a name is **8** characters
- the first character must be a **letter**
- the following characters must be **letters**, **digits** or '_' character
- the naming of a library element is case insensitive.

A text comment is associated to each library element. This comment is entered while creating the element. When a new element is created, the following must be entered:

- its definition for an I/O configuration,
- its parameters for an I/O board,
- its user interface for a function or function block.

When a "C" conversion, "C" function or "C" function block is created, a complete frame of its source code is automatically generated.

Working on existing elements

The "File / Rename" command allows the user to change the name or the comment of the element selected from the list of elements. The "File / Copy" command allows the user to copy the element highlighted in the active library on another element in the same library. If the destination element already exists, all its contents are overwritten. If the destination element does not exist, it is automatically created. The "File / Delete" command removes the currently selected element from the active library. The following components of the element are handled by "Rename", "Copy" and "Delete" commands:

- technical note
- complete definition for an I/O configuration
- parameters for an I/O board or complex equipment
- interface definition for a function or function block
- source code for function and function block written in IEC language
- source code for a C conversion, a function or a function block

If the element is a "C" conversion, "C" function or "C" function block, its name is not automatically updated in the attached source code by a "Rename" or "Copy" command.

If the element is a function written in IEC language, the return parameter name is not changed by a "Rename" or "Copy" command.

Setting password protection

The "File / Set password" command enables the user to define password protection for the selected element in the open library. Refer to the "Password protection" section, at the end of the first part in this manual for further information about password levels and data protection. Passwords are only relative to the selected element. They have no influence on other elements of ISaGRAF

Compiling functions and function blocks

When the library of functions or function blocks written in IEC languages is selected, the "Verify (compile)" command of the "File" menu is used to check the syntax of the selected element and create its object code. Functions and blocks written in IEC languages have to be compiled without errors before they can be used in ISaGRAF projects. This command has no effect if another library is selected.

Technical notes

The "Edit / Technical note" command allows the user to enter the technical note of the element selected in the active library. The technical note is entered with the ISaGRAF text editor. The technical note of an element is its user guide. It will be consulted by the user of the element during its integration in an ISaGRAF project. The technical note on how to use the element should contain the description of its main function, the detailed explanation of its programming interface and parameters, and its context and limits.

The **"Tools / Standard note format"** command allows the user to define a standard text format for all the elements of the currently selected library. When editing the technical note for a new element, this format is used as a main frame. This allows the user to optimise technical note editing.

Parameters

The parameters of an element describe the i**nterface** between the computer operations provided by the element and the use of the element in an ISaGRAF application. Parameters have a different meaning for each type of library element.The parameters of an I/O configuration define the complete set of I/O boards of the configuration, and default variable names used for I/O channels. The parameters of an I/O board or complex equipment define the physical and logical configuration of the board. The parameters of a function or function block define the interface of the element, according to ST language function calling conventions. There is no parameter for a conversion function because it uses a standard pre-defined interface.

Source code

The ISaGRAF Workbench allows the programmer to manage the source code of a library conversion, function or function block. The source code of a function or a block written in IEC language is a text or a diagram described with the language attached to the function. The source code of "C" components ("C" functions, "C" function blocks and conversion functions) is divided in two separate files: a **source header** that contains the exact definition of the **interface**, according to the element's parameter definition and a **source code** file that contains the element's operation implementation.

The ISaGRAF workbench generates the source code file when a new library element is created. It also creates and updates the source header, based on the parameter definition. The programmer can use the ISaGRAF text editor to complete the source code file.

Archiving library elements

The "Tools / Archive" menu command runs the ISaGRAF archive manager to save or restore library elements. You first need to select a library before running the "Archive command". The archive manager shows list of elements for only one library at a time.

22.2 I/O configuration

The ISaGRAF I/O configuration library provides an easy way to initialise new ISaGRAF projects with pre-defined I/O configuration. An I/O configuration defines:

- a set of I/O boards
- default values for I/O boards parameters
- default names for I/O channels

When a new ISaGRAF project is created with a library I/O configuration, the corresponding I/O connection is automatically set, and the I/O variables. When a new ISaGRAF project is created with a library I/O configuration, the corresponding I/O connection is automatically set, and the I/O variables

The definition of an I/O configuration is made with the ISaGRAF I/O Connection tool (the same tool used within a project). Refer to the "I/O Connection" section in this manual for further information about how to use this tool. When inserting a new I/O board in the configuration, all the channels of the new board are declared with standard default names. The standard default name of an I/O channel has the following format:

<direction><type><slot_number>_<channel_number>

The first character indicates the direction of the I/O channel:
"**I**" ..... input channel
"**Q**" ..... output channel

The second character indicates the type of the I/O channel:
"**X**"..... boolean
"**D**"..... analog
"**M**" ..... message

The second character indicates the type of the I/O channel:
"**X**"..... boolean
"**D**"..... analog
"**M**" ..... message

The "Connect I/O channels" command of the I/O Connection Editor is used to modify the default name attached to an I/O channel.

22.3 I/O complex equipment

All the channels of a single board have the same type (boolean, analog or message) and direction (input or output). A complex I/O equipment represents an I/O device with channels of different types or directions. A complex I/O equipment is represented as a list of single I/O boards. It uses only one slot in the I/O connection rack list.

To define a complex I/O equipment, the user has to define the list of single boards which define the I/O equipment. He also has to enter the detailed parameters of each single board. The list of single I/O boards is entered through a dialog box. Pressing the "Append" button allows the user to add a single board at the end of the current list. The "Insert" button is used to insert a new single board before the one currently selected in the list. The "Delete" button removes the selected single board from the list. The "Rename" and "Parameters" button are used to change the name and the parameters of the selected single board. Refer to the following section for a complete explanation of single board parameters. A complex I/O equipment can group up to 16 single I/O boards. The name of a single board (within an I/O equipment) cannot exceed 8 characters.

22.4 I/O board

The ISaGRAF I/O board library defines a standard interface between the application variables and the target hardware. During the description of the application, all the I/O variables are connected to the channels of the target I/O boards. An ISaGRAF I/O board is defined by a name and an "OEM key code" that identifies its supplier. Other I/O board parameters describe the I/O board topology (number of channels, channel direction and type), and its hardware or software configuration.

I/O board parameters

There are two different types of parameters for an I/O board: common parameters which are defined for any ISaGRAF library board, and OEM parameters which are specific to the board implementation, provided by the hardware supplier. Common parameters are entered in the upper part of the I/O board parameters definition box. These parameters (plus the I/O board name) identify the ISaGRAF standard I/O board interface.

The "OEM key code" is a simple number that defines the hardware supplier. All the boards defined by the same supplier must have the same OEM key code. The OEM key code is a 16 bit unsigned word, entered in a hexadecimal format. The reserved OEM key code for CJ International is "1".

Main parameters define the topology of the I/O board. The number of channels defines the number of available channels on the board. The type of the board is the type of the variables that may be connected on the channels of the board. The direction defines whether variables connected on the board are input or output variables.

Note: I/O variables of different types or directions cannot be grouped on the same ISaGRAF I/O board. This feature should require a complex I/O equipment.

The OEM parameters

The OEM parameters are entered in the lower part of the I/O board parameters definition box. These parameters are defined by the I/O board hardware supplier and are specific to the board. There are at most 16 OEM parameters for a board. A board may have no OEM parameters. The ISaGRAF library manager allows the hardware supplier to define the identification and the format of each parameter, and the way the automation programmer enters it.

The box on the left contains the list of the OEM parameters. Each parameter is identified by a name and a logical number, from 0 to 15. The area on the righ contains the detailed description of the parameter selected on the list. A parameter is selected in the list in order to access to its complete description. Pressing the "Clear" button resets the parameter description, and removes it from the parameter list. **Warning**: this command cannot be "undone".

The name of a parameter is used to identify the corresponding input field during the I/O board connection if the field must be defined by the automation operator. The name of a parameter must conform to the following rules:

- the maximum length of a name is **16** characters
- the first character must be a **letter**
- the following characters must be **letters**, **digits** or '_' character

The type of a parameter defines the internal format of the parameter, and its input format during application I/O connection. Below is the list of available internal formats:

word ..... unsigned 16 bit word
long ..... unsigned 32 bit word
word hexa ..... unsigned 16 bit word
long hexa ..... unsigned 32 bit word
boolean ..... unsigned 16 bit word (only lowest bit is used)
character ..... unsigned 16 bit word (only lowest byte is used)
string ..... array of 16 bytes containing a null-terminated string
float ..... single precision 32 bit floating value

Below are available input formats:

word ..... unsigned decimal word
long ..... decimal long word
word hexa ..... unsigned hexadecimal word
long hexa ..... unsigned hexadecimal long word
boolean ..... "true" or "false"
character ..... single character
string ..... ascii string (15 characters max)
float ..... single precision floating value

The "access" box is used to define how the parameter can be accessed by the end user. If the "User defined" option is set, the parameter is shown as an input field during the I/O board connection. The OEM parameter default value is used as default for the parameter editing. If the "Hidden" option is set, the parameter is a constant and does not appear in the I/O board connection box. The OEM parameter default value defines the value of the constant parameter. The "Read only" option indicates that the parameter is visible for the user, but cannot be modified. Its default value is used as a constant value.

22.5 Functions and blocks written in IEC languages

ISaGRAF handles a library of functions and function blocks written in IEC languages. The available languages to describe such a function or block are FBD (Function Block Diagram), LD (Ladder Diagram), ST (Structured Text) or IL (Instruction list). Note that LD and FBD languages can be mixed in the same diagram. SFC language (Sequential Function Chart) cannot be used to describe a function or a block in library. The language attached to a library element is selected when the function is created, and cannot be changed later.

Compiling

Functions and blocks defined in the library must be compiled (verified) before they can be used within an ISaGRAF project. Nothing else has to be changed on the Library side concerning functions and blocks. Elements of the library will automatically appear in box selection menu when using the LD/FBD graphic editor within a project.

⚠ A function defined in the library can call other functions of the library. However, the ISaGRAF system does not support recursive function calling. A function block written in IEC language cannot call other function blocks (neither in IEC nor in "C" language).

Entering source code

The source code of a library function or function block is entered using standard ISaGRAF tools: graphic editor for LD or FBD programs, text editor for ST or IL programs. Refer to the corresponding sections in this manual for more information about these tools. The ISaGRAF Code Generator can be directly called from the graphic or text-editing window, to compile the source code of a library function or block.

Dictionary of local variables

A library function or function block can have local variables, and local defined words. To access the variable declaration, the user must run the commands of the "Dictionary" command of the "File" menu, in the editor window, while editing the source code of the function.

⚠ A library function or function block cannot access a global variable or function block instance. Local variables of a function should be initialised in the function body.

Local variables of a function block written in IEC language are copied (instanced) each time the block is used in a project. Local variables of an instance keep their values from one call to the other.

Defining the interface

Functions or function blocks may have up to 32 parameters (input or output). A function always has one (and only one) return parameter, which must have the same name as the function, in order to conform to ST language writing conventions. The list in the upper left side of the window shows the parameters, in the order of the calling model: first the calling parameters, last the return parameters. The lower part of the window shows the detailed description of the parameter currently selected in the list. Any of the ISaGRAF data types may be used for a parameter. The return parameters must be located after calling parameters in the list. Naming parameters must conform to the following rules:

- the length of the name cannot exceed 16 characters
- the first character must be a letter
- the following characters must be letters, digits or underscore character
- naming is case insensitive

The "Insert" command is used to insert a new parameter before the selected parameter. The "Delete" command is used to erase the selected parameter. The "Arrange" command automatically rearranges (sorts) the parameters, so that the return parameters are put at the end of the list.

22.6 "C" Functions and function blocks

The "C" functions and function blocks are computer functions called from the automation application, according to the ST language function calling interface. Functions are synchronous processes. The ISaGRAF target application is suspended during the function execution. Function blocks associate operations and static hidden data. For example, a "counter" function block represents the counting operation, as well as the counting result. Functions and function blocks may be used to complete the standard automation language capabilities, or to access system resources.

The parameters definition box is used to define the name and the type of each calling or return parameter of the function or function block. The "Edit" menu commands are used to define the parameters of the selected function or function block. A function can have up to 31 calling parameters, and always has one return parameter. A function block can have up to 32 parameters, with any mix of call and return parameters. Below is the correspondence between ISaGRAF types and "C" types:

BOOLEAN	unsigned long	unsigned 32 bit word: 1=true / 0=false
ANALOG	long	signed integer 32 bit word
REAL	float	single precision floating value
TIMER	unsigned long	unsigned integer 32 bit word (unit is 1 ms)
MESSAGE	char *	character string.

When a message value is passed onto a "C" function or function block, it cannot contain null characters. The string passed to the "C" code is null-terminated. Refer to the ISaGRAF Target User's Guide for further information on how to manage the "C" source code of a function or a function block, and how to integrate a new element in the ISaGRAF target system.

22.7 Conversion functions

A conversion function is a "C" function called by the ISaGRAF I/O manager each time the analog variables using this conversion are input to or output from the project.

The function creates the relationship between the electrical value of the variable (read on the input sensor or sent to the output device) and its physical value (used in the application expressions). The function is therefore divided into two parts: input conversion and output conversion. The ISaGRAF library manager allows the user to control the "C" source code of a conversion function.

A conversion can be used for an integer or real analog variable. This implies that the conversion function interface is always defined by floating values. The interface is the same for any conversion function. The "C" definition of this interface is made in the "TACN0DEF.H" definition file.

Refer to the ISaGRAF Target User's Guide for further information on how to manage the "C" source code of a conversion function, and how to integrate a new element in the ISaGRAF target system.

23. Using the Archive utility

The ISaGRAF archive utility enables the user to save the ISaGRAF projects and libraries on diskettes or backup directory. The ISaGRAF archive manager is a dialog box that can be called from ISaGRAF Project Management or Library Management windows.

⚠ To create and maintain reliable archives, it is suggested that the following guidelines be used:

- Write the name and description of the saved object on the disk sticker
- Do not save projects and libraries on the same diskette
- Do not save different projects on the same diskette

23.1 Calling the archive manager

The "Archive" dialog box can be called from the "Tools / Archive" menu of the Project Management window, to save or restore either a project, or common data. The "Archive" dialog box can also be run from the "Tools / Archive" command of the ISaGRAF Library Manager, to save or restore elements of the library currently selected in the Library Management window.

Projects

A project is always saved in its entire form. All the components of the project (program source files, object code and application executable code) are saved together in the same archive file. Selection of the "compression" option reduces the size of the project archive.

Library elements

The elements of ISaGRAF libraries can be saved individually. All the components of a library element (technical note, definition, interface, source code...) are saved together in the same archive file.

Common data

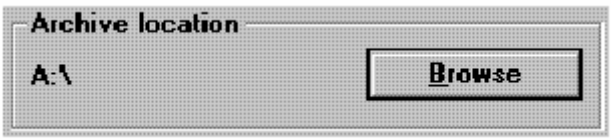
The "Tool / Archive / Common data" command of the Project Management window enables the user to backup or restore the "common range" data existing in the ISaGRAF Workbench. This command does not act on the ISaGRAF libraries. Below is the list of the files that can be copied with this command:

common.eqv ..... common defined words
oem.bat ..... user defined MS-DOS command file

These files are saved one by one on the archive disk, in their original form. The corresponding archive files are never compressed

23.2 Options

The path used for ISaGRAF archives is displayed at the bottom of the dialog box. Press the "Browse" button to browse the disks and select another archive disk and Directory.



When the "Compression" option is set, all the archive files created during a "Backup" procedure are compressed. This option is very useful to reduce the size of a large project archive file, and save it on only one diskette. Archive compression is generally not needed for library components. The ISaGRAF Archive Manager automatically recognises the status of an archive file (compressed or not) when restoring the archive. This implies that the "compression" option has no effect for a "Restore" procedure.

23.3 Backup and restore

The "Workbench" list (on the left) shows the objects existing in the ISaGRAF Workbench installed on the hard disk. The "Archive" list (on the right) shows the objects saved on the specified archive disk and directory.

- ***Backup***

  Saving an object on archive is achieved by selecting the object in the list on the left (objects of the ISaGRAF workbench) and pressing the "Backup" button. More than one object on the list can be selected. The "Backup" button is disabled when an element is selected from the list on the right (restore mode).

- ***Restore***

  Copying an object from the archive to the ISaGRAF Workbench is achieved by selecting the object in the list on the right (archive objects) and pressing the "Restore" button. More than one object on the list can be selected. The "Restore" button is disabled when an element is selected from the list on the left (backup mode).

23.4 Archive files

The ISaGRAF archive manager creates a unique archive file for each saved object. The archive file has the same name as the object. Its file suffix indicates its type. Below are the used suffixes:

.pia..... project

.bia..... I/O board
.lia..... function in IEC language
.aia..... function block in IEC language
.uia..... C function
.fia..... C function block
.cia..... C conversion function
.ria ..... I/O configuration
.xia..... I/O equipment

24. Printing a complete document

The ISaGRAF Document Generator allows the user to build and print a complete document for the selected project. It can be called by the "Project / Print" commands of the Project Management or the Program windows to print a complete document. The Document Generator is also run by the "Print" command of all other ISaGRAF editors to print the contents of a single ISaGRAF document. However, the Document Generator provides the same features in both cases.

The commands of the "Edit" menu are used to define the elements of the project that must be inserted in the document. Doing this the user builds the "table of contents" for the desired document. Any information about the project (programs, variables, options, I/O connection...) may be inserted in the project document. No information from another project or from ISaGRAF libraries may appear in this document.

The "File / Print" command generates the document and send it to the printer, according to the specified table of contents. The "Print" job may take few minutes to build and format the document. It is highly recommended to wait until "Printing Job"is done in the ISaGRAF Document Generator window, before running other commands of the ISaGRAF Workbench. Building the whole document may requir a large space on the hard disk. An error message will be displayed if the disk is full. In such a case, the user will have to either free up disk space by removing files, or reduce the size of the print job. When the "Print" command is run, a dialog box appears. It allows the user to enter a note describing the actual print command. Those notes are stored in a history file, and will be printed on the first page of any future document (including the present one).

24.1 Customising the table of contents

The "Edit" menu contains the commands to define the "Table of Contents" of the document. A choice of commands allow the user to use a default table (with all the components of the project), build a specific table (with only some components) or move items in the table and modify it.

The default list

The "Default list" command of the "Edit" menu defines a standard table of contents for the document, which includes all the components of the project. The standard table consists of:

- Project descriptor
- Hierarchy tree (links between programs)
- Source code for any program
- Diary file for any program
- Common definitions
- Global definitions
- Local definitions for any program
- Global variables
- Local variables for any program

- Application options
- I/O Connection
- Lists of variables
- Conversion tables
- Condensed cross references
- Detailed cross references
- Declaration summary
- Network addresses map
- History of modifications

The table of contents can be saved on disk using the "File / Save" command. This command is greyed when document generator is run from an ISaGRAF editor to print a single document.

Cut and paste

Use "Edit / Cut" and "Edit / Paste" commands to move items in the list, in order to customise the order of the table. The Document Generator allows multiple selection so that a group of items may be cut and pasted.

Clearing the table

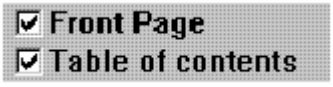
Use "Edit / Clear" command reset the table of contents, so that it can be totally rebuilt using single item insertion.

Inserting items in the table

When the "Edit / Insert" command is run, the "Add item" dialog box appears. It allows the user to insert items (components of the project) into the table of contents. For an item relative to a program, use the "Program" combo box to select a program name. Press the "Add" button to insert the selected item to the table of contents. The same item can appear only once in the table.

24.2 Options

The commands of the "Options" menu are used to define and customise the format of the generated document. Other options are directly available from buttons of the Document Generator window:



When the "Font page" option is set, a header page is printed at the beginning of the document, containing the project title and the history of printouts. When this option is not set, the first item to be printed starts on the first page.

When the "Table of contents" option is set, a table of contents is printed at the end of the generated document. Both options are initially unchecked when the Document Generator is run from a "Print" command of an ISaGRAF editor (program, dictionary...)

SFC charts

The "Separate SFC levels" option directs the system to print, for each SFC program, first the level 1 of the SFC (chart and comments), and then the level 2 programming. When this option is not checked, levels 1 and 2 appear together on the same printout.

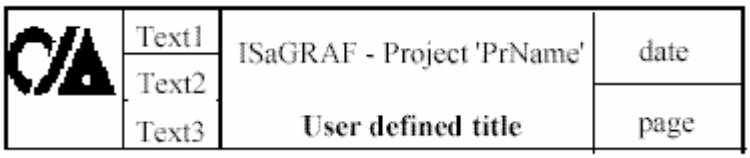
Page format

The "Page format" command of the "Options" menu is used to define the main parameters operated by the Document Generator when formatting a page. The following parameters can be specified:

- **Left margin**: (1 or 2 centimeters, or no margin)
- **Page border**: When this option is selected, a border is drawn around any printed page.

Page title template

The "Page Title" command of the "Options" menu is used to define the contents of the title box printed at the bottom of any page. The standard layout of this box is as follows:



The first line of the main title (with the name of the ISaGRAF project), the current date and the page number are automatically generated by the Document Manager, and cannot be changed.

The three lines of text on the left side of the box (text1, text2, text3) and the second line of the main title are user-defined. The user also can change the logo printed in the box on the left. To use another logo, the user has to specify the pathname of a bitmap image file (.BMP). The image can have any dimension. It will be stretched or shrunk, according to the exact dimensions of the printed page. Clicking on the logo area, in the dialog box, shows the new specified image. The image file must be on the disk (at the specified directory and with the specified filename) when the "Print" command is run.

Selecting character fonts

The "Text font" and "Title font" commands of the "Options" menu are used to define the fonts of characters used when printing text, and titles for any item of the document. The size and style of characters may also be selected for text and titles. The selection of a font is made with the standard dialog box defined by Windows.

Any text (literal programs, names within diagrams...) will be printed with the selected size, style and font of characters. Only titles will be printed with the font selected for titles.

If the fonts of characters are not defined, the standard font of the printer will be used for any text, with the following styles:

- "Normal" style for texts and names within diagrams
- "Bold" style for titles

25. Password protection

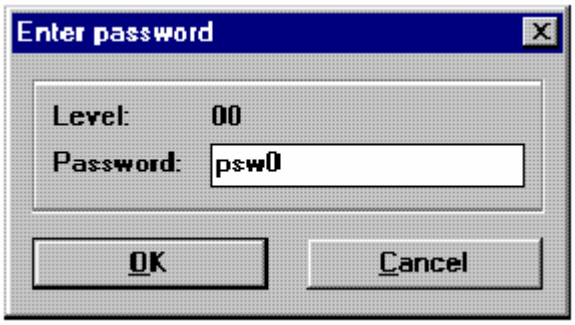
The ISaGRAF Workbench includes a full data protection system, which enables theuser to protect with passwords projects and library elements. A library element can be an I/O configuration, an I/O board or complex equipment, a function or function block written in IEC languages, a "C" function, function block or conversion function. A password protection database is dedicated to one project or library element, and cannot be shared between several ones.

Protection levels

Within one project or library element, the user can define up to 16 access levels, corresponding to different passwords. Access levels are sorted in a hierarchy system. They are numbered from 0 to 15. The higher access level is numbered 0. When a user knows a password, he can access all the items protected by the corresponding access level, plus all the ones protected with lower levels. Each elementary command or data of a project or library element can be separately protected with an access level. For example, the "Make application code" command from the ISaGRAF menus can be protected separately. Elementary data can be a program, a list of options, the technical note of a library element, etc...

Defining password protection

The "Set password" command of the ISaGRAF menus is used to define the passwords and access levels for one project or one library element. This command is called from the menus of the ISaGRAF Project Manager (for a project), or the ISaGRAF Library Manager (for a library element). No password is required whe first running this command. If passwords are already defined, the user must enter the highest level password he knows, before accessing this command. Upper level passwords and protected items then cannot be modified. The "Set password" command enables the user to define the passwords corresponding to the different access levels, and to protect elementary commands or data with the defined levels. Passwords (corresponding to protection levels) are entered by double clicking on a line of the upper list. The following box is used to enter a password.



The list in the lower area shows the different items (data or functions) which can be protected, and current protection level attached to either "read access" or "full access" permissions. Assigning a protection level to "read" permission enables you to prevent users without sufficient permission even to open or print a document. Double click on a line in the lower list to set permissions for the selected item or data. The following box is open:

Both permissions can be set either to "free access", or to a protection level defined by a password. Full access permission cannot be attached to a level with less priority than the one selected for read access.

Note that for some documents, naturally visible when using ISaGRAF Workbench, such as project descriptor, read access cannot be protected with a password.

Accessing protected data

No password or user's name is asked when the Workbench is started. Each time a user wants to have access to a protected data or function, he must enter the required password in a dialog box.

If the user enters the required password (or a password attached to a higher access level), he can continue normally. Each time a password is entered by the user, it is stored in memory, so the user will not have to enter it again later. Stored passwords are held each time an ISaGRAF tool is run from another ISaGRAF tool (for example, the Project Manager runs the Program Manager). Stored passwords are lost when the last remaining ISaGRAF window is closed. Passwords entered during project editing, or by using the Library Manager, or by using the Archive manager cannot be shared. If the user enters a bad password, he cannot run the selected function.

Links with the archive manager

When saving an object (project or library element) on archive disk, the data protection item named "Backup on archive" is invoked. This corresponds to the data protection system attached to the object in the Workbench (hard disk). No test is performed on the data protection system of the object on the archive disk if it already exists. The "Backup" command of the ISaGRAF Archive Manager saves the data protection information with the object on the archive disk.

When restoring an object which already exists in the Workbench (hard disk), the data protection item named "Overwrite with archive" is invoked. This corresponds to the data protection system attached to the object in the Workbench (hard disk). No test is performed on the data protection system of the object on the archive disk. If this command is validated, the restored data protection information will then replace the existing one on the hard disk.

Setting individual protection for variables and I/O channels

The ISaGRAF workbench provides a complete data protection system based on hierarchised passwords. Variable declaration and I/O connection can be globally protected by a password. Additionally, ISaGRAF enables you to set individual protection to any variable or I/O channel. This assumes that:

- passwords are already defined in the password definition system (use the "Project / Set password" command of the Project Management window) so that protection levels are available for individual protection.
- you use protection levels with higher priority for individual protection compared to global variable or I/O protection.

When a variable or an I/O channel has individual protection, a small icon is draw close to its name in dictionary or I/O connection window. Use the "Set protection" and "Remove protection" commands of the "Edit" menu in dictionary or I/O connection windows to set or remove an individual protection for selected variable or channel. Both commands ask you to enter a valid password so that a protection level can be attached to the variable or channel. Then, each time you want to change a variable or a connection to a channel having individual protection you must enter a password with sufficient priority level.

Warning: if a variable or channel is protected with a level, and the corresponding password is removed from protection system, and if no higher level password is defined, variable or channel cannot be changed anymore unless a new password with sufficient level is defined.

26. Advanced programming techniques

This chapter contains more information about the ISaGRAF Workbench and target system. The user is advised to be familiar with the ISaGRAF tools and methods, before reading this section.

26.1 More about ISaGRAF tool

When using the ISaGRAF editing tools, the user can press the right mouse button to open a popup menu, which contains the main editing commands. The menu is opened at the current position of the cursor. This is very useful to reduce mouse operations during cut and paste commands.

The ISaGRAF tools support multiple execution. Although same tool cannot be opened twice to edit the same document, it is possible to open different windows with the same tool and edit different objects as parallel operations.

Other commands are available to find information about graphic buttons in toolbars. Double click an empty area of a toolbar to display the contents of the toolbar as a popup menu. Stay with the mouse cursor on a graphic button displays the corresponding text command.

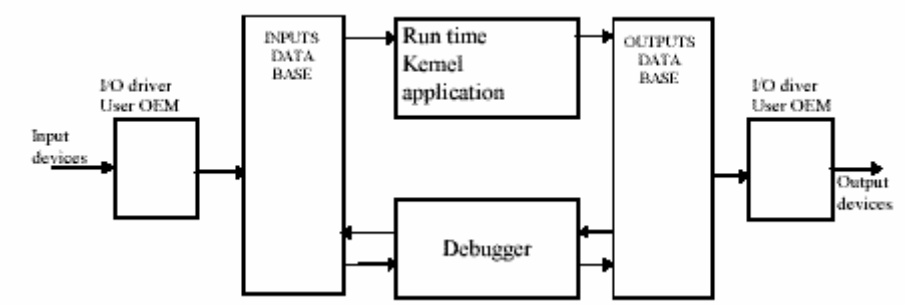
26.2 Locked I/Os and virtual I/Os

Defining an I/O board as virtual disconnects the processing of the physical I/O channels. When a board is defined as virtual, the IsaGRAF kernel operations are not changed. The only difference is that input sensors are not read and output devices are not updated. In this mode, it is possible to use the ISaGRAF debugger to modify the input values. The Virtual attribute applies to a complete board. It is programmed during the I/O board definition, before the application code generation. The virtual attribute is a static feature, and is stored when the application is stopped and restarted.

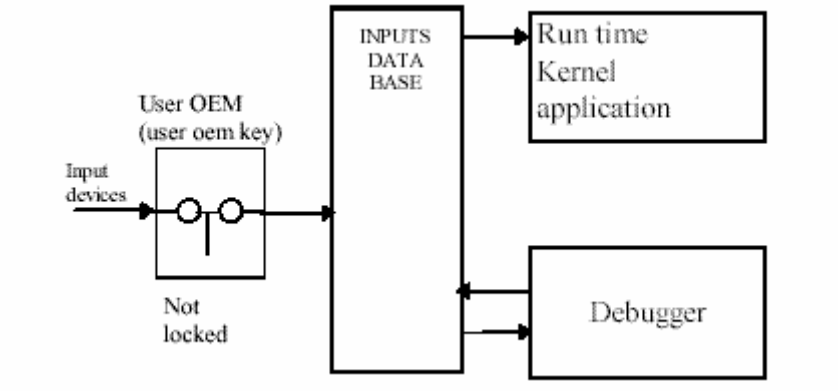
Another possibility is the I/O variable locking. It consists of disconnecting one physical device and the corresponding ISaGRAF I/O variable. Variable locking and unlocking is performed through the debugger. Variable locking is a dynamic operation, and is not memorised when the application restarts. The lock operation applies to only one variable (one I/O channel) at a time. This is the summary of main I/O controlling features:

	Virtual Attribute	*Lock command*
selection tool	I/O board connection	debugger
definition	static	dynamic
selection mode	board	variable
application	validation and tests	maintenance

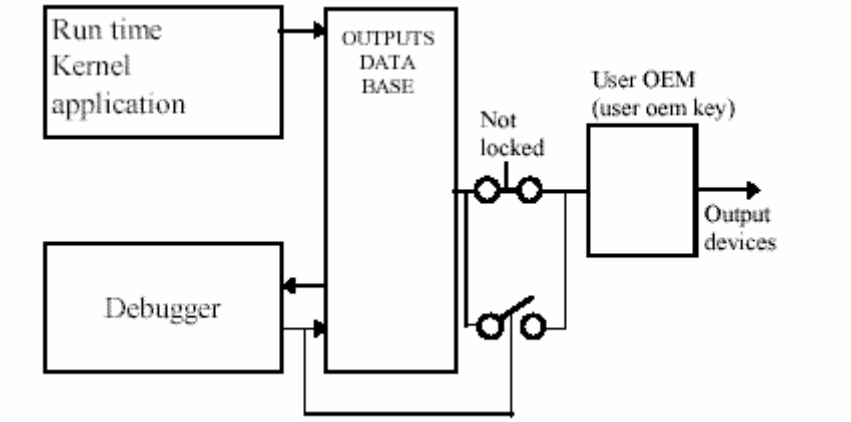
The following chart explains the I/O data flow between the ISaGRAF tasks:



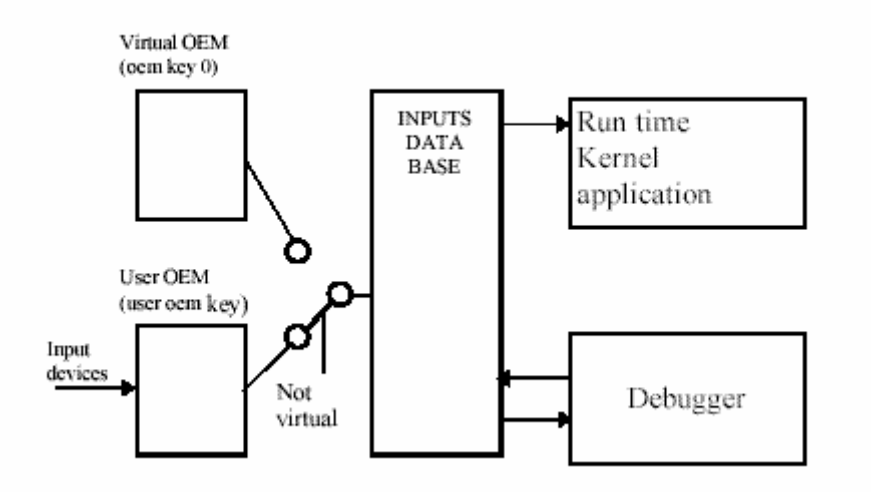
When an input variable is locked, the various accesses to the database are not changed, but the input device is disconnected. Input values can be set with the debugger and processed by the ISaGRAF kernel:



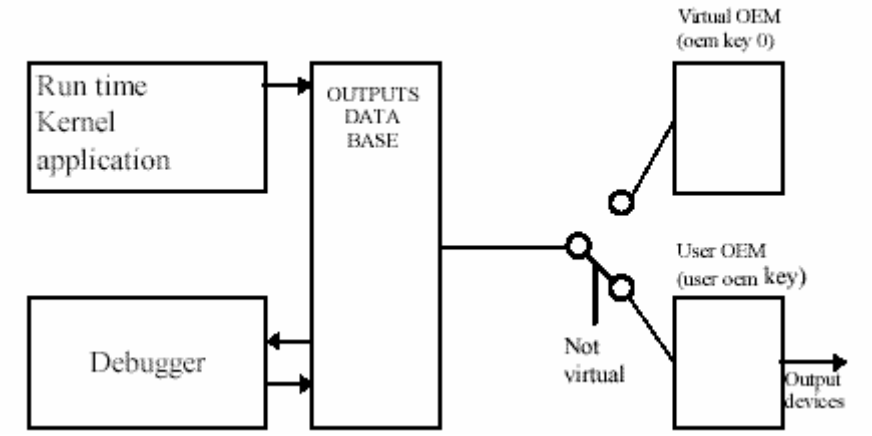
When an output variable is locked, the run-time kernel and the output driver are disconnected. In this case, access is still possible to the output device, via the output driver, with the ISaGRAF debugger:



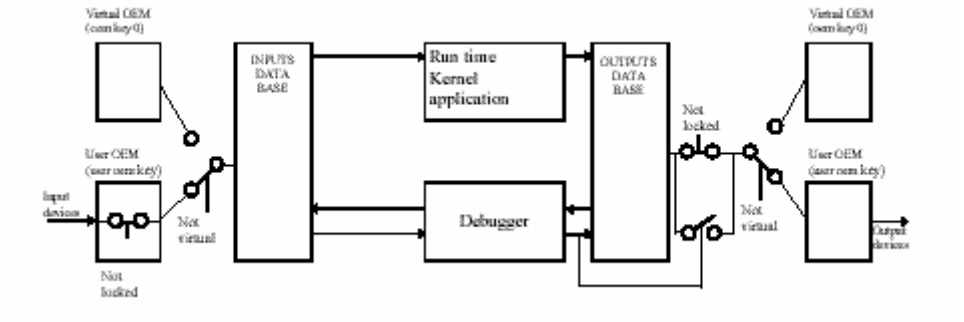
When setting the virtual attribute for an input the input database and the associated input devices are disconnected. A virtual I/O driver replaces the real one.



Setting the virtual attribute follows the same rules for an input board or an output board. For output boards, the ISaGRAF kernel updates the output database. This database and the associated output devices are, however, disconnected. A virtual I/O driver replaces the real one.



To summarise all possibilities:



26.3 PC-PLC link validation

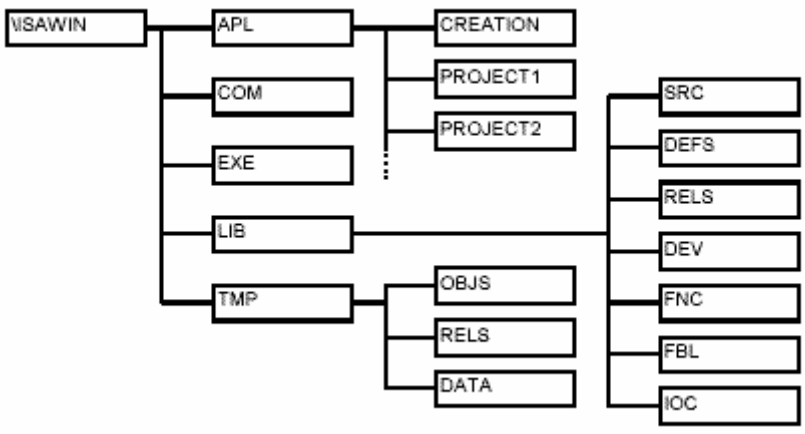
Most of the problems related to poor communication between the ISaGRAF workbench and the target PLC are represented in the debugger window by the "disconnected" status message. Before any diagnostic tests are performed, the communication should be validated when no application is active in the target PLC. This way the serial communication link can be validated on its own, isolating it from execution related effects.

The "C" language, used for description of the conversion functions and C functions, allows direct access to the target system. A programming error in such a software component may generate system errors or incorrect ISaGRAF system behaviour. Such problems may occur when I/O drivers are developed with the ISaGRAF I/O toolkit. System errors, for example, may be caused if an I/O board is connected on an invalid bus address. The following table gives a synthetic summary of error diagnostics:

status	context	Diagnostic
"disconnected" (before download)		- target is not running - no cable / invalid cable - invalid link parameters - ISaGRAF target badly installed
"disconnected" (after download)	cycle to cycle starting mode	- invalid I/O configuration - system crash
	real time starting mode	- invalid I/O configuration - system crash (due of "C" programming)
"no application"		- application not downloaded - application not started (due of "C" programming) - Intel/Motorola mismatch - Invalid target version

26.4 ISaGRAF directories

The ISaGRAF Workbench works on a dedicated disk directory structure. The root directory of this architecture is specified by the user during the installation of ISaGRAF. The default name for the ISaGRAF root directory is ISAWIN. This is the standard disk architecture created by the installation program:



These are the standard ISaGRAF sub-directories

DIRECTORY CONTENTS

APL	root directory for the ISaGRAF projects each project corresponds to one sub directory which contains all the data of the project other directories may exist for other project groups. ISaGRAF installation program creates "**SMP**" directory where are stored samples applications.
COM	"common" range data Data can be used by any project
EXE	ISaGRAF programs and help files
LIB	ISaGRAF libraries: - lists of elements - parameters or interface for each element - technical notes
LIB\IOC	source code for I/O configurations
LIB\FNC	source code of functions written in IEC languages
LIB\FBL	source code of function blocks written in IEC languages
LIB\SRC	source code for conversions and C functions
LIB\DEFS	source header for conversions and C functions
LIB\RELS	Conversions and C functions object code
LIB\DEV	command files for developing "C" libraries makefiles, link lists, etc...
TMP	Temporary files: sub-directories of TMP are reserved for the ISaGRAF Code Generator and cannot be deleted.

The sub-directories can be moved to other disk locations. When the user has a nonstandard architecture, the pathnames of the sub-directories should be declared in the WS001 section, in the ISA.ini initialisation file, in the EXE sub-directory of ISaGRAF. Here are the entries of the WS001 section:

Isa	root directory for ISaGRAF architecture
IsaExe	root directory for ISaGRAF programs and help files

IsaApl	root directory for ISaGRAF projects
IsaTmp	directory for temporary files
IsaSrc	directory for library source code
IsaDefs	directory for library source headers

Note that if you change the IsaTmp entry to another directory, you must create the sub-directories OBJS, RELS and DATA in the new directory. The following example uses the entries of the WS001 section to redefine the standard ISaGRAF disk architecture:

```
;file 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
```

When you want to add "C" functions or function blocks to the ISaGRAF target, the \ISAWIN\LIB\DEV directory is used to store development files: command files, makefiles, maps, etc... The \ISAWIN\LIB\RELS directory is used to store the object files generated during "C" compiling, and the ISaGRAF "C" libraries required for LINK operations.

26.5 Application symbols

Each object of an ISaGRAF application is referenced by a name (entered during variable declaration) and an internal virtual address, calculated by the code generator. The virtual address of a variable is not its network address entered during the declaration of the variable. Virtual addresses are used for communication work, and special "C" applications using the OEM option. When the ISaGRAF code generator is run, it makes an ASCII file with the logical correspondence between names and virtual addresses for all the objects (variable, programs, steps...) of the project. This file can be easily interrogated for information about the ISaGRAF static database from any user's application. The file is named "APPLI.TST" and is located in the directory of the ISaGRAF project: "\ISAWIN\APL\proname" (proname is the name of the project). This section describes the detailed format of the "APPLI.TST" file. The main notations used for the following descriptions, is shown below:

```
VA          virtual address
ATTR        attribute of a variable
USP         "C" function
```

Possible values for the attributes of a variable are shown below. Such values occur in the "attributes" fields:

```
+X          internal variable
+C          read-only internal variable
+I          input variable

+O          output variable
```

All the numbers, except virtual addresses, are expressed as decimal integers. The virtual addresses (VA) are expressed as hexadecimal 4 digit numbers, and are preceded by the character "!". For example:

```
123         this is a decimal number
!A003       this is an hexadecimal virtual address
```

The main structure of the file "APPLI.TST" is shown below. The file is structured as a list of blocks. A block is a list of records. Each record is described on one line of text. Each block begins with a header, put on one line of text.

```
Start block
description blocks
end block
```

The general syntax of one block is shown below:

```
@ <block_name> <arguments>
#record...
#record...
...
```

The structure of the first block, containing the main Information about the application, is shown below:

```
@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 ..... application symbols checksum
appli_name ..... name of the application
version ..... ISaGRAF workbench version number
creation_date.... application generation date
nbprg..... number of programs
nbstep ..... number of SFC steps
nbtra..... number of SFC transitions
nbpro..... number of "C" functions used
nbvar ..... total number of variables

The structure of the last block, which signals the end of the file, is shown below:

```
@END_SYMBOLS
```

The structure of the block used to describe the programs of the application, is shown below:

```
@PROGRAMS,<nbprg>
#<va>,<name>

#...
```

nbprg..... number of programs defined in this block
va..... virtual address of the program
name..... program name

The structure of the block used to describe the SFC steps of the application is shown below. Note that there is one virtual step defined for each non-SFC program:

```
@STEPS,<nbsteps>
#<va>,<name>,<father>
#...
```

nbsteps ..... number of steps defined in this block
va..... virtual address of the step
name..... step name
father ..... virtual address of the father

The structure of the block used to describe the SFC transitions of the application, is shown below:

```
@TRANSITIONS,<nbtrans>
#<va>,<name>,<father>
#...
```

nbtrans..... number of transitions defined in this block
va..... virtual address of the transition
name..... transition name
father..... virtual address of the father

The structure of the block used to describe the boolean variables of the application, is shown below:

```
@BOOLEANS,<nb_boo>
#<va>,<name>,<attr>,<program>,<eq_false>,<eq_true>
#...
```

and if variable number exceeds 4095:

```
X#(1.<varno>),<name>,<attr>,<program>,<eq_false>,<eq_true>
```

nb_boo ..... number of variables in this block
va ..... virtual address of the variable
varno ..... range of the address (within boolean data type)
name ..... name of the variable
attr ..... attribute of the variable
program ..... virtual address of the parent program
..... or "**!0000**" for a global variable
eq_false ..... string used for false value
eq_true ..... string used for true value

The structure of the block used to describe the analog variables of the application, is shown below:

```
@ANALOGS,<nb_ana>
#<va>,<name>,<attr>,<program>,<format>,<unit>
#...
```

and if variable number exceeds 4095:

```
X#(2.<varno>),<name>,<attr>,<program>,<format>,<unit>
```

nb_ana ..... number of variables in this block
va ..... virtual address of the variable
varno ..... range of the address (within analog data type)
name ..... name of the variable
attr ..... attribute of the variable
program ..... virtual address of the parent program
..... or "**!0000**" for a global variable
format ..... = "**I**" for an integer variable
..... = "**F**" for a real variable
unit ..... unit string

The structure of the block used to describe the timer variables of the application, is shown below:

```
@TIMERS,<nb_tmr>
#<va>,<name>,<attr>,<program>
#...
```

and if variable number exceeds 4095:

```
X#(3.<varno>),<name>,<attr>,<program>
```

nb_tmr ..... number of variables in this block
va ..... virtual address of the variable
varno ..... range of the address (within timer data type)
name ..... name of the variable
attr ..... attribute of the variable (always +X: internal)
program ..... virtual address of the parent program
..... or "**!0000**" for a global variable

The structure of the block used to describe the message variables of the application, is shown below:

```
@MESSAGES,<nb_msg>
#<va>,<name>,<attr>,<program>,<max_len>
#...
```

and if variable number exceeds 4095:

```
X#(4.<varno>),<name>,<attr>,<program>,<max_len>
```

nb_msg ..... number of variables in this block
va..... virtual address of the variable
varno ..... range of the address (within message data type)
name..... name of the variable
attr ..... attribute of the variable
program ..... virtual address of the parent program
..... or "**!0000**" for a global variable
max_len..... maximum length (declared capacity)

The structure of the block used to describe the "C" functions used in the application, is shown below:

```
@USP,<nb_usp>
#<va>,<name>
#...
```

nb_usp ..... number of C functions in this block
va..... virtual address of the C function
name..... name of the C function

The structure of the block used to describe the "C" function block instances used in the application, is shown below:

```
@FBINSTANCES,<nb_fb>
#<va>,<inst_name>,<fb_name>
#...
```

nb_fb ..... number of instances of a C function blocks in this block
va..... virtual address of the C function block instance
inst_name ..... name of the C function block instance
fb_name ..... name of the reference C function block

26.6 Limits of ISaGRAF "LARGE" (WDL) workbench

There are some limitations for the objects used in the ISaGRAF Workbench. Of course, many other practical limits are due to the configuration of the computer used (available memory and disk space), and the capabilities of the ISaGRAF target system (available memory, available hardware and software resources...). The following numbers absolute limits that cannot be exceeded.

- ***For a project:***

Object	*Maximum*	*Notes*
Programs	255	grouping main, sub and child programs
Levels in the hierarchy	20	

The number of projects installed on the Workbench is only limited by the available space on the hard disk.

- ***For names:***

Name for:	*Maximum*	*Notes*
Project	8 char	
Program	8 char	
Variable	16 char	+ 60 characters for comment
Defined word label	16 char	
Defined equivalence	255 char	+ 60 characters for comment
Conversion table	16 char	
List of variables	16 char	
function / f.block (lib)	8 char	this applies to C functions, C function blocks or functions written in IEC languages
function parameter (lib)	16 char	this applies to C functions, C function blocks or functions written in IEC languages
IO board	8 char	
IO configuration	8 char	
Board oem parameter	16 char	
Conversion function	8 char	

Editing (for one program):

Object	*Maximum*	*Notes*
SFC rows	600	
SFC columns	20	
SFC steps	4095	for the whole project, grouping steps, initial steps, beginning and ending steps
SFC transitions	4095	for the whole application
LD/FBD editing	200 cols	
	2000 rows	this is the size of the editing area in cell units.
Quick LD editing	no limit	limits are imposed by the PC capacity
IL labels	251	in the same IL program
Text editing	40KBytes	or less according to the system configuration

For the dictionary (for one project):

Object	*Maximum*	*Notes*
Boolean variables	65535	
Analog variables	65535	grouping integer and real variables
Timers	65535	
Message variables	65535	
Defined words	4095	in the same list (same range)
Defined words	255	used in the same program
Conversion tables	127	used in the application
Points in one table	32	defined in the same conversion table

The limits given for maximum number of boolean, analog or message variables group internal, input and output variables. It also includes all hidden temporary or

variables allocated by compilers. The number of variables edited together (same type, same scope), in the dictionary editor cannot exceed 16000. Depending on PC configuration, the limit can be less than 16000. The application cannot run on an ISaGRAF target version V3.21 or earlier if the total number of variable for one type exceeds 4095. The standard "Modbus" link using network addresses does not cannot be used if number of variables for one type exceeds 4095.

IO connections:

Object	*Maximum*	*Notes*
IO Boards	256	defined for the same application (boards or complex equipments)

Number of I/O boards including single boards and items of complex equipments cannot exceed 256.

IO channels	128	on the same board

For libraries:

Object	*Maximum*	*Notes*
Functions (IEC lang.)	255	installed together in the library
Function blocks (IEC lang.)	255	installed together in the library
C functions	255	installed together in the library
C function blocks	255	installed together in the library
function blocks instances	4095	for the same type of function block in the same application
Function input parameters	31	this applies to C functions and functions written in IEC languages
Function block parameters	32	freely distributed between input and output parameters. At least 1 output parameter is required.
Conversion function	128	installed together in the library
IO configurations	255	installed together in the library
IO boards	255	installed together in the library
Complex IO equipt.	255	installed together in the library
Board oem parameters	16	