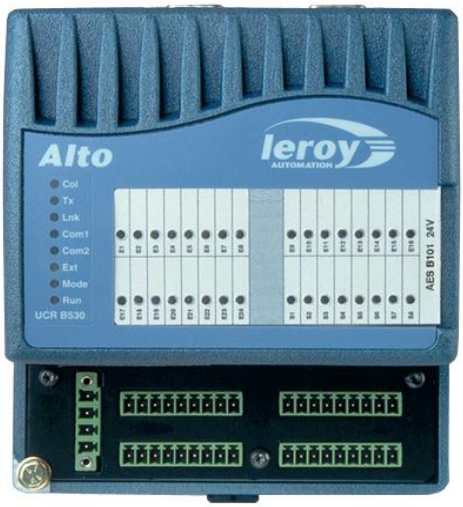
DEVELOPMENT TOOLS
FOR ALTO PLC
ISaGRAF WORKBENCH

Alto ISaGRAF

USER'S MANUAL



P DOC ALTO 300E-V1.7

SCOPE OF SUPPLY

Alto **ISaGRAF Kit** includes:

- 1 RS232 cable from the PC to Alto.
- 1 "Alto ISaGRAF Libraries" CD Rom.

Alto implementation manual is supplied with Alto.

This documentation describes the functions of:

- **embedded software (kernel):**
  - version : 4.55
- **Alto ISaGRAF libraries: version 3.0**

TECHNICAL SUPPORT:

Phone: (33).(0)5.62.24.05.46
Fax: (33).(0)5.62.24.05.55
e-mail: support@leroy-autom.com

ISaGRAF is a registered trademark of ICS Triplex.
MS-DOS and Windows are registered trademarks of Microsoft Corporation.

All other brand or product names mentioned herein are registered trademarks of their respective owners.

LEROY Automatique Industrielle is constantly developing and improving its products. The information contained herein is subject to change without notice and is in no way legally binding upon the company. This manual may not be duplicated in any form without the prior consent of LEROY Automatique Industrielle.

Leroy Automatique Industrielle

Head office : Boulevard du Libre échange
31650 Saint Orens
Phone: (33).(0)5.62.24.05.50
Fax: (33).(0)5.62.24.05.55

Web site : http://www.leroy-automation.com

CONTENTS

1. General Overview

Alto ISaGRAF made by LEROY Automation is programmed using the ISaGRAF workbench.

This documentation describes how to start up the ISaGRAF workbench on Alto target. Details concerning the installation and use of the ISaGRAF workbench (creating a project, programming, etc.) can be found in the ISaGRAF User's Guide supplied with the workbench.

The following diagram provides an overall view of ISaGRAF architecture on Alto:

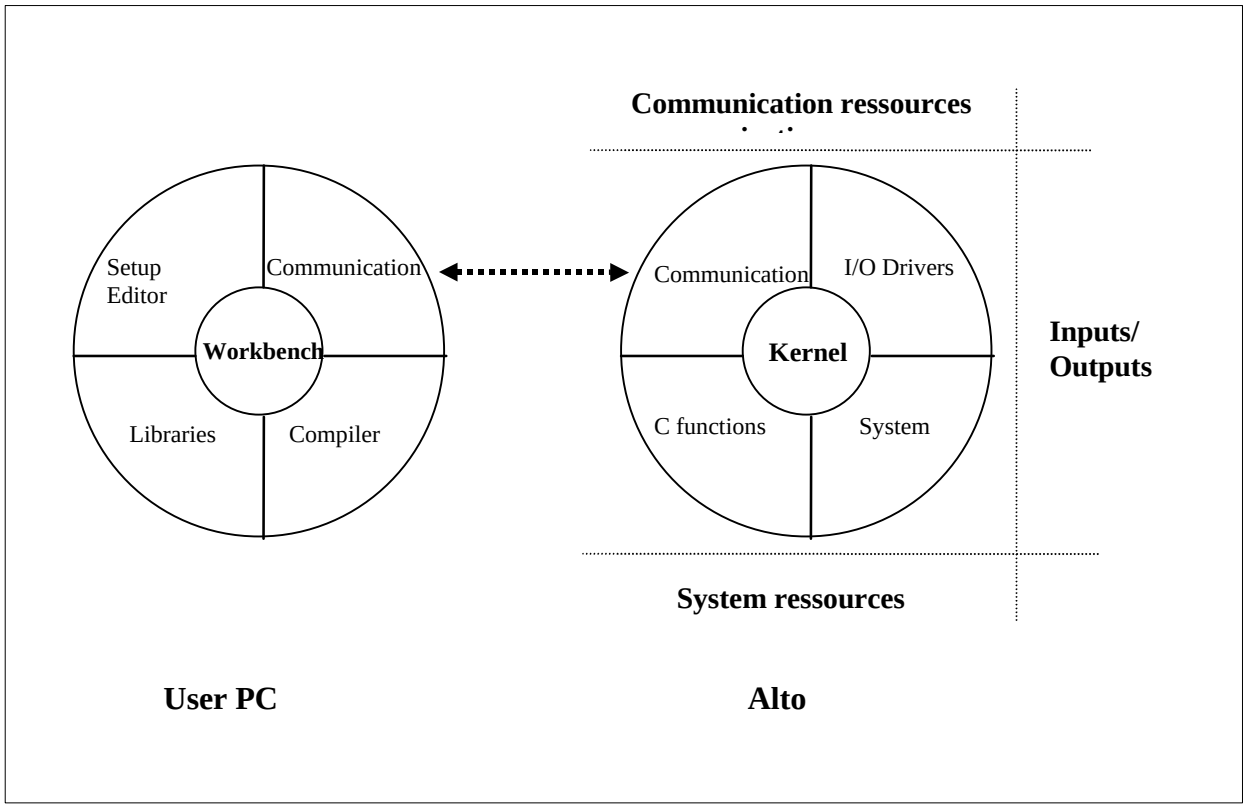


Figure 1: ISaGRAF Architecture on Alto

Alto target supports the **ISaGRAF kernel** and provides access to the following resources:

- Alto system (clock, memory, etc.),
- remote input/output drivers,
- communication functions.

The ISaGRAF workbench is designed to create and modify new projects intended for the target. The "**Libraries**" utility allows the user to manage Alto. This includes the following functions:

- adding local input/output boards,
- adding remote communication drivers (Modbus,Jbus, Slave Modbus),
- input/output drive functions (leds),
- Alto diagnostics functions (board status read, etc.).
- ...

1.1. Hardware Resources of Alto

Alto is a hardware work base. The ISaGRAF kernel is run on this base and uses the available hardware resources. For this reason, even a description of the hardware may help to understand the operating modes of Alto and the related functions.

Processor Intel 386 Ex

Flash memory : 512 Kb : 64ko used for TIC IsaGraf code

RAM memory : 512 Kb

Port Ethernet 10 base T on connector J1

Asynchronous com 232/485 channel 0 (com1) on connector J2

Asynchronous com 485 channel 1 (com2) on connector J3

Backed-up clock

FRAM backed-up (saved data)

An infra red CAN bus 500kbit allows to connect to a UC master two extension UC placed of each side of the master. (see UCR documentation)

1.2. Cycle of Processes Performed

Following the well-known operating principle of a PLC, the ISaGRAF kernel runs the following processing cycle:

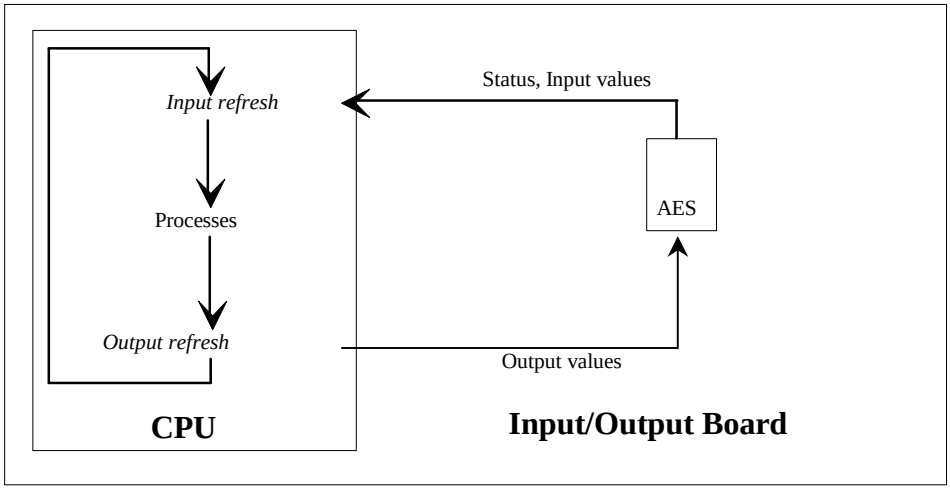


Figure 2: Alto ISaGRAF Processing Cycle

1.3. CPU and AES Alto units

Available CPU unit :

UCR Type	Function	Communication Ports	Plug In cards and compact flash
UCR B521	PALT FCT 301	1 RS232/RS485 Port and 1 RS485 Port	No
UCR B530	PALT FCT 406	1 Ethernet Port, 1 RS232/RS485 Port, 1 RS485 Port and Infrared Port	Yes
UCR B531	PALT FCT 305	1 Ethernet Port, 1 RS232/RS485 Port and 1 RS485 Port	No

Available plug in CPU units :

UCR Type	Function	Communication Ports
UCR B510	PALT FCT 000 X	Infrared

Available I/O units :

AES Type	Digital Inputs	Digital Outputs	Analog Inputs	Analog Outputs	User's manual
Bx01	**24**	**8**			**P ALT DOC 002 E**
Bx02	**4**	**4**	**8 universals**	**2**	**P ALT DOC 005 E**
Bx03	**16**	**8 relays**			**P ALT DOC 007 E**
Bx04	**8**	**8 relays**			**P ALT DOC 007 E**
Bx06	**16**	**8**			**P ALT DOC 002 E**
Bx07	**8**	**8**			**P ALT DOC 002 E**
Bx08	**4**	**4**	**10**	**2**	**P ALT DOC 006 E**
Bx09	**4**	**4**	**8**		**P ALT DOC 006 E**
Bx10	**4**	**4**	**4**		**P ALT DOC 006 E**
Bx11	**4**	**4**	**4**	**2**	**P ALT DOC 006 E**
Bx12	**32**				**P ALT DOC 008 E** **P ALT DOC 010 E**
Bx13	**24 safety**	**4 relays**			**P ALT DOC 004 E**
Bx14	**16 safety**	**4 relays**			**P ALT DOC 004 E**
Bx15	**8 safety**	**4 relays**			**P ALT DOC 004 E**
Bx16	**24 safety**				**P ALT DOC 004 E**

2. Programming Environment

2.1. Installing the IsaGraf Workbench

Refer to the IsaGraf "User's Guide" and follow the instructions starting from the chapter entitled "Startup". The hardware and software configuration required for the IsaGraf workbench is sufficient for operation with Alto target.

Start up IsaGraf workbench.

2.2. Integrating Alto-specific Functions

Leroy Automation has developed **Alto-specific libraries.** These are designed to use resources specific to Alto: inputs/outputs, remote communication protocols, etc.

These libraries are supplied on the "Alto IsaGraf Libraries" floppy disk included in this programming kit. In order to install them on the workbench correctly, select **"Libraries**" in the "**Tools**" menu in the main window (see A22 in the IsaGraf User's Guide). Access to the library is also possible by clicking on the "**Libraries**" icon in the IsaGraf program group.

2.2.1. Source files decompression:

Insert the "Alto IsaGraf Libraries" CD Rom into your CD drive.

Execute the file « LibrariesAltoIsagraf.exe » , choose a new directory : for example, « C:\Isawin\Lai » and start decompression.

The directory you choose contain now the following IsaGraf objects:

Family	Object type	File Extension
Libraries	IO configurations	*.ria,
	IO boards	*.bia
	IO complex boards	*.xia
	Functions	*.iia
	Function blocks	*.aia
	C functions	*.uia
	C functions blocks	*.fia
Projects		*.pia

2.2.2. Integrating Alto specific libraries in IsaGraf workbench

Start up the IsaGraf libraries soft.

In menu "**File**"/"**Other libraries**", select a library type: « **IO Boards** » for example.

In menu "**Tools**"/"**Archive**»: select each element contained in "**Archive**" list and click on button "**Restore**". Don't forget to select the right archive location: directory « C:\Isawin\Lai » choose at decompression step.

Repeat that operation for each different object type.

Once all the objects have been integrated into the workbench, all the C functions specific to Alto may be used as standard ISaGRAF functions. C functions are listed further on in this manual.

The "Libraries" utility supplied with the ISaGRAF workbench and used to generate these functions is designed to display the "**Data sheet**" specific to each function. It contains all the information required to use functions or input/output boards (parameters, return codes, restrictions, examples, etc.).

On-line application modification and source code backup (remote read) functions are not available on Alto.

2.2.3. Integrating Alto projects examples in ISaGRAF workbench

Start up the ISaGRAF project management soft.

Same restore operation than the libraries is to perform for all projects examples : in menu "**Tools**"/"**Archive**"/"**Projects**", select each element contained in "**Archive**" list and click on button "**Restore**".

Projects examples list :

- Alext : I/O extension modules implementation
- Alccf : compact flash implementation
- ALhtr : Backed up RT Clock implementation
- Aljbusm1 : modbus master asynchronous protocol implementation
- Aljbusm2 : modbus master asynchronous protocol implementation with SFC chart
- Aljbuss : modbus slave asynchronous protocol implementation
- AlNulcar : simple communication with null character treatement implementation
- AlNulpro : simple communication : transmission and reception byte protocol implementation
- Alpid : pid implementation
- Alsaveva : Retained variables implementation
- Altosnmp : SNMP protocol : MIB and Traps implementation
- AlTCPe : modbus/TCP slave protocol implementation
- AlTCPm : modbus/TCP master protocol implementation
- Altemail : SMTP protocol implementation : sending electronic mail
- Altxem1 : horodated emails implementation
- Altxem2 : horodated emails implementation
- Alxmult1 : multi communication protocol implementation : modbus/TCP (master and slave), asynchronous modbus (master and slave)
- Alxmult2 : multi communication protocol implementation : modbus/TCP slave, serial link, modbus asynchronous master :
- Alxmult3 : multi communication protocol implementation : modbus/TCP slave, NullTCP, modbus asynchronous master, I/O extensions
- Alxmult4 : multi communication protocol implementation : modbus/TCP slave, NullTCP, asynchronous modbus master, Compact Flash storage
- Alxmult5 : Gateway function : Ethernet network / asynchronous network : modbus/TCP slave protocols and modbus asynchronous master implementation.
- Alxmult6 : multi communication protocol implementation :SNMP : MIB and timestamped TRAPS

2.3. First Program

A minimum program "without treatments " can be made as follows:

- create a new project by selecting: "File", "New Project" in the "Project Manager" window,
- set "B53x" on slot 0 of the input/output wiring : CPU unit,
- set "Bxxx" on slot 1 of the input/output wiring : AES unit,
- In the list of "**Compiler options**", only select "**TIC code for INTEL**" *.
- generate the application,
- download it to Alto.

This minimum program doesn't perform any processing.

Important: the **console link** (or workbench link) is located by default on **J2 connector**. It can be located on the Ethernet link (see "CPU Management").

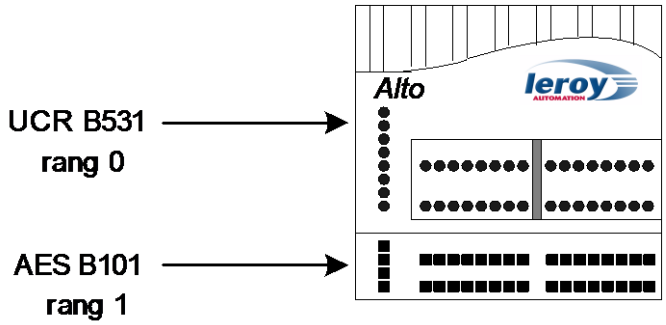
In the following sections of this document, details are given of specific processes that can be programmed on Alto ISaGRAF.

2.4. Hardware Configuration

Alto hardware configuration is achieved by selecting **"IO connection"** in the **"Project"** menu. This window proposes 256 input/output board slots. Only the first 6 can be used to program an Alto.

The first slot is reserved for the CPU: B520, B530 or B531.

Only three types of CPU are used for all Alto devices.



The communication Ports associated with each CPU are declared in the ISaGRAF program using specific functions.

Alto rack is shown vertically and the input/output boards are numbered from 0 to 5. The following is an configuration example showing the corresponding items on Alto:

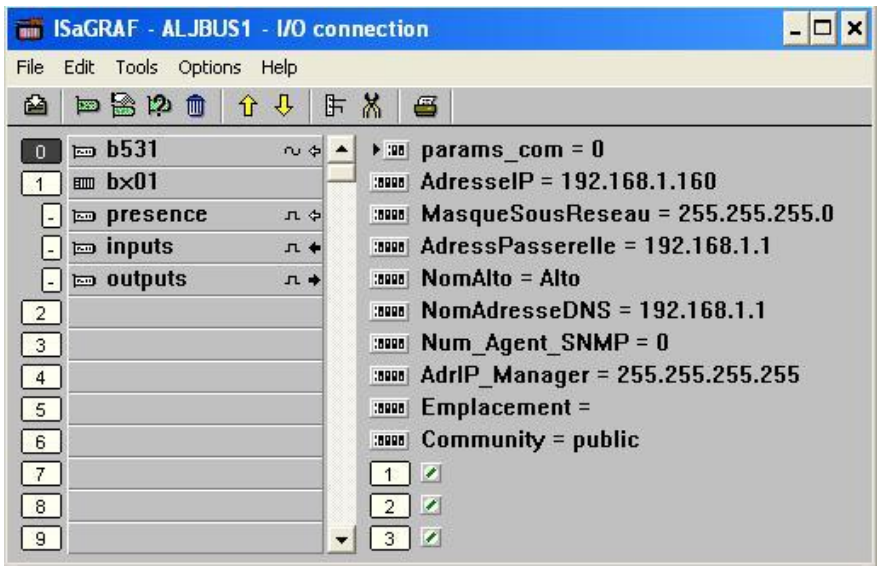


Figure 3: Input/Output Wiring Principle

The logical order of each Alto board is indicated below it: CPU = 0, 1st I/O = 1. The B101 board is in fact composed of several boards. It only occupies one slot in the list of declared boards.

Note: extension blocks are not configured. On the wiring page, an Alto is considered as a single rack.

2.5. Make application

- In menu "**Make**"/"**Compiler option**" , select the option : "**ISA86M : TIC code for INTEL**".
- Make the application : menu "**Make**" / "**Make application**"

2.6. Connecting the ISaGRAF Workbench to Alto

When you switch on an Alto ISaGRAF, it runs the following algorithm:

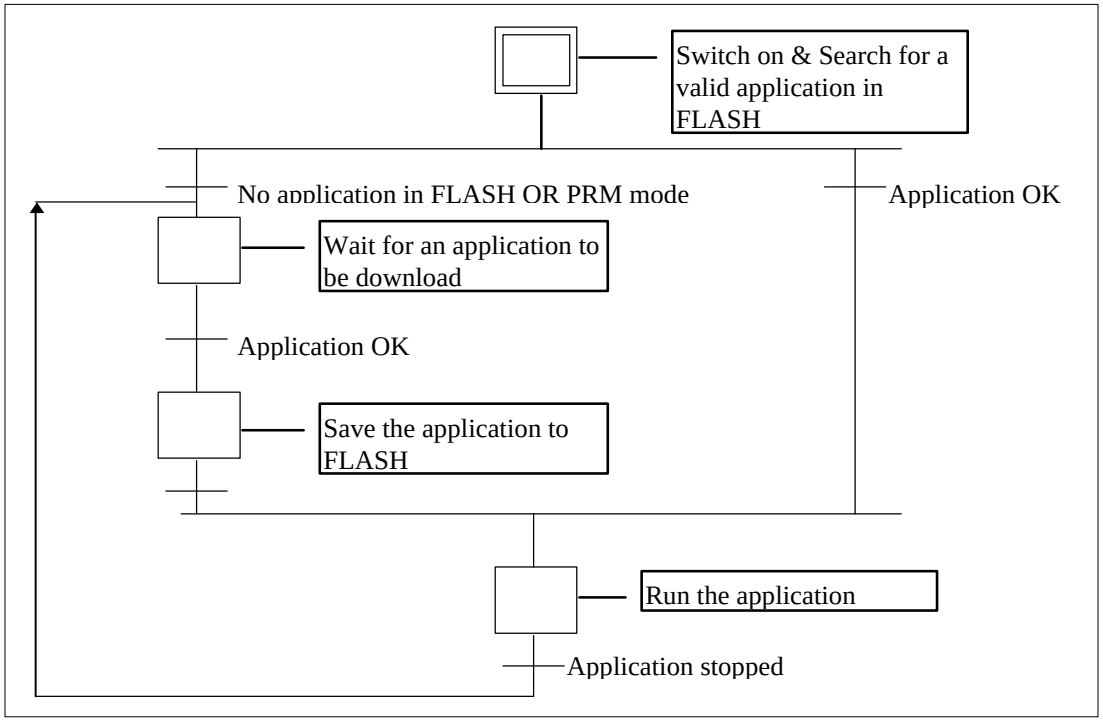


Figure 4: Alto ISaGRAF: Theory of Operation

The "RUN" LED flashes slowly (at 1 sec intervals) while the application is running on Alto.

To connect to Alto, proceed as follows:

- connect the RS232 cable delivered with the kit between a COM on the PC and **J2 connector** on Alto,
- in the "**Debug**" menu, set up the communication parameters as follows: **slave 1, 19200 bps, no parity, 1 stop bit, 8 data bits and no flow control**. Set the time-out to 10 seconds, for example, and the number of tests to 3. Then select the communication Port used on the PC.
- select "**Debug**" in the "**Debug**" menu.

According to the diagram shown above, two cases may arise:

- no application has been retrieved from the FLASH memory: the message in the debugger display window is **"No application**",
- an application has been retrieved from the FLASH memory: the message in the debugger display window is **"name of active application"**. Data relating to the cycle time and status of this application then appears in the main display window of the debugger.

Refer to section A.15 "**Debugging**" of the ISaGRAF User's Guide" for details concerning the correct use of the debugger.

If no connection has been made between Alto and the workbench, the workbench can be forced not to retrieve an application from the FLASH memory: this is Alto **parameter setting mode** (done with SSTB soft).

2.7. Downloading an Application

An application is downloaded from the workbench to Alto by selecting "**Download**" in the "**File**" menu of the debugger.

In order to make the transfer, the **message "No application"** must be displayed in the debugger display window:

- either in PRM mode,
- or by switching off the active application from the debugger : by selecting "**Stop Application**" in the "**File**" menu of the debugger.

Select "**Download**" to start downloading the application. The debugger display window indicates what percentage of the application has been transferred. At the end of the transfer, the application is automatically saved to FLASH by Alto. The application is then run in real-time mode.

If an error occurs during the write to FLASH, it is reported to the workbench as a number ranging from 100 to 255 (see section on Errors).

2.8. Debugging an Application

An application can be debugged in one of two ways:

- either on the PC using the simulator accessed by selecting "**Simulate**" in the " **Debug**" menu,
- or on Alto using the debugger accessed by selecting "**Debug**" in the "**Debug**" menu.

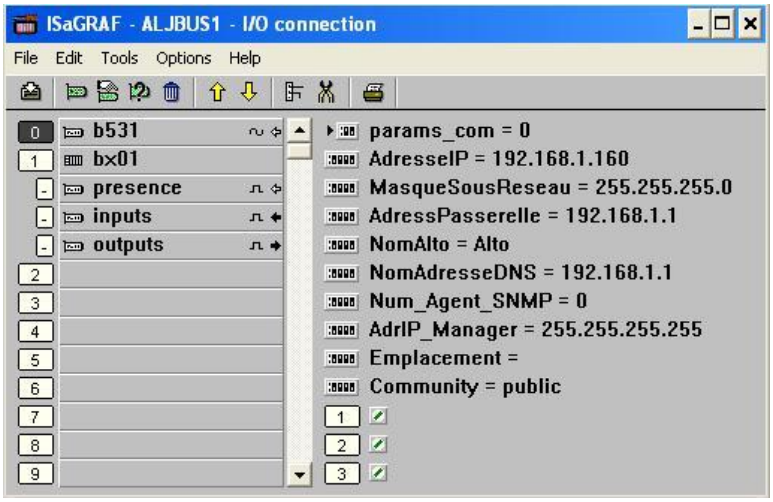
Details on how to use these two debugging modes can be found in the ISaGRAF User's Guide.

The size of the TIC generated by the ISaGRAF workbench corresponds to the size of the **appli.x8m** file. This file is located in the \isawin\apl\"application name"\ directory.
Application size is limited to 64 KB: appli.x8m file < 64KB.

3. Main CPU and Extension CPU

3.1. main CPU configuration

The main CPU must always be located on the **first slot** of the I/O wiring editor.
Example of the B531 configuration :



The output « 1 » indicate the version number of the embedded kernel

The output « 2 » indicate the number of actives tasks

The output « 3 » indicate the RAM memory available in byte

3.1.1. consol link parameter

by default, consol link is on RS232 serial port named J2, and params_com=0 ;

the communication parameters are:

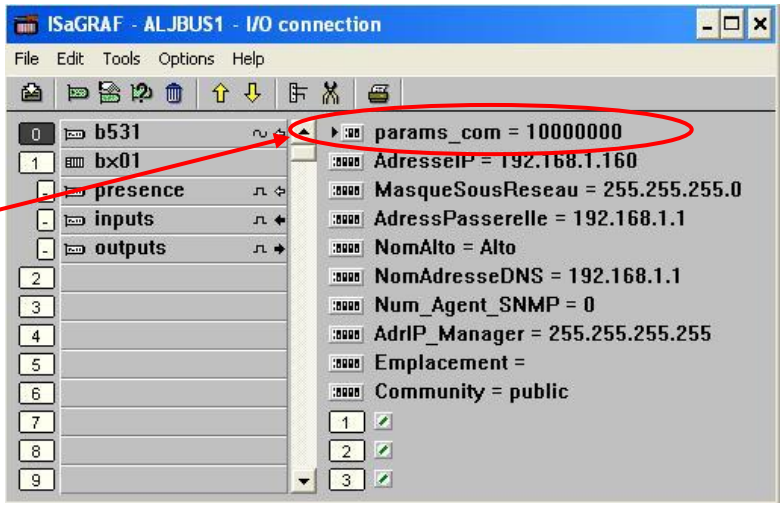
- slave 1
- speed 19200 bauds
- parity none
- stop bit 1
- data 8 bits.

Console Link on the Ethernet port

The Ethernet port can support the console link that enables communication between Alto and the ISaGRAF workbench.

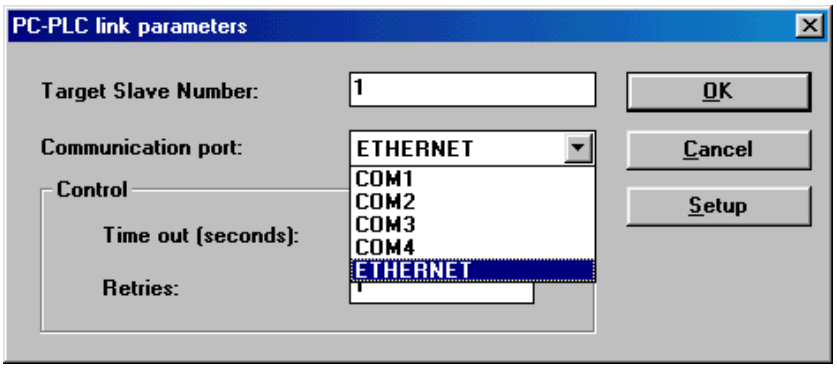
In order to achieve this you just need to modify the params_com in the cabling window as indicated below : params_com=10000000 designates the Ethernet port as the console link .

In this case of an Ethernet port the slave number is replaced by the IP address of Alto and the communication format is imposed by the IEEE 802.3 (10Base-T) standard.

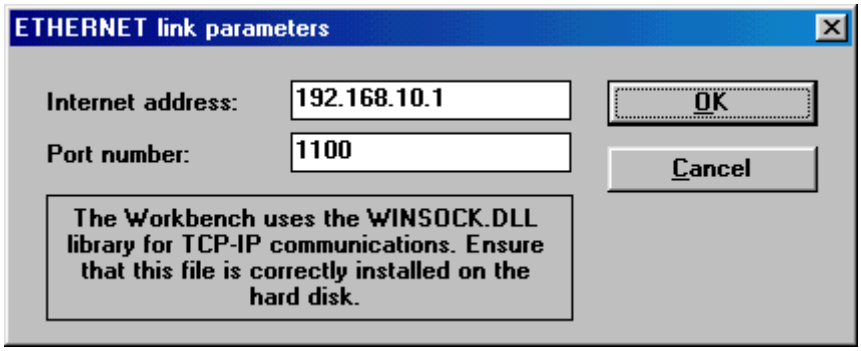


After having modified the params_com, which acts on Alto, generated and loaded the application to the PLC you must select the corresponding port in the ISaGRAF workbench. Click on « **Link setup** » in the « **Debug** » menu in the « Programs » window to do this.

The following window will appear. Select « **Ethernet** » in the scrollable « **Communication Port** » menu. The slave number of this port should be « 1 ». Now click the « **Setup** » button.



Enter the IP address of your Alto in the « **Internet address** » field.



3.1.2. Particular parameters for the B530 and B531 boards :

- **IP address:** this identifies the network and the device (Alto PLC) on a TCP/IP network.
By default, the IP address is 255.255.255.255. In this case, Alto ignores the other parameters and uses a BOOTP address server, which will send a free IP address to Alto.
Format : xxx.xxx.xxx.xxx where xxx [0..255]

- **Sub-network mask:** address mask used to show the breakdown of the IP address into sub-network address and device address on the sub-network. This 32-bit mask is composed entirely of 1's for all the sub-network address parts and entirely of 0's for the device address parts. Using the sub-network mask, Alto determine if it must contact the gateway to reach a recipient according to the IP address of the recipient and the sub-network mask according to the following algorithm:
Format : xxx.xxx.xxx.xxx where xxx [0..255]

- **Gateway address**: IP address of the gateway on the network. If Alto wishes to communicate outside the network to which it belongs, it must address this gateway. By default, this address is 127.0.0.1 and identifies Alto itself (not the gateway).
  Format : xxx.xxx.xxx.xxx where xxx [0..255]

- **Alto Name:** Alto symbolic name. (Must be defined).
  Format : 10 alphanumeric characters at the most

- **DNS Address**: DNS *(Domain Name Server)* IP address. This server returns an IP address from a symbolic name identifying device or server on a TCP/IP network.
  Format : xxx.xxx.xxx.xxx where xxx [0..255]

SNMP parameters :

- **Num_Agent** : SNMP Agent number in the branche LAI(4273) ; by default to 0 : SNMP service deactivated
- **AdrIP_Manager** : SNMP manager IP address : response only for the request of this manager ; by default to 255.255.255.255 : responses to request from any SNMP manager.
- **Emplacement** : « location » field in Alto MIB II
- **Community** : « public » by default ; can be personnalised : Alto will respond only to request send by a manager of his community.

3.1.3. Non Volatile Variables

Alto is equipped with a **1024 byte backed-up memory**. To back up a variable in the event of an Alto power failure, simply **check the "non volatile" box** when declaring the variable in the Dictionary. With an Alto, the execution parameters of an ISaGRAF application do not have to be configured as specified in the ISaGRAF User's Guide.

The space occupied per type of variable is as follows:

- 1 byte per Boolean variable,
- 4 bytes per analog variable + 4 bytes for all the analog variables together
- 5 bytes per time-out variable
- 1 byte per character of a message variable + 3 bytes per message variable.

The only constraint is that if 1 non volatile variable is checked, then one of each type must be checked. 4 types of variables can be non volatile: Boolean, analog, time-out and message.

3.1.4. Backed-up Clock

Alto ISaGRAF is equipped with a **backed-up software clock**. This clock **gives the date, time and day of the week**. This data can be read or written using C functions in the ISaGRAF workbench:

Function	**DayTim_O**
ACTION	Initializes access to the clock on Alto
SYNTAX	*Boolean Status DayTim_O();*
PARAMETERS	None
RETURNED VALUE	Status: TRUE=initialization correct FALSE=initialization error
DESCRIPTION	Only one initialization is required for a project
EXAMPLE	Status:= DayTim_O(); (* initializes access to the clock *)

Function	**DayTim_W**
ACTION	Sets the time (date and time of day) on Alto clock.
SYNTAX	*Boolean Status DayTim_W(analog InfoType, message String);*
PARAMETERS	InfoType: 0: date 1: time 2: day of the week

	String: message according to the type of modified info:
RETURNED VALUE	Status: TRUE= initialization correct FALSE= initialization error
DESCRIPTION	For the date: YYYY/MM/DD for the time: HH:MM:SS.hh for the day: "Sunday" = 0 / "Monday" = 1 / "Tuesday" = 2 / "Wednesday" = 3 / "Thursday" = 4 / "Friday" = 5 / "Saturday" = 6
EXAMPLE	**Write or update** a date, time and day of the week. For example, Friday December 31 2002 at 23 hours 58 minutes 10 seconds and 02 hundredths of a second: Boolean1:= DayTim_W(0, '2002/12/31'); Boolean2:= DayTim_W(1, '23:58:10.02'); Boolean3:= DayTim_W(2, '5');

Function	**Day_Time**
ACTION	Gives the date, time or day in the form of a message string.
SYNTAX	*message Status Day_Time(analog InfoType);*
PARAMETERS	**see User's Guide**
RETURNED VALUE	**see User's Guide**
DESCRIPTION	**ISaGRAF FUNCTION: see User's Guide** Alto ISaGRAF also shows hundredths of a second. The time is therefore given in the following format: HH:MM:SS.hh
EXAMPLE	**see User's Guide**

3.1.5. Compact Flash memory

Alto support a Compact Flash memory to store information. Only UCR B530 board accept it.

A PC soft, named ICF Manager allows you to download all storage data, at CSV, XLS, or HTML format.

3.1.5.1. Compact Flash implementation

9 C functions allows you to initialize, format, prepare, save data on Compact flash :

- CCF_O and CCF_C functions : initialize and close Compact Flash embedded soft.
- CCF_FOR function : format the Compact Flash : function not required if the Compact Flash is already formatted.
- CCF_SU and CCF_SF functions : prepare a structure data to be stored, then close it (storage in a buffer, and option writing in Compact Flash). **Writing on Compact Flash is automatically managed**, to optimize the number of writing operation (max number writing : 300000 for a commercial Compact Flash, 1000000 for an industrial Compact Flash).
- CCFW_B, CCFW_A, CCFW_T, CCFW_M functions : store data in buffer : those functions must be used after CCF_SU and before CCF_SF.

Function	**CCF_O**	
ACTION	Initialize Compact Flash embedded soft	
SYNTAX	*analog CCF_O(analog Percent, analog Channel);*	
PARAMETERS	Percent [0..100]	occupation percentage of Compact Flash memory : over it a writing operation return status at 1000
	Channel []	Number of TCP channel for backup PC program ICF Manager
RETURNED VALUE	See list CCF errors	
DESCRIPTION	Only one open operation is necessary for one project.	
EXAMPLE	Open_CCF := CCF_O(85, 700); (*85 is occupation percentage over it one alarm is send, 700 is the TCP channel number*)	

Function	CCF_C
ACTION	Close Compact Flash embedded soft
SYNTAX	*analog CCF_O();*
PARAMETERS	None
RETURNED VALUE	See list CCF errors
DESCRIPTION	Only if Compact Flash embedded soft Initialized
EXAMPLE	Close_CCF := CCF_C(); (*Close Compact Flash embedded soft *)

Function	CCF_FOR	
ACTION	Format Compact Flash board	
SYNTAX	*analog CCF_FOR(analog Percent, boolean Quick);*	
PARAMETERS	Percent (0..100)	occupation percentage of Compact Flash memory : over it a writing operation return status at 1000
	Quick (True, false)	True: Quick format (for a new board or ICF volume). False: obligatory to format an MFS volume
RETURNED VALUE	See list CCF errors	
DESCRIPTION	Only one format is necessary : in case of Compact Flash isn't formatted : CCF_O function allow to know if the Compact Flash is or not formatted.	
EXAMPLE	Format_CCF := CCF_FOR(85, True); (* Format the Compact Flash : 85 is the occupation percent, True = Quick format operation*)	

Function	CCF_SU	
ACTION	Ouvre un enregistrement en donnant son format : nombre et type de variables qu'il comportera.	
SYNTAX	*analog CCF_SU(analog NBBOO, analog NBANA, analog NBTIME, analog NBMSG);*	
PARAMETERS	NBBOO [0..99]	Nombre de variables de type booléennes par enregistrement.
	NBANA [0..99]	Nombre de variables de type entières par enregistrement.
	NBTIME [0..99]	Nombre de variables de type date/heure par enregistrement.
	NBMSG [0..99]	Nombre de variables de type message par enregistrement.
RETURNED VALUE	See list CCF errors	
DESCRIPTION	Cette fonction est à utiliser avant chaque enregistrement.	
EXAMPLE	Enreg_CCF := CCF_SU(2, 4, 1, 3); (* Prépare le format d'un enregistrement qui comportera : 2 booléens, 4 analogs, 1 date, 3 messages *)	

Function	CCFW_B	
ACTION	Ecrit un booléen dans le cache de la compact flash	
SYNTAX	*analog CCFW_B(booléen Value);*	
PARAMETERS	Value	Valeur booléenne qui sera écrite dans la mémoire cache .
RETURNED VALUE	See list CCF errors	
DESCRIPTION	Cette fonction est à utiliser après avoir définit le format de l'enregistrement (fonction CCF_SU), et autant de fois que le paramètre NBBOO l'a défini dans cette même fonction CCF_SU.	
EXAMPLE	EnregB_CCF := CCFW_B(B_DG1); (*B_DG1 est par exemple un booléen définit dans le dictionnaire Isagraf, correspondant à une entrée TOR : ici état ouvert ou fermé d'un disjoncteur ; la fonction écrit cette variable dans le cache*)	

Function	CCFW_A	
ACTION	Ecrit un analog dans le cache de la compact flash	
SYNTAX	*analog CCFW_A(analog Value);*	
PARAMETERS	Value	Valeur entière qui sera écrite dans la mémoire cache .
RETURNED	See list CCF errors	

VALUE	
DESCRIPTION	Cette fonction est à utiliser après avoir définit le format de l'enregistrement (fonction CCF_SU), et autant de fois que le paramètre NBANA l'a défini dans cette même fonction CCF_SU.
EXAMPLE	EnregB_CCF := CCFW_B(A_NIV1); (*A_NIV1 est par exemple un analog définit dans le dictionnaire Isagraf, correspondant à une entrée analogique : ici, niveau d'eau ; la fonction écrit cette variable dans le cache*)

Function	**CCFW_T**	
ACTION	Ecrit une date et une heure dans le cache de la compact flash	
SYNTAX	*analog CCFW_T(message Date, message Heure);*	
PARAMETERS	Date	Variable message au format suivant : 'AAAA/MM/JJ' : AAAA : année ; MM : mois ; JJ : jour
	Heure	Variable message au format suivant : 'HH:MM :SS.CC ' : HH : heure ; MM : minutes ; SS : secondes ; CC : centièmes de secondes
RETURNED VALUE	See list CCF errors	
DESCRIPTION	Cette fonction est à utiliser après avoir définit le format de l'enregistrement (fonction CCF_SU), et autant de fois que le paramètre NBTIME l'a défini dans cette même fonction CCF_SU.	
EXAMPLE	EnregT_CCF := CCFW_T(Date, Heure); (*Date et heure étant définis dans le dictionnaire Isagraf en tant que messages, et utilisant directement l'horloge interne d'Alto : Date :=Day_Time(0) ; Heure:=Day_Time(1) ; voir l'utilisation de l'horloge pour plus de renseignements ; la fonction écrit ces variable dans le cache*)	

Function	**CCFW_M**	
ACTION	Ecrit un message dans le cache de la compact flash	
SYNTAX	*analog CCFW_T(message Msg);*	
PARAMETERS	Msg	Variable message qui sera écrite dans la mémoire cache : taille maximale de cette variable : 255 caractères.
RETURNED VALUE	See list CCF errors	
DESCRIPTION	Cette fonction est à utiliser après avoir définit le format de l'enregistrement (fonction CCF_SU), et autant de fois que le paramètre NBMSG l'a défini dans cette même fonction CCF_SU.	
EXAMPLE	EnregM_CCF := CCFW_M(Msg1); (*Msg1 est défini dans le dictionnaire Isagraf en tant que message ; la fonction écrit cette variable dans le cache*)	

Function	**CCF_SF**	
ACTION	Ferme un enregistrement ouvert, met à jour la mémoire cache, et l'écrit si demandé dans la compact flash.	
SYNTAX	*analog CCF_SF(booléen Doflush);*	
PARAMETERS	Doflush	True: Forçage du flush des caches sur la compact flash : c'est l'écriture des données sur la compact flash. False: Ecriture gérée automatiquement : les caches ne sont flushés que si ils sont pleins.
RETURNED VALUE	See list CCF errors	
DESCRIPTION	Cette fonction est à utiliser après l'écriture complète de l'enregistrement en mémoire cache, avec les functions CCFW_x ; attention à la durée de vie d'une compact flash. Taille des caches par type de variable : 4096 booléens, 128 analogs, 16 time, 2 messages : lorsque ces tailles sont atteintes, l'écriture est forcée ; correspondent par secteur (512 octets).	
EXAMPLE	Flush_CCF := CCF_SF(True); (* Force le flush des caches *)	

CCF list errors : returned values by precedent functions :

Returned Value	-30000 : internal error. -22000 : defect Compact Flash board . -21000 : Writing error. -20000 : Reading error. -1000 : Compact Flash full : occupation over percent defined in CCF_O function . -650 : Not correct number of elements at writing operation. -600 : Not correct number of elements has been writing -500 : Closing error -400 : Opening error. -300 : Format error. -200 : no more allocation memory . -100 : Not correct parameters. 1 : No error, correct execution. 500 : Embedded Compact Flash soft Not initialized. 1000 : Logical max threshold exceeded for a data type. 3000 : Quick format operation done. 5000 : Compact Flash board occupied.

Example : see project « AICCF »

This program is storing data every 10 seconds.

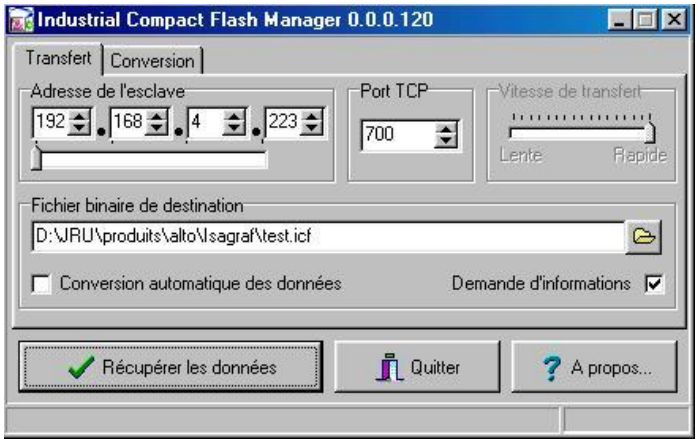
In the sequential function chart « CCF » :

- Step 1 : initialization : function CCF_O
- Step 2 : timer 10 seconds : backup every 10 seconds
- Step 3 : storage format preparation : function CCF_SU
- Step 4 : data storage in memory buffer
- Step 5 : closing current data buffer and automatic storage in compact flash memory

3.1.5.2. ICF Manager :

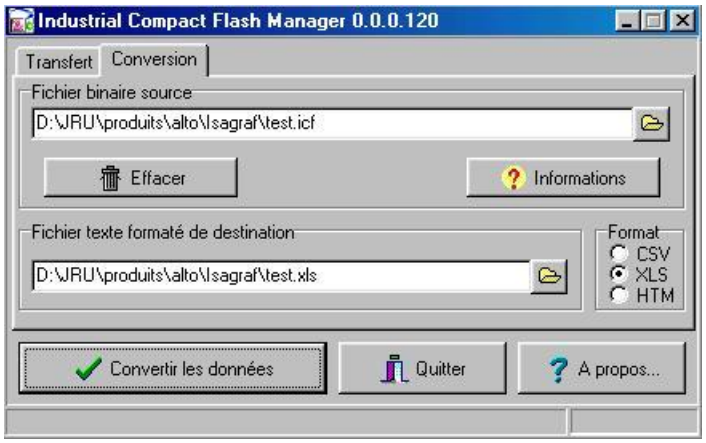
Start ICF Manager : two windows are available. In the transfer window, you have to choose :

- The IP address of Alto ISaGRAF ;
- The TCP communication port : 700 for example.
- A new binary destination file name
- Select the information data mode or transfer data mode
- Select Automatic conversion data option if you don't want to do it on the second window of ICF Manager

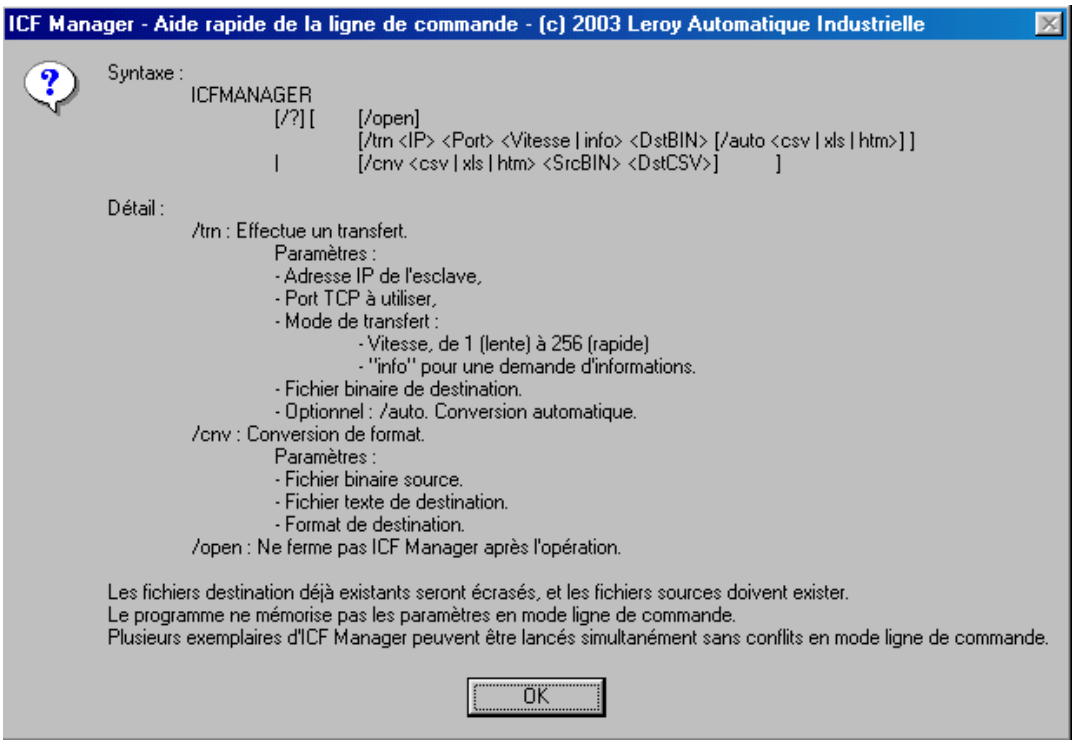


In the convert window, you have to select :

- the source binary file to convert
- the format and name of destination file ; three format are available : CSV, Excel, Html.



You can automate the data backup with batch file :

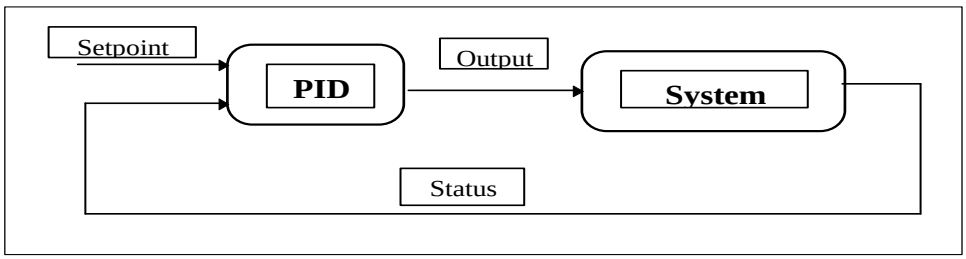


3.1.6. PID control

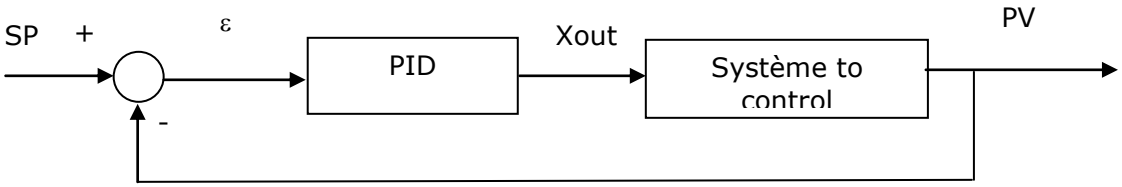
Principle on Alto ISaGRAF :

A PID controller is used to control industrial processes.

It processes the difference between the value of the set point and the output value:



PID model implanted on Alto ISaGRAF :



SP = Set Point, PV = Point Value

$Xout(t)=Kp(\varepsilon(t)+\frac{1}{Ti}\int_0^t \varepsilon(t)d(t)+Td\frac{d\ \varepsilon\ (t)}{dt})$ in continuous

$Xout(k)=Kp(\varepsilon(t)+\frac{Ts}{Ti}I(k)+\frac{Td}{Ts}(\varepsilon(k)-\varepsilon(k-1)))$ in discrete with $I(k)=I(k-1)+\varepsilon(k)Ts$.

C functional block characteristics : PID_AL :

Block functional	**(i) PID_AL**	
ACTION	Initialize, parameter and refresh a PID block	
SYNTAXE	*Instance(*boolean Auto, real Pv , real Sp, real X0, real Kp, real Ti, real Td, timer Ts, real Min, real Max*);* *Commande :=Instance.Xout;*	
PARAMETERS	Auto	function mode : automatic : Auto=True manual : Auto=False
	Pv	Point Value
	Sp	Set Point
	X0	Output pid value for the manual mode
	Kp	proportional gain general to all actions
	Ti	integration constant (unit : s)
	Td	dérivation constant (unit : s)
	Ts	Sampling period (unit : ms)
	Min, Max	limit values low and high for output
RETURN VALUE	Xout	Output value
EXAMPLE	pid1 is an instance of pid_al (dictionary, « BF instances ») pid1(Auto, Pv , Sp, X0, Kp, Ti, Td, Ts, Min, Max); Output :=pid1.xout ;	

Nota : parameter Auto must be to false at initialization.

This processing is broken down into **three actions: proportional, integral, derivative**. Each action can be adjusted separately meaning that control can be achieved from each of the three actions.

A PID is implemented through the use of a C functional block called ***Pid_AI()*** as follows:

- **declare a PID instance "*pid1*"** in the dictionary,
- once during the cycle, **call the instance with its parameters**,
- example: *pid1(Auto1, PointValue1, setpoint1, X01, kp1, ti1, td1, ts1, Min1, Max1);*
- *setpoint1* and are analog inputs, *output1* is an analog output,
- the **output value** returned is in: *output1:= pid1.Xout;*

It is possible to **build a P, I, PI PD, ID or PID controller**. To do this, simply disable the action which is not used. An action is considered disabled when the parameters on which it depends are set to zero.

See example project : «pid_dem.pia »

Setting the PID Controller

The PID controller is set by selecting the following parameters: Kp, Ti, Td.

The Kp, Ti and Td parameters can be determined using the experimental analysis methods of the process.

For example, the typical specifications for devices controlling chemical or thermal processes are as follows:

- Ti from 3 to 1000 seconds,
- Td from 3 to 150 seconds.

An on-line setting method: the trial and error method.

On-line setting can be carried out in an empirical manner using the procedure summed up below:

- install the control system,
- remove the integral and derivative actions,
- set the gain, Kp, to a low value,
- vary the setpoint slightly and observe the system response. Since the gain is very low, the response will only be slight,
- double the gain and repeat the above step. Go on in this way until the response starts to oscillate. Let us call this value Kpu (ultimate Kp),
- set Kp to (Kpu / 2),
- repeat the same operation, reducing Ti by a factor of 2 until an oscillating response is obtained for a slight variation in the setpoint,
- set Ti to twice this value,
- proceed in the same way for the derivative constant: increase Td until an oscillating response is obtained, then set Td to 1/3 of this value.

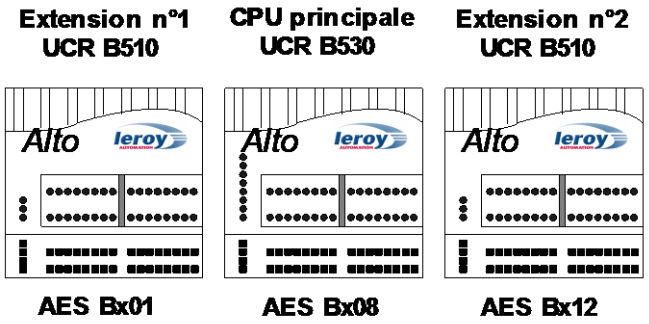
3.2. Extension CPU configuration

Only **UCR B530** board is able to manage extension UCR board.

Extension boards are named **UCR B510** : this board allows communication with UCR main board. Communication between UCR main board and UCR extension board is done with an infrared net using CAN protocol.

2 extension boards can be add : they must be placed from each side of UCR main board ; analog **AES boards can't be placed under one UCR extension board.**

Example of material configuration :



To do this, two settings must be done :

3.2.1. Mechanical setting

On UCR B510 are placed 5 switches :

- Switches 1 and 2 : binary coding of logic number extension
- Switches 3 and 4 : ON
- Switch 5 : OFF

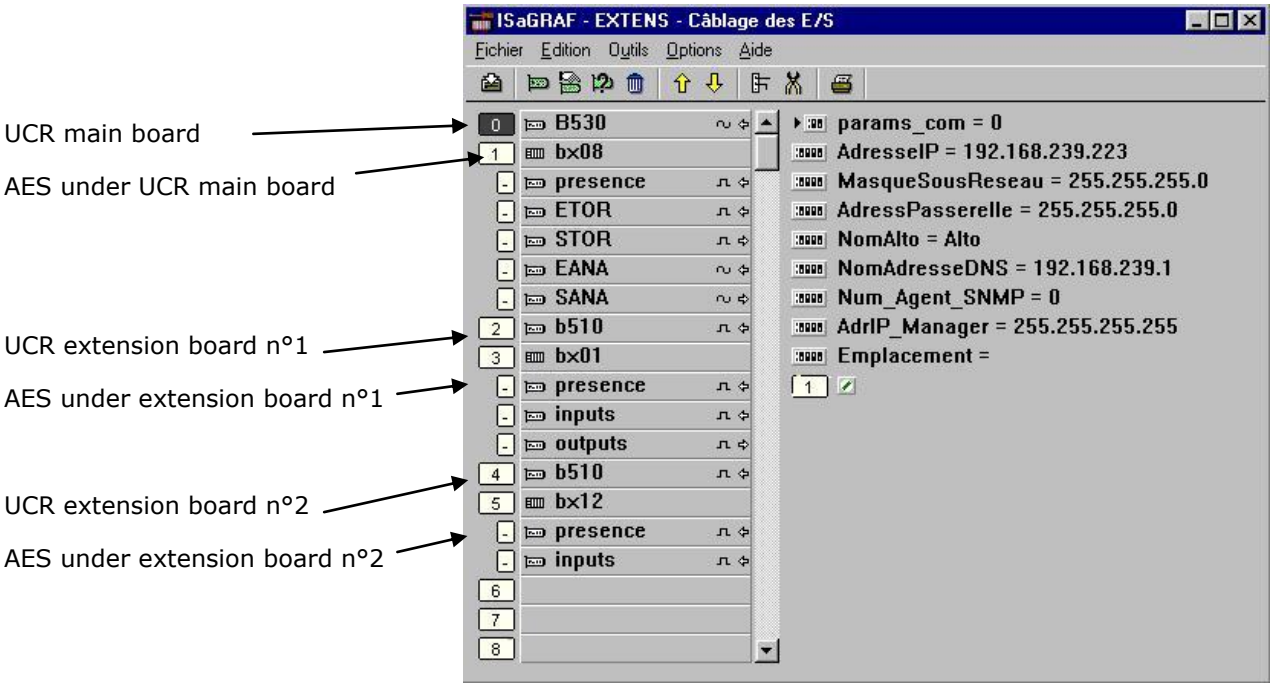
Example :

	Switch1	Switch2	Switch3	Switch4	Switch5
extension number 1	OFF	ON	ON	ON	OFF
extension number 2	ON	OFF	ON	ON	OFF

3.2.2. Soft settings

In your project, modify your I/O configuration : select the menu « project » + « I/O Connection » and add UCR B510 extension boards (slots 2 and 4), and AES I/O boards (slots 3 and 5) under B510 : then connect the I/O variables to the AES boards.

The example corresponding to the precedent material configuration is :



4. Serial Communication Management

Alto ISaGRAF is equipped with 2 **serial links** and **one Ethernet link**.

J2 connector is the initial console link. This console link can be connected to the Ethernet port as shown before. The console link enables the communication with ISaGRAF workbench (downloading, debugging...).

The different types of UCR are as follows:

Connector number	J1	J2	J3
UCR Type	UCR 53x	UCR 52x and UCR 53x	UCR 52x and UCR 53x
Connector Type	RJ45	SubD9	SubD9
Link	Ethernet	RS232/RS485	RS485
Protocols availables	TCP/IP : Modbus/TCP (master/slave), SNMP, SMTP	* Modbus asynchronous master or slave *Transmission/Reception bytes	* Modbus asynchronous master or slave *Transmission/Reception bytes
Console Link	Customizable	**initial**	
Soft Identification		**Terminal Block 1 / Com0**	**Terminal Block 1 / Com1**

Figure 5: Alto Communication Configurations

Communication on Alto is managed via **2 software layers**.

The **bottom layer** is specific to each serial link. It stores the bytes received, detects an end-of-frame character when the silent period is overrun and transmits any answer from Alto. It is interrupt-driven (specific to the serial link) and is transparent for the user program.

The **top layer** is independent of the serial links. It analyzes the received frame, carries out any work requested by the master and prepares the answer to be transmitted. This layer is processed by user functions specific to each protocol.

The following protocols are discussed in the following sections of this manual:

Protocols on RS232 and RS485 network :

- Slave Jbus,
- Master Jbus,
- Simple transmission/reception protocol.

Caution: the choice of serial link (RS232 or RS485) depends entirely on the wiring adopted.

Protocols on Ethernet :

- ModBus/TCP : slave and master
- SNMP : Alto SNMP variables are read/write accessible for a SNMP manager.
- SMTP : Alto can send emails.

4.1. Theory of Communication on serial communication Ports

The other RS232/485 or RS232C serial ports support either the **Modbus/Jbus protocol,** or a specific protocol based on **byte transmission/reception**, or the **console link** if necessary.

The Modbus/Jbus protocol is used on an Alto port as follows:

initializing (or declaring) a Slave or Master Modbus/Jbus communication Port is done by using a specific C function from the workbench. This function is used to define the communication parameters on the Port and link them to an exchange table of n words.

Frames coming from a master or slave will refresh this table. The data of each table can be used from the workbench (dictionary variables) using specific functions:

- **Word_R():** reads a word from a table to a dictionary analog variable in unsigned form,
- **Word_W():** writes an analog variable from the dictionary to a word in a table,
- **Bit_R():** reads a bit from a table to a dictionary Boolean variable,
- **Bit_W():** writes a Boolean variable from the dictionary to a bit in a table,
- **DWord_R():** reads two words from a table to a dictionary analog variable,
- **DWord_W()**: writes an analog variable from the dictionary to two words in a table,
- **WordS_R:** reads a word from a table to a dictionary analog variable in signed form.

Example of a Slave Jbus protocol:

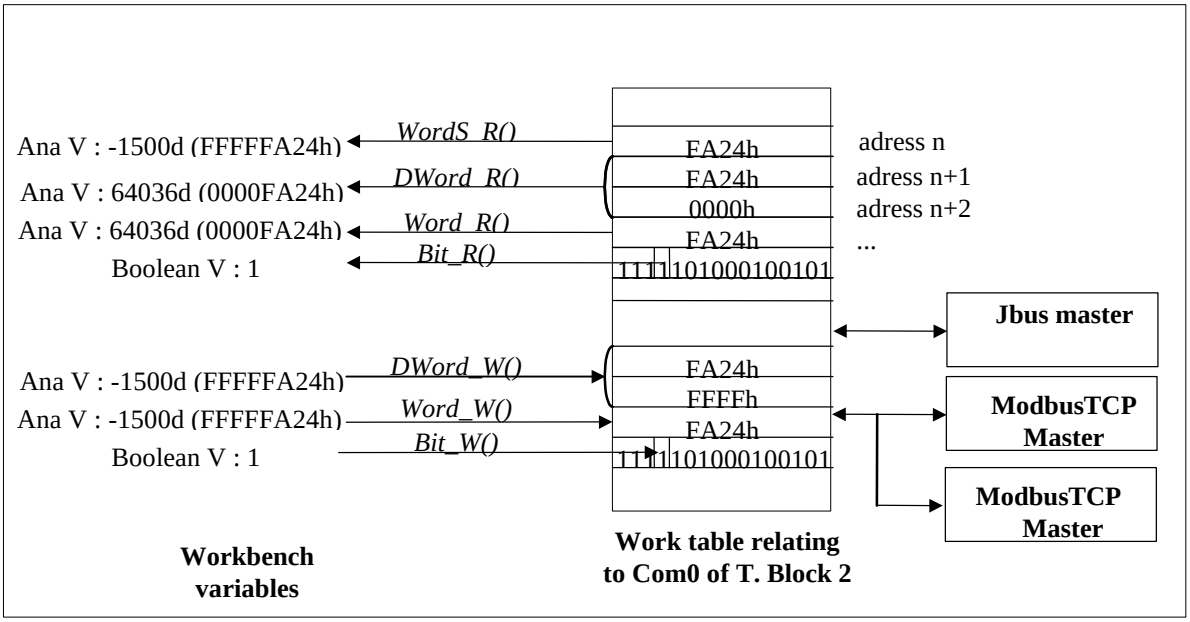


Figure 6: Slave Jbus: Theory of Communication

Size and form of analog variables and words contained in exchange tables:

Under ISaGRAF, integer variables are encoded on 32 bits. They can be represented in signed decimal form [-2147483648..2147483647] or unsigned hexadecimal form [00000000..FFFFFFFF]. Word variables (16 bits) contained in an exchange table are read in their unsigned form by the Word_R() function. In order to obtain these variables in their signed form, the WordS_R() function must be used. *See the examples given in figures 8 and 9.*

Example for a Master Jbus protocol:

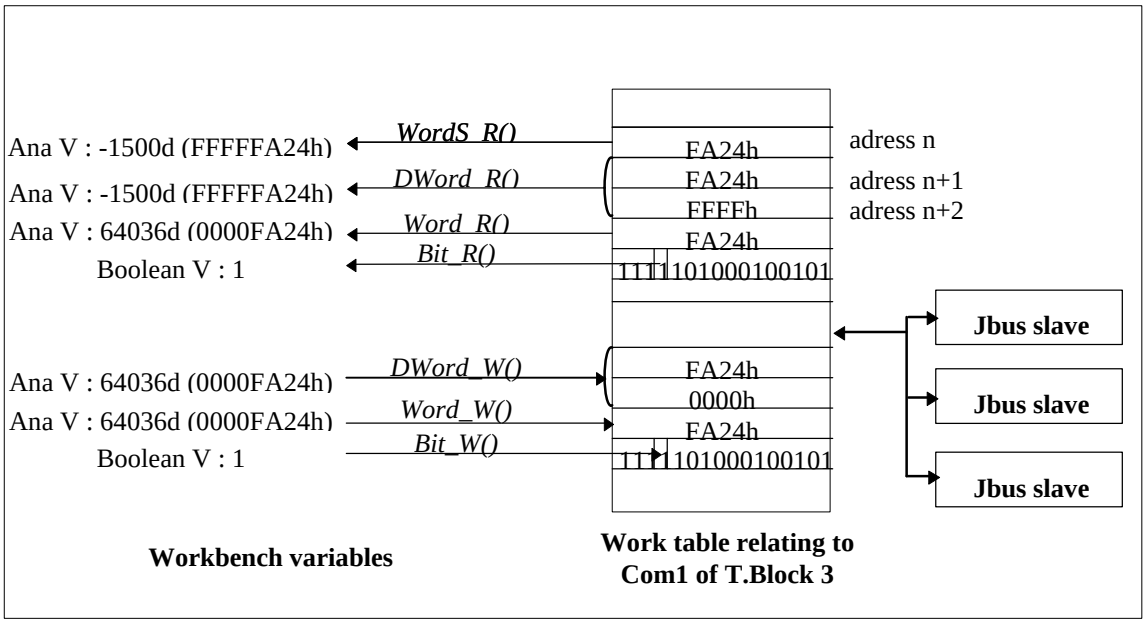


Figure 7: Master Jbus: Theory of Communication

The following sections of the manual describe the procedures for implementing each protocol.

Important note: 1 table can be associated with several communication ports. The theory is to declare a table by its number (from 1 to 7), then to use its number when declaring another port. If you are using a table that has already been declared, the length entered as a parameter must be the same as that of the initial table.

The following Jbus/Modbus orders are recognized and processed by Alto ISaGRAF:

Function	Function Code
read several bits	1 and 2 *(1)*
read several words	3 and 4 *(2)*
write a bit	5
write a word	6
write several words	16

(1) Alto does not distinguish between output bits and input bits
(2) Alto does not distinguish between input words and output words.

Function	**Word_R**
ACTION	Reads 1 word, in unsigned form, in a network table associated with an Alto communication port
SYNTAX	*analog Data Word_R(analog TableNum, analog WordAddr);*
PARAMETERS	TableNum: [1..7] WordAddr: [0.."max value entered when opening Com - 1"]
RETURNED VALUE	Data: analog value read [0..FFFFh]. **The most significant bits (2nd word) of the analog variable are set to zero.** If error: 10000h
EXAMPLE	*Data:= Word_R(1, 2);* (* Reads a word at address 2 of table 1 *)

Function	**DWord_R**
ACTION	Reads 2 words in a network table associated with an Alto communication

	port
SYNTAX	*analog Data DWord_R(analog TableNum, analog WordAddr);*
PARAMETERS	TableNum: [1..7] WordAddr: [0.."max value entered when opening Com - 2"]
RETURNED VALUE	Data: analog value read [0..FFFFFFFFh] initialized at the value: -1
EXAMPLE	*Data:= DWord_R(1, 2);* (* Reads 2 words at address 2 of table 1 *)

Function	**WordS_R**
ACTION	Reads 1 word, in unsigned form, in a network table associated with an Alto communication port
SYNTAX	*analog Data WordS_R(analog TableNum, analog WordAddr);*
PARAMETERS	TableNum: [1..7] WordAddr: [0.."max value entered when opening Com - 1"]
RETURNED VALUE	Data: analog value read [0..FFFFh]. **The most significant bits (2nd word) of the analog variable are set to 1.** If error: 10000h
EXAMPLE	*Data:= WordS_R(1, 2);* (* Read a word at address 2 of table 1 *).

Function	**Bit_R**
ACTION	Reads 1 bit in a network table associated with an Alto communication port
SYNTAX	*analog Data Bit_R(analog TableNum, analog WordAddr, analog BitOrder);*
PARAMETERS	TableNum: [1..7] WordAddr: [0.."max value entered when opening Com - 1"] BitOrder: [0..Fh]
RETURNED VALUE	Data: analog value • bit set to 0: 0 • bit set to 1: 1 • error: -1
EXAMPLE	*Data:= Bit_R(1, 2, 5);* (* reads bit 5 of word 2 of table 1 *).

Function	**Word_W**
ACTION	Writes 1 word in a network table associated with an Alto communication port
SYNTAX	*Boolean Status Word_W(analog TableNum, analog WordAddr, analog Data);*
PARAMETERS	TableNum: [1..7] WordAddr: [0.."max value entered when opening Com - 1"] Data: analog value to write [0..FFFFh]. **The most significant bits (2nd word) are ignored.**
RETURNED VALUE	Status: [TRUE, FALSE];
EXAMPLE	*Word:= 16#FF;* *Status:= Word_W(1, 2, Word);* (* Writes the analog variable Word at address 2 of table 1 *)

Function	**DWord_W**
ACTION	Writes 2 words in a network table associated with an Alto communication port
SYNTAX	*Boolean Status DWord_W(analog TableNum, analog WordAddr, analog Data);*
PARAMETERS	TableNum: [1..7] WordAddr: [0.."max value entered when opening Com - 2"] Data: analog value to write [0..FFFFFFFFh]
RETURNED VALUE	Status: [TRUE, FALSE];
EXAMPLE	*Word:= 16#FF008800;* *Status:= DWord_W(1, 2, Word);* (* Writes the analog variable Word at addresses 2 and 3 of table 1 *)

Function	**Bit_W**
ACTION	Writes 1 bit in a network table associated with an Alto communication port
SYNTAX	*Boolean Status Bit_W(analog TableNum, analog WordAddr, analog BitOrder, Boolean Data);*
PARAMETERS	TableNum: [1..7] WordAddr: [0.."max value entered when opening Com - 1"] BitOrder: [0..Fh] Data:[TRUE, FALSE]
RETURNED VALUE	Status:[TRUE, FALSE]
EXAMPLE	*BitStatus:= TRUE;* *Status:= Bit_W(1, 2, 5, BitStatus);* (* bit 5 of word 2 of table 1 set to1 *)

4.2. Protocols on RS232 and RS485 network :

4.2.1. Slave Jbus Protocol

In order to use the Slave Jbus protocol on a communication port, 3 C functions are available.

Function	**JbusS_O**
ACTION	Opens a Slave Jbus port on an Alto
SYNTAX	*Boolean JbusS_O(analog TerminalBlock, analog Com, analog SlaveNum, analog TableNum, analog TableLength);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3 SlaveNum: [0..255] TableNum: [1..7] TableLength: [1..4095] 4095 words (16 bits)
RETURNED VALUE	FALSE: Not opened. TRUE: Correctly opened.
EXAMPLE	*Status:= JbusS_O(1, 1, 12, 3, 100);* (* declares J3 on SlaveJbus. A table of 100 words, accessible in read or write mode, is associated with this port. This table is identified by the number 3. The slave no. is 12. By default, the communication parameters are: 19200 bauds, no parity, 1 stop bit, 8 data bits. This table starts at address 0 *)

Function	**JbusS_P**
ACTION	Configures the communication parameters of a Slave Jbus port on an Alto.
SYNTAX	*Boolean JbusS_P(analog TerminalBlock, analog Com, analog Speed, analog Parity, analog StopBit, analog Data, analog Access, analog Silence, analog basic NetworkAddr);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3 Speed: • 75Bauds=1, • 110Bauds=2, • 150Bauds=3, • 300Bauds=4, • 600Bauds=5, • 1200Bauds=6, • 2400Bauds=7, • 4800Bauds=8, • 9600Bauds=9, • 19200Bauds=10, • 38400Bauds=11, • 76800Bauds=12, (only on J2) • 115kbauds=13, (only on J2) • 200Bauds=14. (only on J2) Parity: • No parity=0, • Even=1, • Odd=2, • Forced to 0=3, • Forced to 1=4. StopBit: • 1Stop=1, • 2Stop=2. Data: • 5Bits=5, • 6Bits=6,

	• 7Bits=7, • 8Bits=8. Access: • Read/Write=0 • Read=1 • Write=2 • None=3 Silence: [0..7FFFh] NetworkAddr: [0..FFFFh]
RETURNED VALUE	FALSE: Not configured. TRUE: Correctly configured.
EXAMPLE	*Status:= JbusS_P(1, 1, 9, 0, 1, 8, 2, 0, 1000);* (* modifies Slave Jbus parameters on J3. The com parameters are now: 9600 bauds, no parity, 1 stop bit, 8 data bits. Access is in read-only mode: 2. The related Jbus table starts at address 1000. *)

Function	**JbusS_C**
ACTION	Closes a Slave Jbus port on an Alto.
SYNTAX	*Boolean JbusS_C(analog TerminalBlock, analog Com);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3
RETURNED VALUE	FALSE: Not closed. TRUE: Correctly closed.
EXAMPLE	*Status:= JbusS_C(1, 1);* (* closes the Slave Jbus port on J3*)

The **console link (J2 by default)** supports the **ISaGRAF slave Modbus**. This protocol provides access to ISaGRAF application variables via their network address. This network address is defined in the workbench dictionary.

Only ***Boolean*** or ***Analog*** variables are accessible. The Modbus functions recognized by the ISaGRAF protocol are as follows:

1	Read n bits
3	Read n words
5	Write 1 bit
6	Write 1 word
16	Write n words

Caution: the ISaGRAF Modbus protocol does not manage error codes such as "unknown Modbus address".

The following list shows the default communication parameters of this console link:

- slave no.: 1,
- speed: 19200 bauds,
- parity: none,
- data: 8 bits,
- stop bits: 1.

4.2.2. Master Jbus Protocol

In order to use the Master Jbus protocol on a communication port, 4 C functions are available:

Function	**JbusM_O**
ACTION	Opens a Master Jbus port on an Alto
SYNTAX	*Boolean JbusM_O(analog TerminalBlock, analog Com, analog TableNum analog TableLength, analog TimeOut, analog NbrTests);*
PARAMETERS	TerminalBlock: [1]

	Com: 0= J2 ; 1 = J3 TableNum: [1..7] TableLength: [1..4095] 4095 words (16 bits) TimeOut: [0..7FFFh] in milliseconds NbrTests: [0..9] number of additional tests upon reception of a "no slave" answer
RETURNED VALUE	FALSE: Not opened. TRUE: Correctly opened.
EXAMPLE	*Status:= JbusM_O(1, 1, 1, 100, 500, 3);* (* declares the J3 connector as a Master Jbus. An exchange table of 100 words is associated with this port. This table is identified by the number 1. The time-out is 500 ms and the number of retries in the event of a "no slave" answer is 3. By default, the com parameters are 19200 bauds, no parity, 1 stop bit, 8 data bits. *)

Function	**JbusM_P**
ACTION	Configures communication on a Master Jbus port on an Alto
SYNTAX	*Boolean JbusM_P(analog TerminalBlock, analog Com, analog Speed; analog Parity; analog StopBit; analog Data, analog RecTOut, analog BrdTOut, analog BusyRet);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3 Speed: • 75Bauds=1, • 110Bauds=2, • 150Bauds=3, • 300Bauds=4, • 600Bauds=5, • 1200Bauds=6, • 2400Bauds=7, • 4800Bauds=8, • 9600Bauds=9, • 19200Bauds=10, • 38400Bauds=11, • 76800Bauds=12, (only on J2) • 115kbauds=13, (only on J2) • 200Bauds=14. (only on J2) Parity: • No parity=0, • Even=1, • Odd=2, • Forced to 0=3, • Forced to 1=4. StopBit: • 1Stop=1, • 2Stop=2. Data: • 5Bits=5, • 6Bits=6, • 7Bits=7, • 8Bits=8. RecTOut: [0.. 7FFFh] in milliseconds silent period in addition to the 3 end-of-frame detection characters: 0 by default, BrdTOut: [0.. 7FFFh] in milliseconds silent period in addition to the 3 end-of-frame characters on broadcast: 0 by default, BusyRet: [0.. 9] number of retries following a "slave busy" answer: 0 by default.
RETURNED VALUE	FALSE: Not configured. TRUE: Correctly configured.

EXAMPLE	*Status:= JbusM_P(1, 1, 9, 0, 1, 8, 5, 0, 1);* (* modifies the Master Jbus parameters on J3 port. The com parameters are now: 9600 bauds, no parity, 1 stop bit, 8 data bits. Additional silent period at the end of frame transmission: 5 ms. 1 retry following a "slave busy" answer *)

Function	**JbusM_T**
ACTION	Sends a Master Jbus frame to an Alto port and reads the communication status
SYNTAX	*analog JbusM_T(analog TerminalBlock, analog Com, analog SlaveNum, analog FunctionCode, analog SlaveAddress, analog Length, analog DataAddress, Boolean SendFrame);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3 SlaveNum: [0..255] FunctionCode: • 1 or 2: read n bits • 3 or 4: read n words • 5: write 1 bit • 6: write 1 word • 15: write n bits • 16: write n words SlaveAddress: [0..FFFFh] Length: [1..128] DataAddress: [0.. "max value entered when opening Com - 1"] SendFrame: Boolean: TRUE to send the frame; FALSE to read the status of the previous frame
RETURNED VALUE	If SendFrame = TRUE: 0: Frame not sent 1: Frame correctly sent. If SendFrame = FALSE: result of sent frame (status to JBUS standard) e.g. see appendix
EXAMPLE	(* On com0 of TerminalBlock 2 *) (* read 1 word at address 5 on slave 1, store at address 2 *) *Status:= JbusM_T(1, 1, 1, 3, 5, 1, 2, TRUE);* (* read status of previous frame *) *Status:= JbusM_T(1, 1, 1, 3, 5, 1, 2, FALSE);*

Function	**JbusM_C**
ACTION	Closes a Master Jbus port on an Alto
SYNTAX	*Boolean JbusM_C(analog TerminalBlock, analog Com);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3
RETURNED VALUE	FALSE: Not closed. TRUE: Correctly closed.
EXAMPLE	*Status:= JbusM_C(1, 1);* (* closes the Master Jbus port on J3 port *)

Example: declaration of a Master Jbus communication port to J3.

Use in an ISaGRAF project:

- Open the communication port: JbusM_O(1, 1, 100, 500, 3),
- Configure the communication parameters: JbusM_P(1, 1, 9, 0, 1, 8, 5, 0, 1),
- Read (function code 3) 1 word at address 5 on slave no. 6 and place it at address 12 of the table:
  - JbusM_T(1, 1, 6, 3, 5, 1, 12, TRUE); sends the read frame,
  - JbusM_T(1, 1, 6, 3, 5, 1, 12, FALSE); returns the communication status.
- Close the communication port by ending the program: JbusM_C(1, 1).

4.2.3. Byte Transmission/Reception Protocol

Alto ISaGRAF users can install a byte transmission/reception protocol on the available serial links (except the console link). The C functions provided can be used to install and manage FIFO queues, one for transmission and one for reception. A serial link can be managed in either the RS232 or RS485 standard. This simple protocol is designed to manage terminals, devices with an ASCII protocol, without the time constraints associated with byte transmission and reception. Low-level management of a serial port is carried out by Alto during an interrupt.

After initializing the serial link, users can read or write bytes in the transmission and reception queues. The bytes are transmitted or received on the line by Alto ISaGRAF during an interrupt.

The following C functions are provided:

- **NulPro_0():** opens a simple communication sequence on an Alto port.
- **NulPro_P():** configures a simple communication sequence on an Alto port.
- **NulPro_S():** writes in the transmission queue on an Alto port.
- **NulPro_R():** reads in the reception queue on an Alto port.
- **NulPro_N():** reads the number of characters located in the reception queue on an Alto port.
- **NulPro_C():** closes a simple communication sequence on an Alto port.

Example: declaration of a simple communication sequence on a communication port: com0 of terminal block 2. The communication parameters are:

- TerminalBlock: 2,
- Com: 0,
- reception queue: 1020 words,
- transmission queue: 510 words
- Mode: Half Duplex
- speed: 19200 bauds,
- parity: odd,
- 1 stop bit,
- 8 data bits.

Use in an ISaGRAF project:

- Open the communication port: NulPro_O(1, 1, 1020, 510, 1);
- Read 12 characters in the reception file and place them in the message located at network address 20h of the dictionary NulPro_R(1, 1, 16#20, 12);
- Write the characters of the message located at network address 20h of the dictionary to the transmission file: NulPro_S(1, 1, 16#20, 0);
- Close the communication port: NulPro_C(1, 1);

Example : Printer Management

The byte transmission/reception functions on a serial link can be used for simple control of a **serial printer**. Any Alto serial port can be used to manage a serial printer. RS232C links can also be used to manage control signals such as DTR or XON/XOFF.

Only messages with a network address in the dictionary can be printed.

Send a message to a printer connected to J3:

- Open the communication port: NulPro_O(1, 1, 1020, 510, 1);
- Configure the communication parameters if necessary NulPro_P(...);
- Write the characters of the message located at address 20h of the dictionary NulPro_S(1,1, 16#20, 0);

Example of tested printer: EPSON LX300 (serial and parallel).

Function	**NulPro_O**
ACTION	Opens a simple communication sequence on an Alto port
SYNTAX	*Boolean NulPro_O(analog TerminalBlock, analog Com, analog RecTabLength, analog TransmitTabLength, analog Mode);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3 RecTabLength: [1..1020] reception queue length (1020 = 4 messages)

	TransmitTabLength: [1..510] transmission queue length (510 = 2 messages) Mode: [0..1] HalfDuplex (1) or FullDuplex (0) mode
RETURNED VALUE	FALSE: Not opened. TRUE: Correctly opened.
EXAMPLE	*Status:= NulPro_O(1, 1, 1020, 510, 1);* (* declares a simple communication sequence on J3 A reception queue of 1020 words is associated with this port A transmission queue of 510 words is associated with this port (the current mode is Half Duplex *)

Function	**NulPro_S**
ACTION	Writes in the transmission queue on an Alto port
SYNTAX	*Boolean NulPro_S(analog TerminalBlock, analog Com, analog MsgNetworkAddr, analog NrbChar);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3 MsgNetworkAddr: [0..FFFFh] network address of the ISaGRAF dictionary Note: the message network address must be declared (>0) NbrChar: [1..255] 255: max length of an ISaGRAF message if NbrChar = 0 writes all the characters of the message if NbrChar = n writes n characters of the message
RETURNED VALUE	FALSE: Not written. TRUE: Correctly written.
EXAMPLE	Status:= NulPro_S(1, 1, 16#A0, 0); (* sends the message located at address 16#A0 on the transmission queue associated with J3 *)

Function	**NulPro_R**
ACTION	Reads in the reception queue on an Alto port
SYNTAX	*analog NulPro_R(analog TerminalBlock, analog Com, analog MsgNetworkAddr, analog NbrChar);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3 MsgNetworkAddr: [0..FFFFh] network address of the ISaGRAF dictionary Note: the message network address must be declared (>0) NbrChar: [1..255] 255: max length of an ISaGRAF message if NbrChar = 0 reads all characters of the message if NbrChar = n reads n characters of the message
RETURNED VALUE	[1..n]: message read correct 0: message not read -1: message read but incorrect: the message does not store all the characters -2: message read but incorrect: a character could not be read in the reception queue or the character read request concerns more characters than in the message. -3: no characters in the buffer
EXAMPLE	Status:= NulPro_R(1, 1, 16#A0, 20); (* reads 20 characters in the reception queue associated with J3 these characters are placed in the message located at network address 16#A0 *)

Function	**NulPro_N**
ACTION	Reads the number of characters located in the reception queue on an Alto port.
SYNTAX	*analog NulPro_N(analog TerminalBlock, analog Com);*

PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3
RETURNED VALUE	NbrChar: number of characters in the reception queue
EXAMPLE	NbrChar:= NulPro_N(1, 1); (* reads the number of characters located in the reception queue associated with J3*)

Function	**NulPro_P**
ACTION	Configures a simple communication sequence on an Alto port.
SYNTAX	*Boolean NulPro_C(analog TerminalBlock, analog Com, analog Speed, analog Parity, analog StopBit, analog Data);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3 Speed: • 75Bauds=1, • 110Bauds=2, • 150Bauds=3, • 300Bauds=4, • 600Bauds=5, • 1200Bauds=6, • 2400Bauds=7, • 4800Bauds=8, • 9600Bauds=9, • 19200Bauds=10, • 38400Bauds=11, • 76800Bauds=12, (only on J2) • 115kbauds=13, (only on J2) • 200Bauds=14. (only on J2) Parity: • No parity=0, • Even=1, • Odd=2, • Forced to 0=3, • Forced to 1=4.. StopBit: • 1Stop=1, • 2Stop=2. Data: • 5Bits=5, • 6Bits=6, • 7Bits=7, • 8Bits=8.
RETURNED VALUE	FALSE: Not configured. TRUE: Correctly configured.
EXAMPLE	Status:= NulPro_P(1, 1, 9, 0, 1, 8); (* modifies the communication parameters on J3 The com parameters are now: 9600 bauds, no parity, 1 stop bit, 8 data bits *)

Function	**NulPro_C**
ACTION	Closes a simple communication sequence on an Alto port.
SYNTAX	*Boolean NulPro_C(analog TerminalBlock, analog Com);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3
RETURNED VALUE	FALSE: Not closed. TRUE: Correctly closed.
EXAMPLE	StatusPro_C(1, 1); (* Closes a simple communication sequence on J3*)

Read / Write bytes :

Three functions allows to write, read, and count not ASCII characters but directly bytes : it's interesting when we want to transmit or receive bytes with null bytes ; with Nulpro functions, Null byte is considered as a end character.

Function	**BinPro_S**
ACTION	Writes in the transmission queue on an Alto port
SYNTAX	*Boolean BinPro_S(analog TerminalBlock, analog Com, analog Variable, analog NbrByte);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3 Variable: 32 bits integer containing the bytes to be send NbrByte: [1..4] : writes n characters of the variable
RETURNED VALUE	FALSE: Not written. TRUE: Correctly written.
EXAMPLE	Status:= BinPro_S(1, 1, 26, 1); (* sends one byte (value 26) on the transmission queue associated with J3 *)

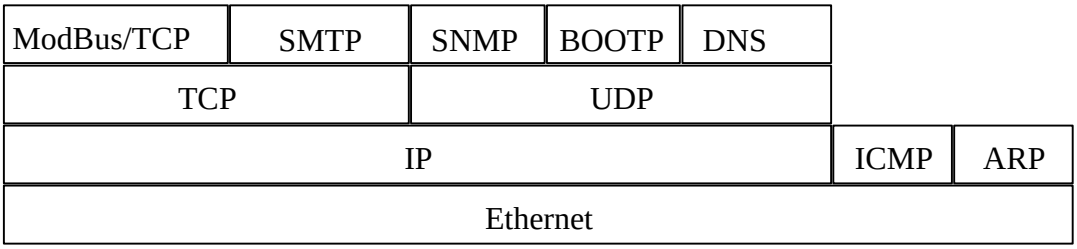
Function	**BinPro_R**
ACTION	Reads in the reception queue on an Alto port
SYNTAX	*analog BinPro_R(analog TerminalBlock, analog Com, analog NbrByte);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3 NbrByte: [1..4] : reads n bytes in the reception queue
RETURNED VALUE	Variable containing the read bytes
EXAMPLE	Variable:= BinPro_R(1, 1, 4); (* read 4 bytes in the reception queue associated with J3*)

Function	**BinPro_N**
ACTION	Reads the number of bytes located in the reception queue on an Alto port.
SYNTAX	*Analog BinPro_N(analog TerminalBlock, analog Com);*
PARAMETERS	TerminalBlock: [1] Com: 0= J2 ; 1 = J3
RETURNED VALUE	NbrBytes: number of bytes in the reception queue
EXAMPLE	NbrBytes:= BinPro_N(1, 1); (* reads the number of bytes located in the reception queue associated with J3*)

4.3. Ethernet protocols :

Once these network parameters have been correctly entered the Ethernet link will support:

- **IP** (Internet Protocol) : a set of industry protocol standards enabling communication in a heterogeneous environment. A protocol of the transport layer of the OSI model, it supplies a routable enterprise network management protocol as well as Internet access.
- **TCP** (Transmission Control Protocol) : Protocol for the Transport and Session layers of the OSI model. TCP verifies if the data have been correctly transmitted over the network and if they are in the appropriate order. This reliable connection oriented protocol also ensures the multiplexing of IP connections to the applications. It is a « connected » protocol.
- **UDP** (User Datagram Protocol) : UDP is a Datagram protocol without connection that enables applications to directly access a Datagram transmission service. UDP is used for applications which are satisfied by a « request/response » model type. The reply being used as a positive acknowledgement of reception.
- **ARP** (Address Resolution Protocol) :The link layer Protocol of the OSI model, ARP allows finding the physical address of a target machine by knowing its IP address latter.
- **ICMP** (Internet Control Message Protocol) : The interconnection protocol. ICMP allows gateways and equipment to exchange information related to abnormal conditions.



This protocol suite, over that of the Ethernet, determines the computer communication mode and inter-network connection procedures.

Note: the "ping" function (ICMP protocol) will allow you to verify the presence of an equipment on the Ethernet network.

Example : under DOS session, tape the line : "ping xxx.xxx.xxx.xxx" where xxx.xxx.xxx.xxx is the IP ADDRESS of the equipment you want to test

Identification of Alto on Ethernet :

Function	**AddrIP**
ACTION	Returns the IP address of Alto
SYNTAX	*message Data AddrIP();*
PARAMETERS	None
RETURNED VALUE	IP address of Alto
EXAMPLE	*AddressIP = AddrIP();*

4.3.1. Telnet protocol

This protocol consists in including bytes transmissions in IP frames and extract bytes from IP frames : it use the TCP connected mode.

Five C functions are available :

Function	**NulTcp_O**	
ACTION	Open a telnet client connection	
SYNTAX	*boolean NulTcp_O(message Addrip, analog Numport, message Login, message Password);*	
PARAMETERS	Addrip	IP address or symbolic address (DNS Address) of distant machine
	Numport	Number of port on distant machine

	Login	Identification name to connect to the port on distant machine
	Password	Password to connect to port on distant machine
RETURNED VALUE	FALSE: Not opened. TRUE: Correctly opened.	
EXAMPLE	*Status := NulTcp_O('192.168.239.210', 2100, 'root', 'ddd');* (*Open a telnet connection with a machine (IP address 192.168.239.210, on his port 2100 *)	

Function	**NulTcp_C**
ACTION	Close a telnet client connection
SYNTAX	*boolean NulTcp_C();*
PARAMETERS	None
RETURNED VALUE	FALSE: Not opened. TRUE: Correctly opened.
EXAMPLE	*Status := NulTcp_C();* (*Close a telnet connection with a machine *)

Function	**NulTcp_S**	
ACTION	Send a message in a telnet channel	
SYNTAX	*boolean NulTcp_S(message Msg, boolean Crlf);*	
PARAMETERS	Msg	Message to transmit
	Crlf	False : message send just so True : transmission with CRLF characters add to the message
RETURNED VALUE	FALSE: Writing not done. TRUE: Correct writing	
EXAMPLE	*Status*:= NulTcp_S(Message2, False); (* send variable Message2 in transmission queue of the telnet channel*)	

Function	**NulTcp_R**
ACTION	Read a message in the reception queue of a telnet channel
SYNTAX	*message NulTcp_R();*
PARAMETERS	None
RETURNED VALUE	Message : characters read in the reception queue (max 255)
EXAMPLE	Message1 := NulTcp_R(); (* all characters in the reception queue will be put in the Message1variable ; reception queue will be empty*)

Function	**NulTcp_N**
ACTION	Reads the number of characters located in the reception queue
SYNTAX	*analog NulTcp_N();*
PARAMETERS	None
RETURNED VALUE	Message : characters read in the reception queue (max 255)
EXAMPLE	Message1 := NulTcp_R(); (* all characters in the reception queue will be put in the Message1variable ; reception queue will be empty*)

4.3.2. Modbus/TCP protocol

This protocol consists of encapsulating the Modbus exchanges in the IP frames. It uses the TCP connected mode. It offers the same functionality as the « Serial ModBus slave » Ports on asynchronous links of the product. The differences with the Modbus protocol over asynchronous Port are as follows :

- No slave number (between 1 and 255), as the addressing is undertaken with the IP address
- Usage of the TCP connected mode. Alto can open four simultaneous channels with numerous masters and slaves on the network. Leroy Automation sets the limit, to 4 masters.
- No diffusion available.

4.3.2.1. Modbus/TCP slave protocol

Three C functions are available for using this protocol :

Function	**TCPMbS_O**
ACTION	Opens 4 Modbus/TCP Slave channel on an Ethernet Alto **WARNING : this function must be used only once**
SYNTAX	Boolean tcpmbs_O(analog TableNum, analog TableLength, analog Access);
PARAMETERS	TableNum: [1..7] TableLength: [1..4095] 4095 words (16 bits) Access: [0..2] 0: R/W, 1: Read, 2: Write
RETURNED VALUE	FALSE: Not opened. TRUE: Correctly opened.
EXAMPLE	Status = tcpmbs_O(1, 100, 0); (* declares a Slave Modbus/TCP protocol on an Ethernet Alto CPU. This channel is associated with a read/write accessible table of 100 words. This table is identified by the number 1. The slave number is Alto IP address. Note: Four masters can access this table "simultaneously" *)

Function	**TCPMbS_S**
ACTION	Monitors the presence of 1 to 10 masters on the ModBus/TCP slave channel
SYNTAX	Boolean tcpmbs_S(message IPAddress, analog TimeOut);
PARAMETERS	IPAddress : message type variable containing the IP address TimeOut : [0..FFFF] TimeOut in milliseconds
RETURNED VALUE	FALSE : Master absent. TRUE : Master present.
EXAMPLE	Status = tcpmbs_S(address_M, 5000); (* monitor the presence of the master whose IP address is contained in the « address_M » message type variable. If this master is absent for at least 5 seconds, the status will switch from TRUE to FALSE *)

Function	**TCPMbS_T**
ACTION	Set the timeout value for closing the TCP connection after any more activity from the modbus/TCP master ; the default value is 300 seconds or 5 minutes
SYNTAX	Boolean tcpmbs_T(analog TimeOut);
PARAMETERS	TimeOut : [0..FFFF] TimeOut in seconds
RETURNED VALUE	FALSE : timeout not set. TRUE : timeout set
EXAMPLE	Status = tcpmbs_S(10); (* If the master don't transmit modbus request during 10 seconds, the modbus/TCP slave close the TCP connection with the master*)

4.3.2.2. Modbus/TCP master protocol

Two C functions are available for using this protocol :

Function	**TCPMbM_O**
ACTION	Opens one Modbus/TCP Master channel on an Ethernet Alto **WARNING : only three Modbus/TCP master channel can be opened simultaneously.**
SYNTAX	Boolean tcpmbs_O(analog LineNum, analog TableNum, analog TableLength, analog Access);
PARAMETERS	LineNum: [1..3] TableNum: [1..7] TableLength: [1..4095] 4095 words (16 bits) Access: [0..2] 0: R/W, 1: Read, 2: Write
RETURNED VALUE	FALSE: Not opened. TRUE: Correctly opened.
EXAMPLE	Status = tcpmbm_o(2, 1, 100, 0); (* Open a line number 2 Master Modbus/TCP protocol on an Ethernet Alto CPU. This channel is associated with a read/write accessible table of 100 words. This table is identified by the number 1.

Function	**TCPMbM_T**
ACTION	Sends a Master modbus/TCP frame to Alto Ethernet port and reads the communication status
SYNTAX	analog tcpmbm_T(analog LineNum, message SlaveNum, analog FunctionCode, analog SlaveAddress, analog Length, analog DataAddress);
PARAMETERS	LineNum: [1..3] SlaveNum: IP address or DNS Address of the modbus/TCP slave FunctionCode: • 1 or 2: read n bits • 3 or 4: read n words • 5: write 1 bit • 6: write 1 word • 15: write n bits • 16: write n words SlaveAddress: [0..FFFFh] Length: [1..128] DataAddress: [0.. FFFFh] Com - 1"]
RETURNED VALUE	see appendix for the list of the code of returned values
EXAMPLE	Status:= TCPMBM_T(2, "192.168.2.4", 3, 5, 1, 4); (* read 1 word at address 5 on slave , "192.168.2.4", store at address 4 in table associated to line 2*)

Function	TCPMbM_S
ACTION	Sends a Master modbus/TCP IO scanning frame to Alto Ethernet port and reads the communication status
SYNTAX	analog tcpmbm_S(analog LineNum, message SlaveNum, analog SlaveAddressRead, analog LengthRead, analog DataAddressRead, analog SlaveAddressWrite, analog LengthWrite, analog DataAddressWrite);
PARAMETERS	LineNum: [1..3] SlaveNum: IP address or DNS Address of the modbus/TCP slave SlaveAddressRead: [0..FFFFh] LengthRead: [1..128] DataAddressRead: [0.. 4096] : address of data in Alto table SlaveAddressWrite: [0..FFFFh] LengthWrite: [1..128] DataAddressWrite: [0.. 4096] : address of data in Alto table
RETURNED VALUE	see appendix for the list of the code of returned values
EXAMPLE	Status:= TCPMBM_S(2, "192.168.2.4", 5, 1, 4, 20, 2, 30); (* request on line 2 to "192.168.2.4" modbus/TCP slave: - read 1 word at address 5 on slave , store at address 4 in local Alto table - write 2 words at address 20 on slave , words are at address 30 in local Alto table *)

Function	TCPMbM_b
ACTION	Set the timeout value for closing the TCP connection with the slave after any more activity from the modbus/TCP master ; the default value is 300 seconds or 5 minutes
SYNTAX	Boolean tcpmbm_b(analog LineNum, analog TimeOut);
PARAMETERS	TimeOut : [0..FFFF] TimeOut in seconds 0 : close the TCP connection after each modbus/TCP exchange >0 : close after the timeout value set
RETURNED VALUE	FALSE : timeout not set. TRUE : timeout set
EXAMPLE	Status = tcpmbm_b(1, 10); (* If the master don't transmit new modbus request during 10 seconds, the modbus/TCP master close the TCP connection with the slave*)

4.3.3. SNMP Protocol

- **SNMP**: *Simple Network Management Protocol*: Standard protocol used on Internet for the administration of hosts, routers and other devices on the network.

Alto is an SNMP agent. Alto SNMP variables are read/write accessible for a SNMP manager.

SNMP is the protocol of the OSI model application layer that depends on the UDP protocol. The port number, which identifies the SNMP application protocol, is 161.

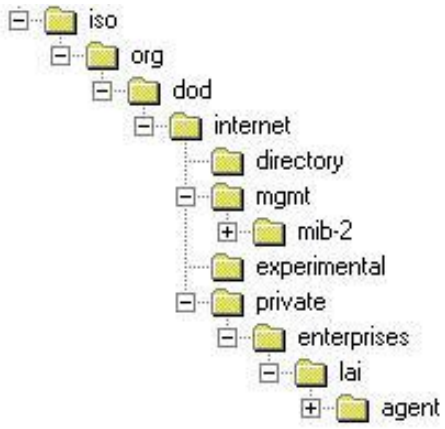
The variables that are managed by the SNMP protocol belong to a unique structure called MIB.

The operations supported by Alto are :

- Get : allows a manager to extract the value of an object (SNMP variable : OID) in an agent (Alto).
- GetNext : allows a manager to extract the next value of an object in an agent (Alto).
- Set : allows a manager to modify the value of an object in an agent (Alto).

SNMP Alto variables are accessible in read / write mode by the SNMP manager : see paragraph 3.1 for the IP manager parameters.

The SNMP protocol enables access to Alto variables defined by ISaGRAF in the MIB (Management Information Base). The MIB is a database defined formally in the ASN1 (Abstract Syntax Notation 1) language whose tree structure could be the following:



The « input » variable name is iso. org.dod.internet.private.enterprises.lai.agent.entrées and its identifier is written as *1.3.6.1.4.1.4273.10.1.0* (0 being the instance of the variable with this name).

4273 is the identifier of LAI PLCs, 10 the identifier agent.

4.3.3.1. MIB II

MIB II is the standard MIB : all agents owned it. The MIB II ID is : 1.3.6.1.2.1… ; MIB II is composed by 10 groups : Alto has only the first group implemented : group System(1)

System content basic information to recognize the agent :

Note : the notation is : name-object(position, type, access).

- **sysDescr**(1, octet string, read-only): agent description.
- **SysObjectID**(2, object identifier, read-only) : Identification : Pointing to the branch of device (1.3.6.1.4.1.4273.10 for LAIand agent 10)
- **SysUpTime**(3, Time Ticks, read-only) : Time passed since initialisation.
- **SysContact**(4, octet string, read-write): Contact.
- **SysName**(5, octet string, read-write) : Node name.
- **SysLocation**(6, octet string, read-write): Location of device.
- **SysServices**(7, integer, read-only): Service level available (between 1 and 7 : OSI layers).

4.3.3.2. MIB LAI

Leroy Automation obtained from « Internet Assigned Numbers Authority – MIB » a branch identified in the Enterprises branch : « LAI », number (4273).

In LAI (4273) branch, a sub branch is defined with the agent number : the parameter « Num_Agent_SNMP » is in wiring diagram of UCR board : : see paragraph 3.1 and in this sub branch, are defined the ISaGRAF variables.

The **SnmpVA_C(), SnmpVM_C()** functions enable respectively creating the following SNMP variable types:

- 32 bit signed integers
- messages as character strings whose length is defined at the time of the variable creation

Function	**SnmpVA_C**
ACTION	Defines the SNMP order, in the 1.3.6.1.4.1.4273.agent branch of an analog variable in the ISaGRAF dictionary
SYNTAX	Boolean Status:= SnmpVA_C(analog Flag, analog OID, analog AddrAnaVar);
PARAMETERS	Flag: [1..4] : not used, set to 0 by default OID: [1..32767] order of the SNMP variable used to identify it in the branch AddrAnaVar: [0..FFFF] dictionary network address (hex) of the analog variable
RETURNED VALUE	FALSE: Operation not carried out. TRUE: Operation correctly carried out.
EXAMPLE	Status = SnmpVA_C(0, 4, 16#20); (* the analog variable with address 16#20 in the dictionary can be accessed by SNMP; its identifier is the following 1.3.6.1.4.1.4273.2.**4**.0*)

Function	**SnmpVM_C**
ACTION	Defines the SNMP order, in the 1.3.6.1.4.1.4273.agent branch of a message variable in the ISaGRAF dictionary
SYNTAX	Boolean Status:= SnmpVM_C(analog Flag, analog OID, analog AddrMesVar);
PARAMETERS	Flag: [1..4] : not used, set to 0 by default OID: [1..32767] order of the SNMP variable used to identify it in the 1.3.6.1.4.1.4273 branch AddrMesVar: [0..FFFF] dictionary network address (hex) of the message variable
RETURNED VALUE	FALSE: Operation not carried out. TRUE: Operation correctly carried out.
EXAMPLE	Status = SnmpVM_C (0, 14, 16#30); (* the message variable with address 16#30 in the dictionary can be accessed by SNMP; its identifier is the following 1.3.6.1.4.1.4273.2.**14**.0*)

Example : see project « Alsnmp »

Adress of SNMP variable « entier4 » :

iso.org.dod.internet.private.enterprises.lai.NumeroAgentSNMP.entier4

its identification, named OID, is *1.3.6.1.4.1.4273.2.4.0* (0 is the instance of variable).

MIB manager additions with Alto variables defined in language ASN1 :

```
LAI DEFINITIONS ::= BEGIN
        IMPORTS
              enterprises
                    FROM RFC1155-SMI
           OBJECT-TYPE
              FROM RFC-1212;
```

```
lai      OBJECT IDENTIFIER ::= { enterprises 4273 }
agent  OBJECT IDENTIFIER ::= { lai 2 }

        entier4       OBJECT-TYPE
            SYNTAX  INTEGER
            ACCESS  read-only
            STATUS  mandatory
            DESCRIPTION
                "test variable entière avec alto Isagraf"
            ::= { agent 4 }

        message4   OBJECT-TYPE
            SYNTAX  OCTET STRING (SIZE (0..255))
            ACCESS  read-only
            STATUS  mandatory
            DESCRIPTION
                "test variable message avec alto Isagraf"
            ::= { agent 14 }

END
```

4.3.3.3. Traps SNMP V1 :

Standard Trap : A coldstart trap is transmitted automatically to the manager at each power on of device.

Spécifics Traps : It's possible to transmit to the Manager des spécifics traps with two C functions :

- **Trapint** : to transmit a code and a value.
- **Trapstr** : to transmit a code and a message.

Function	**Trapint**
ACTION	Transmit a trap with a value
SYNTAX	*boolean Trapint (analog Code, analog Value, analog OID);*
PARAMETERS	Code : value of trap code. Value : analog value to transmit OID : [1..32767] order of the SNMP variable used to identify it in the 1.3.6.1.4.1.4273.Num_Agent branch.
RETURNED VALUE	FALSE: Operation not carried out. TRUE: Operation correctly carried out.
EXAMPLE	*Status := Trapint(3, Entier, 4);* *OID associated is : 1.3.6.1.4.1.4273.2.4.0*

Function	**Trapstr**
ACTION	Transmit a trap with a message
SYNTAX	*boolean Trapstr (analog Code, message Mess, analog OID);*
PARAMETERS	Code : value of trap code Mess : Message to transmit OID : [1..32767]] order of the SNMP variable used to identify it in the 1.3.6.1.4.1.4273.Num_Agent branch
RETURNED VALUE	FALSE: Operation not carried out. TRUE: Operation correctly carried out..
EXAMPLE	Status := Trapstr(7, Mess*, 14*); *OID associated is : 1.3.6.1.4.1.4273.2.14.0*

Example : see project « Alsnmp »

4.3.4. Sending electronic mail

- **SMTP** : *Simple Mail Transfer Protocol* : Standard internet protocol for sending electronic mail

Number of mail transmission attempts : unlimited as long as Alto has not been able to connect to the SMTP server. A mail transmission attempt is aborted if the server refuses to send the mail on.

The object of electronic mail sent from Alto is the following: « Alto Message number X » where X=Alto serial number.

The Email_I() and Email_S() functions respectively enable the address initialisation of the server of mail being sent and to send an electronic mail.

Function	**EMail_I**
ACTION	Initialize the SMTP server address. This address may be an IP address or a server name (ex : smtp.anydomain) **CAUTION : The mail server must be unique ; it is therefore prohibited initialize it several times.**
SYNTAX	Boolean Status := EMail_I(Message Address);
PARAMETERS	Address: message type variable containing the SMTP server address
RETURNED VALUE	FALSE : Operation failed. TRUE : Operation succeeded.
EXAMPLE	Status = EMail_I(Address_Server); (* The address of the outgoing mail server (SMTP) is initialized. It's value is that contained in the message type variable Address_Server *)

Function	**EMail_S**
ACTION	Send an electronic mail via the SMTP server.
SYNTAX	Boolean Status := EMail_S(Message TO, Message FROM, Message Content);
PARAMETERS	TO : message type variable containing the destination address of the e-mail. This address can be in the IP or literal form (ex : symbolic_adress@anydomain) FROM : message type variable containing the address of the e-mail sender. This address can be in the IP or literal form (ex : symbolic_adress@anydomain) Content: message type variable containing the body of the e-mail.
RETURNED VALUE	FALSE : Operation failed. TRUE : Operation succeeded.
EXAMPLE	Status = EMail_S(Adresse_Destination, Adresse_Sender, Mail); (* The message contained in the Mail variable is sent from the Sender Address to the Destination Address *)

Two other functions allows to verify the SMTP server presence :

Function	**US_PING**
ACTION	Initialize the function of checking the SMTP server presence
SYNTAX	Boolean Status := US_PING(Message Address, analog period, analog delay);
PARAMETERS	Address: message type variable containing the SMTP server address Period: period of request in minute Delay : timeout authorized for the server response
RETURNED VALUE	FALSE : Operation failed. TRUE : Operation succeeded.

Function	**PING_V**
ACTION	Refresh the function of checking the SMTP server presence
SYNTAX	Analog Status := PING_V();
RETURNED VALUE	256 : server present ; 1280 : server absent

5. Input/Output Boards

Each of the following I/O modules has a bit named "presence" : it can be used by the application to know if module is OK or not.

For each input/output board, a **data sheet** (Help menu) is available in the ISaGRAF workbench.

AES Module	Input « presence »	Digital input	Digital output	Analog input	Analog output
Bx01	**1**	**24**	**8**		
Bx02	**1**	**4**	**4**	**8 (2)**	**2 (2)**
Bx03	**1**	**16**	**8**		
Bx04	**1**	**8**	**8**		
Bx06	**1**	**16**	**8**		
Bx07	**1**	**8**	**8**		
Bx08	**1**	**4**	**4**	**10 (3)**	**2 (3)**
Bx09	**1**	**4**	**4**	**8 (3)**	
Bx10	**1**	**4**	**4**	**4 (3)**	
Bx11	**1**	**4**	**4**	**4 (3)**	**2 (3)**
Bx12	**1**	**32**			
Bx13	**1**	**24 (1)**	**4**		
Bx14	**1**	**16 (1)**	**4**		
Bx15	**1**	**8 (1)**	**4**		
Bx16	**1**	**24 (1)**			

(1) : particulars parameters of Bx13 / Bx14 / Bx15 and Bx16 boards:

Those boards are equipped with an adjustable comparison device used to check the wiring of sensors by connecting a network of 2 resistors to them : safety inputs These resistor networks are of 2 types: the **serial arrangement** (i.e. the 2 serial resistors) and the **parallel arrangement** (i.e. the 2 parallel resistors). The serial resistor is always present. In the parallel arrangement, the sensor is mounted in series with Rp which it eliminates by opening. In the serial arrangement, the sensor is mounted in parallel with Rp which it eliminates by closing.

In order to preserve the general nature of parameter setting, ISaGRAF can indicate the equivalent resistance of the resistor network when the sensor is **normally open (Rcno)** and when the sensor is **normally closed (Rcnf**). Resistance values are given **in OHMS**.

Parallel Arrangement	Serial Arrangement
Rcnf = Rs//Rp + Rline	Rcnf = Rs + Rline
Rcno = Rs + Rline	Rcno = Rs+Rp + Rline

CAUTION: Parameter setting is unique for the resistance values of **all safety boards** and is therefore the same for all the channels of a single board.

	min value (by defaut)	max value
RCNO	2000 Ω	6600 Ω
RCNF	1000 Ω	6/7 x RCNO – 700 Ω

See specific board documentation, for wiring diagram and calculation of RCNO et RCNF.

- "Masque" : 32-bit mask for the wiring check of the 32 inputs. The wiring check is active at input n if the bit of order n is set to 1. By default, the 32 bits of the mask are set to 1(« FFFFFF »)

  Example : input 2 and input 16 in normal mode : mask must be : « FF7FFD»that correspond in binary to : 1111 1111 0111 1111 1111 1101

For each input, the status bit (« inputs ») and alarm bit (« states »)encode 4 possible states :

Description	Bit « input »	Bit « states »
Sensor normally open	0	0
Sensor normally closed	1	0
Input not connected or short-circuit at 0V	0	1
Short-circuit at +V	1	1

(2) : particulars parameters of Bx02 board :

This board has extra parameters to :

- 8 parameters « Type voie » to select the analog input type
- 2 parameters « Type voie » to select the analog output type

Caution : see wiring documentation (P ALT DOC 005 F) to parameter switches of this board.

Type	unit	Input range
00	mV	-10V/+10V
01	0.1mV	-1V/+1V
02	0.01mV	-100mV/+100mV
08	µA	-20mA/+20mA
09	µA	+4mA/20mA
10	0.1°C	PT100
11	0.1°C	PT1000
12	0.1°C	NI1000
20	°C	Thermocouple B
21	°C	Thermocouple C
22	°C	Thermocouple E
23	°C	Thermocouple J
24	°C	Thermocouple K
25	°C	Thermocouple M
26	°C	Thermocouple N
27	°C	Thermocouple P
28	°C	Thermocouple R
29	°C	Thermocouple S
2A	°C	Thermocouple T

Type	unit	Output range
00	mV	0V/+10V
01	µA	0mA/+20mA
03	µA	4mA/20mA

(3) : particulars parameters of Bx08/Bx09/Bx10/Bx11 boards :

Those devices have extra parameters :

- 10 parameters « Type voie » to select the analog input type
- 2 parameters « Type voie » to select the analog output type

Caution : see the wiring documentation (P ALT DOC 006 F) to parameter switches.

Type	unit	Input range
00	mV	-10V/+10V
01	mV	-5V/+5V
02	0.1mV	-1V/+1V
03	µA	-20mA/+20mA
07	µA	+4mA/20mA

Type	Unit	Output range
01	MV	0V/+10V
00	µA	0mA/+20mA
02	µA	4mA/20mA

6. Alto monitoring and diagnostic

6.1. Errors Transferred to the Workbench

Two types of errors can be transferred to the workbench:

- **errors encoded by CJ**: text explaining the error with a number between 0 and 99. These are listed in the workbench user's guide.
- **errors encoded by LAI**: number between 100 and 255.

The following errors are encoded by LAI:

- 100: type of Flash (not AMD512K Bottom). TIC size is limited in this case to 128 KB.
- 101 to 105: error when saving the TIC application to FLASH.
    - 101: read error in FLASH.
    - 102: write error in FLASH: data written in spaces reserved for an application.
    - 103: FLASH sector access error.
    - 104: FLASH sector erase error.
    - 105: read error: data in spaces reserved for an application.
- 110 to 112: error when reading the TIC application in FLASH.
    - 110: FLASH type read error
    - 111: memory allocation error: spaces reserved for an application.
    - 112: FLASH sector access error.
- 113: checksum error in TIC application read in FLASH.
- 120 and 121: Alto dynamic memory damaged, Alto automatic reboot.
- 130: illegal board added in the case of an Alto80 (order[1..3]).
- 140 to 143: save error in non volatile variables:
    - 140: error: if 1 or more non volatile variables are checked, there must be at least one of each type.
    - 141: Alto memory allocation error.
    - 142: backed-up memory full.
    - 143: backed-up memory read error.
- 150: backed-up clock initialization error.
- 151: backed-up clock write error.
- 160: R2232 control signal read error.
- 161: RS232 control signal write error.
- 170: communication parameter setting error on com1 of terminal block 1: params_com parameter on the cpu3xx board.
- 200 : calibration configuration error in analog AES board.
- 201 : configuration error in analog output AES board.
- 202 [i]: AES board number « i » not recognized.
- 205 [1]: CF failure
- 205 [2]: CF opening failure
- 205 [3]: server of CF data start failure
- 205 [4]: CF formatting failure
- 205 [5]: CF : record opening failure
- 205 [6]: CF : record closing failure
- 205 [7]: CF closing failure
- 205 [10]: CF : record writing failure

6.2. Switching Alto to Parameter Setting Mode

For an Alto ISaGRAF, switching to parameter setting mode means running only the ISaGRAF kernel with no TIC (Target Independent Code) application downloaded, and restablishing the console link on J2.

This mode, called **PRM**, is symbolized by LED RUN light on orange on the CPU.

To switch to PRM mode from Windows95/98/NT/2000/XP :

- switch off Alto,
- connect Alto (J2) and the PC (com1 or com2) using an RS232 cable,
- run the "**SSTBsetup.exe**" program included on the "Libraries" diskette under Windows95. This program waits until it recognizes an Alto.
- select the communication channel on the PC [1..4],
- select "**Run Prm function**",
- switch on Alto,
- in its initialization sequence, Alto switches to PRM mode and its RUN LED lights up without flashing. The PC displays the message "**Alto set to PRM by default**"
- Alto ISaGRAF is now in PRM mode.

6.3. Alto ISaGRAF LEDs : CPU and I/O boards

6.3.1. CPU LEDs

- **red "Col" LED**
  - ◊ OFF if operation is correct.
  - ◊ ON if lot of collision on Ethernet network : network overloaded : solution reduce its traffic.
- **green "Lnk" LED**
  - ◊ ON : if Ethernet connection correct.
  - ◊ OFF : incorrect Ethernet connection : verify Ethernet wires.
- **green "Com1" LED**
  - ◊ ON if J2 activate : console link (params_com=0) or programmation in modbus protocol or byte transmission reception.
  - ◊ OFF otherwise
- **green "Com2" LED**
  - ◊ ON if J3 activate : programmation in modbus protocol or byte transmission reception.
  - ◊ OFF otherwise
- **"Ext" LED**
  - **green** :
    - ◊ flashes green if communication is OK between main UCR and extension UCR.
  - **orange** : one of extension UCR B510 board is not responding to main UCR
  - **red** : any of extension UCR B510 board is responding to main UCR
- **"Mode" LED**
  - **green** :
    - ◊ OFF if incorrect.
    - ◊ ON if program read in Flash memory is correct
  - **red** : AES board under main UCR not recognized
- **"Run" LED**
  - **green** :
    - ◊ flashes green slowly (1 s) if the TIC application (ISaGRAF) is run correctly :
    - ◊ flashes green rapidly (1/10s) if the equipment is in PRM mode or if the application is stopped by ISaGRAF --> the kernel is active but does not run the TIC application.
  - **orange** fixe : Alto is in prm mode

6.3.2. AES Leds

The state of an Alto input and output is identifiable with 32 green leds in front face of UCR board. Correspondence of those LEDs is describe afterwards for each AES board. The signification of a led ON or OFF is describe depending of input or output type.

Type	Notation	Led OFF	Led ON	Led flashing
Digital input	Ei ou Eti (*)	Sensor open	Sensor closed	
Safety input	Ei (*)	Sensor open	Sensor closed	Line default
Digital output	Si or Sti or Ri (relay output)	Output open	Output closed	
Analog input	EAi	parametrable threshold not exceed	Exceeding of parametrable threshold	
Analog output	SAi		toujours	

(*) Nota : i est l'indice sur les entrées et les sorties

AES Bx01

E1	E2	E3	E4	E5	E6	E7	E8			E9	E10	E11	E12	E13	E14	E15	E16
E17	E18	E19	E20	E21	E22	E23	E24			S1	S2	S3	S4	S5	S6	S7	S8

AES Bx02

ET1	ET2	ET3	ET4	ST1	ST2	ST3	ST4			EA1			EA2			EA3	
SA1	SA2			EA4			EA5			EA6			EA7			EA8	

AES Bx03

E1	E2	E3	E4	E5	E6	E7	E8			R1		R2		R3		R4	
E9	E10	E11	E12	E13	E14	E15	E16			R5		R6		R7		R8	

AES Bx04

E1	E2	E3	E4	E5	E6	E7	E8			R1		R2		R3		R4	
										R5		R6		R7		R8	

AES Bx06

E1	E2	E3	E4	E5	E6	E7	E8			E9	E10	E11	E12	E13	E14	E15	E16
										S1	S2	S3	S4	S5	S6	S7	S8

AES Bx07

E1	E2	E3	E4	E5	E6	E7	E8										
										S1	S2	S3	S4	S5	S6	S7	S8

AES Bx08

ET1	ET2	ET3	ET4	ST1	ST2	ST3	ST4			EA1		EA2		EA3		EA4	
	EA5		EA6		EA7		EA8			EA9		EA 10				SA1	SA2

AES Bx09

ET1	ET2	ET3	ET4	ST1	ST2	ST3	ST4			EA1		EA2		EA3		EA4	
	EA5		EA6		EA7		EA8										

AES Bx10

ET1	ET2	ET3	ET4	ST1	ST2	ST3	ST4			EA1		EA2		EA3		EA4	

AES Bx11

ET1	ET2	ET3	ET4	ST1	ST2	ST3	ST4			EA1		EA2		EA3		EA4	
																SA1	SA2

AES Bx12

E1	E2	E3	E4	E5	E6	E7	E8			E9	E10	E11	E12	E13	E14	E15	E16
E17	E18	E19	E20	E21	E22	E23	E24			E25	E26	E27	E28	E29	E30	E31	E32

AES Bx13

E1	E2	E3	E4	E5	E6	E7	E8			R1		R2		R3		R4	
E9	E10	E11	E12	E13	E14	E15	E16			E17	E18	E19	E20	E21	E22	E23	E24

AES Bx14

E1	E2	E3	E4	E5	E6	E7	E8			R1		R2		R3		R4	
E9	E10	E11	E12	E13	E14	E15	E16										

AES Bx15

E1	E2	E3	E4	E5	E6	E7	E8			R1		R2		R3		R4	

AES Bx16

E1	E2	E3	E4	E5	E6	E7	E8										
E9	E10	E11	E12	E13	E14	E15	E16			E17	E18	E19	E20	E21	E22	E23	E24

APPENDIX : Modbus/Jbus asynchronous and Modbus/TCP return codes

Decimal	Hexa	Comment
0	**0**	**exchange in progress**
256	**100**	**exchange correct**
769	301	exception code: unknown function
770	302	exception code: wrong address
771	303	exception code: invalid data
772	304	exception code: slave busy
773	305	exception code: acknowledge
774	306	exception code: no acknowledge
775	307	exception code: write error
776	308	exception code: zone overlap
896	380	connection error
897	381	connection warning
1024	400	wrong slave number
1025	401	wrong function code
1026	402	wrong length
1027	403	wrong sub-function code
1028	**404**	**wrong address**
1029	405	wrong data
1030	406	wrong frame length
1280	**500**	**no slave**
1281	501	CRC error
4096	1000	in transmission mode: frame in progress
4097	1001	in transmission mode: broadcast error
4099	1004	in transmission mode: wrong length
4100	1005	in transmission mode: offset error
4101	1006	in transmission mode: function error
4102	1007	in transmission mode: sub-function error
4103	1008	in transmission mode: sub-function data error
4104	1009	in transmission mode: storage error

The most frequently encountered communication status error codes are shown in **bold type**.

TABLE OF FIGURES

C FUNCTIONS INDEX