leroy AUTOMATION

DEVELOPMENT TOOLS
FOR LT80 & LT160 PLCs
ISaGRAF WORKBENCH

LT ISaGRAF

USER'S MANUAL

P DOC ISA 002 E - V1.8

SCOPE OF SUPPLY

The **LT ISaGRAF Kit** includes:

- This LT ISaGRAF User's Manual.
- 1 RS232 cable from the PC to the LT.
- 1 "LT ISaGRAF Libraries" floppy disk.

The LT (80 or 160) implementation manual is supplied with the LT.

This documentation describes the functions of:

- **embedded software (kernel):**
  - no ethernet version : 1.41
  - Ethernet version : 1.22
- **LT ISaGRAF libraries: version 1.8**

TECHNICAL SUPPORT:

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

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

I. General Overview

The parameters of the **LT (LT80 or LT160)** made by LEROY Automatique Industrielle can be programmed using the ISaGRAF workbench. In this case, the LT is called **LT ISaGRAF**. ISaGRAF is used on LTs integrating 386Ex processors.

This documentation describes how to start up the ISaGRAF workbench on the LT 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 the LT:

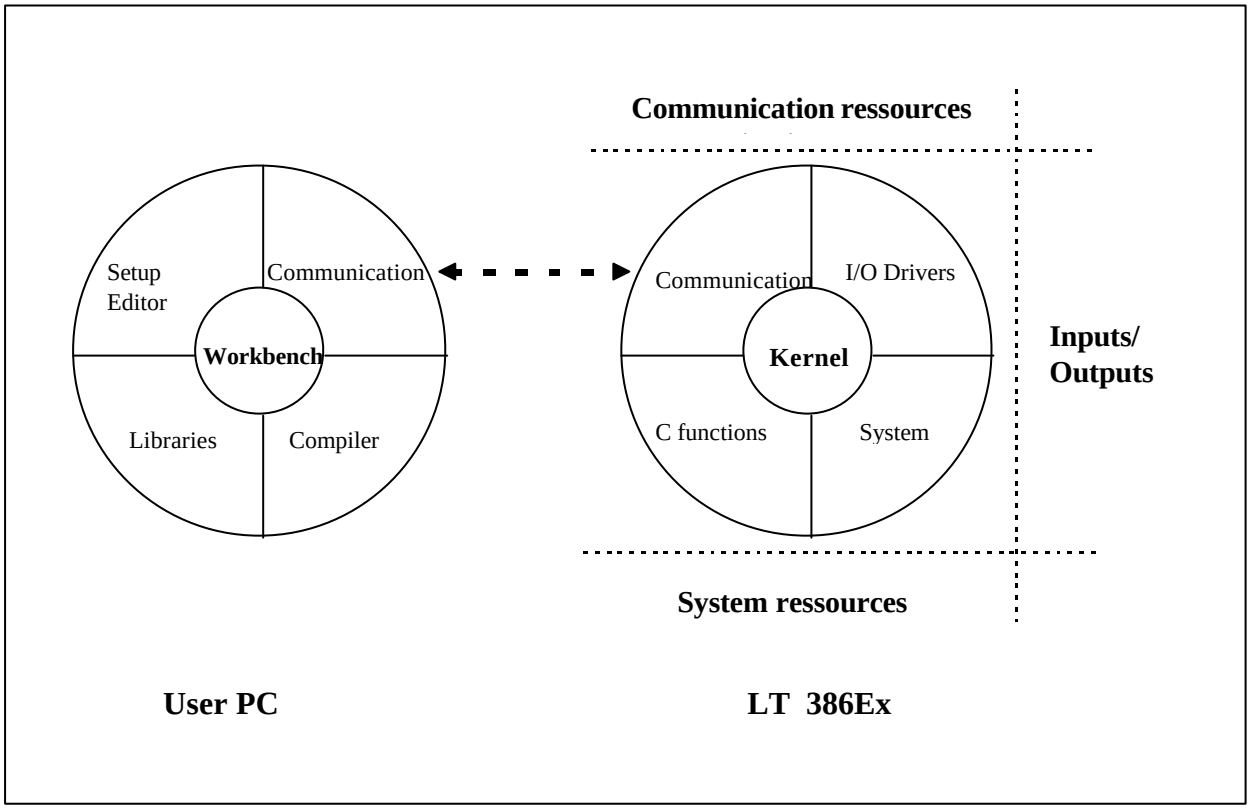


Figure 1: ISaGRAF Architecture on the LT

The **LT 386Ex target** supports the **ISaGRAF kernel** and provides access to the following resources:

- LT 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 the LT. This includes the following functions:

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

I.1. Hardware Resources of the LT Equipped with the 386Ex Processor

The LT 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 the LT and the related functions.

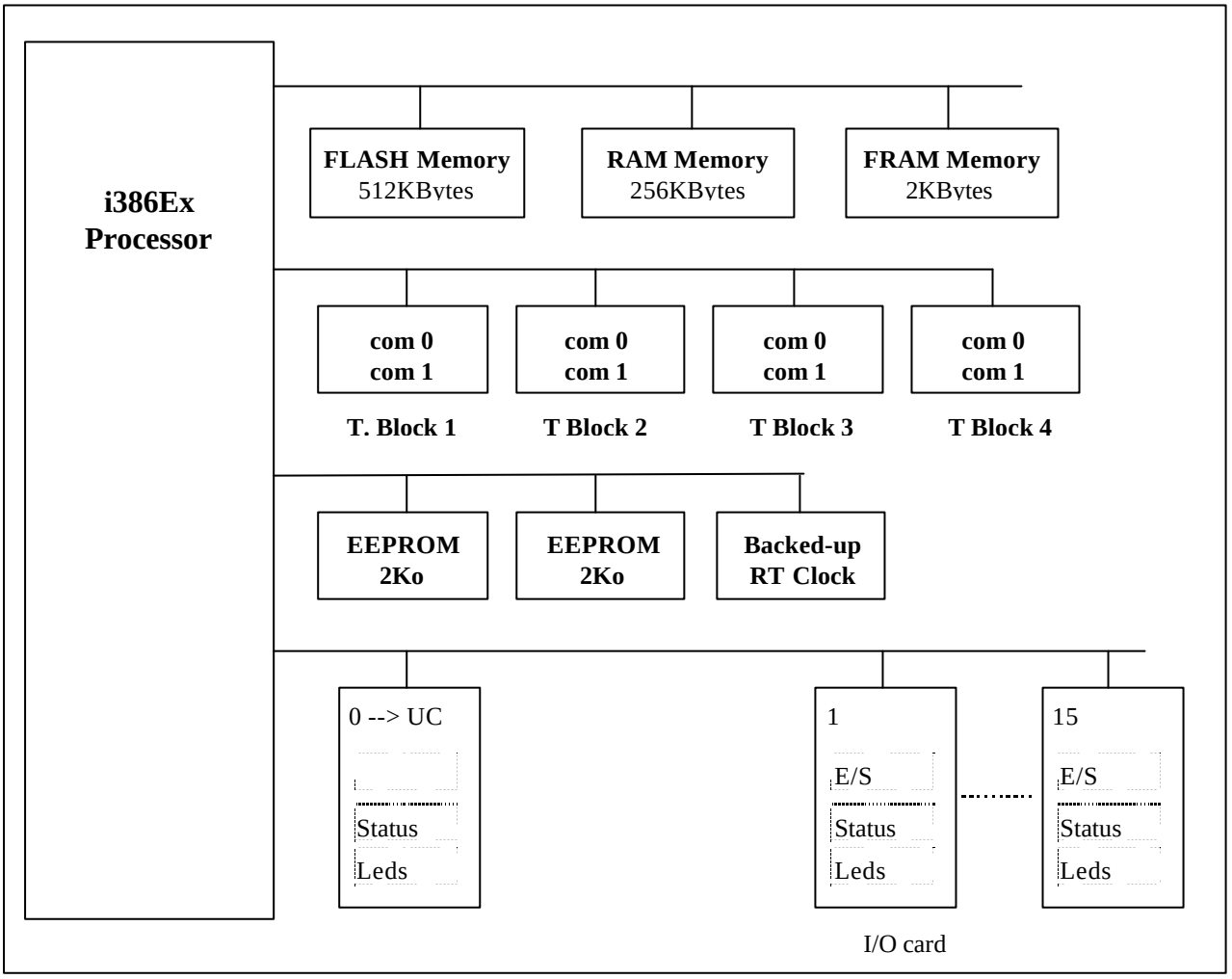


Figure 2: LT160 Hardware Resources

The **LT80** has the following constraints:

- 128 Kbytes of RAM,
- A single communication terminal block (com 1 and com 0),
- 3 input/output board slots,
- A single EEPROM.

I.2. Cycle of Processes Performed

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

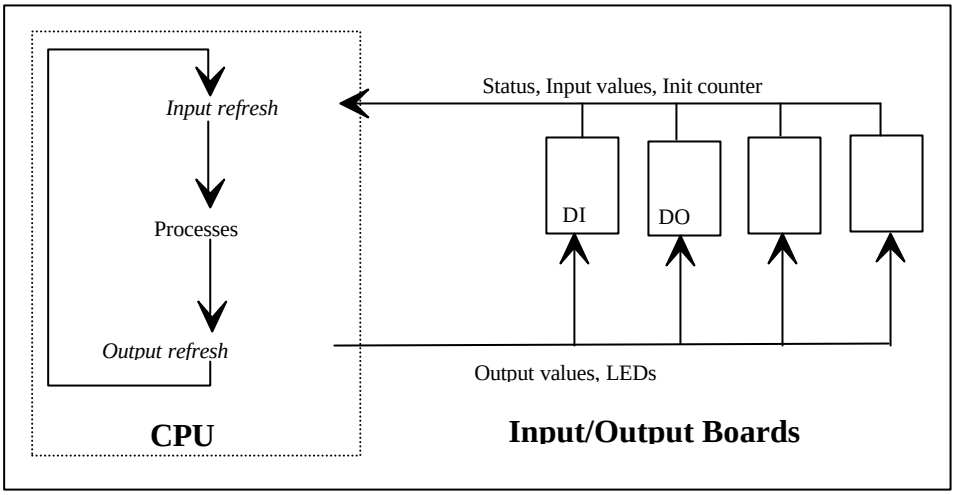


Figure 3: LT ISaGRAF Processing Cycle

II. Programming Environment

II.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 the LT target.

Start up the ISaGRAF workbench.

II.2. Integrating LT-specific Functions

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

These libraries are supplied on the "LT 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.

Insert the "LT ISaGRAF Libraries" diskette into drive a. This diskette contains objects of the following types:

- Projects,
- IO configurations
- IO Complex equipements,
- IO boards,
- Functions,
- Function blocks,
- C functions,
- C function blocks.

Access to the "**Archive**" list is possible using the "**Archive**" option in the "**Tools**" menu.

For each type of object, select the required option in the "**File**" menu. The "**Archive**" list contains all the items to be restored. Restore each item using the "**Restore**" button.

Once all the objects have been integrated into the workbench, all the C functions specific to the LT 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 the LT

II.3. First Program

A minimum program "which does not do anything" can be made as follows:

- create a new project by selecting: "File", "New Project" in the "Project Manager" window,
- select "I/O Configuration" LT or set "cpu3xx" on slot 0 of the input/output wiring,
- generate the application,
- download it to the LT.

This minimum program does not manage any LT input/output resources and does not perform any processing.

Important: the **console link** (or workbench link) is located by default on **com1 of Terminal Block 1**. It can be located on any other communication port except FIP (see "Serial Communication Management).

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

II.4. Hardware Configuration

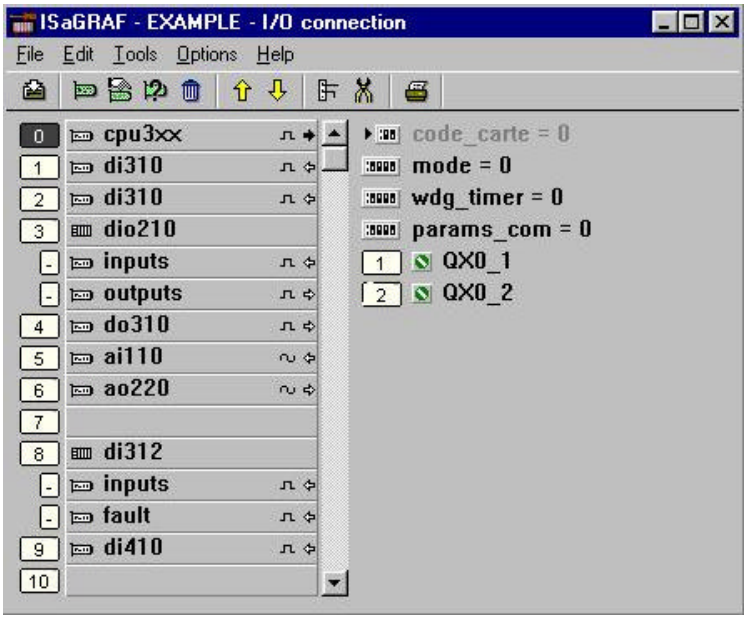
LT hardware configuration is achieved by selecting **"IO connection"** in the **"Project"** menu. This window proposes 256 input/output board slots. Only the first 16 can be used to program an LT. In the case of an LT80, only the first 4 slots can be used: 1 CPU + 3 I/O boards.

The first slot is reserved for the CPU: CPU3xx or CPUeth. Only two types of CPU are used for all LT 386 devices.

The power supply or PSD (PSD300 or PSD331) is not configured graphically. It is declared automatically by the kernel.

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

The LT rack is shown vertically and the input/output boards are numbered from 0 to 15. The following is an configuration example showing the corresponding items on the LT:



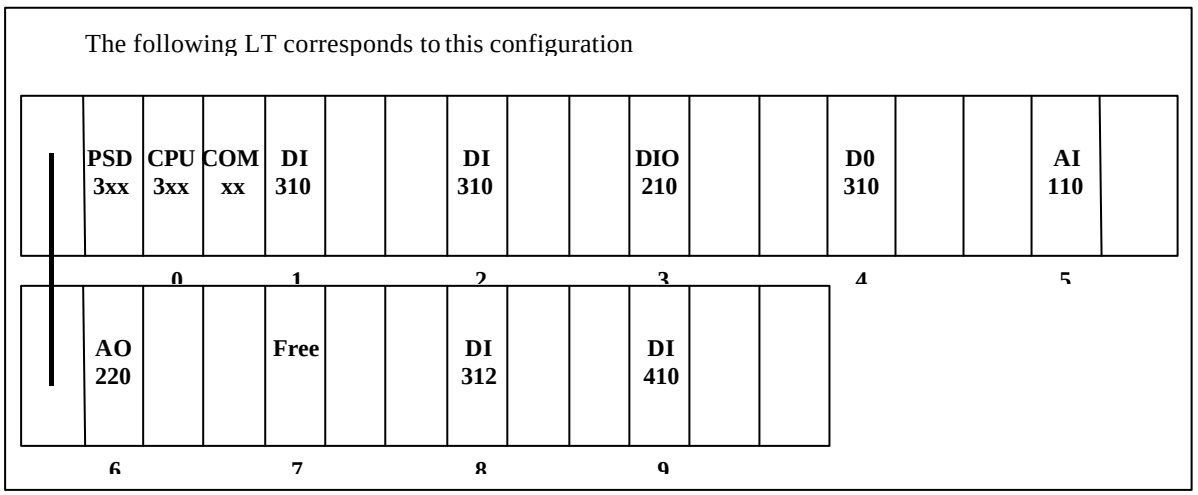


Figure 4: Input/Output Wiring Principle

The logical order of each LT board is indicated below it: CPU = 0, 1st I/O = 1, ... , 15th I/O = 15. Note that this order matches the numbering of the ISaGRAF wiring editor. The DIO210 board is in fact

composed of several boards. It only occupies one slot in the list of declared boards. The same is true of the DI312 board.

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

II.5. Connecting the ISaGRAF Workbench to the LT

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

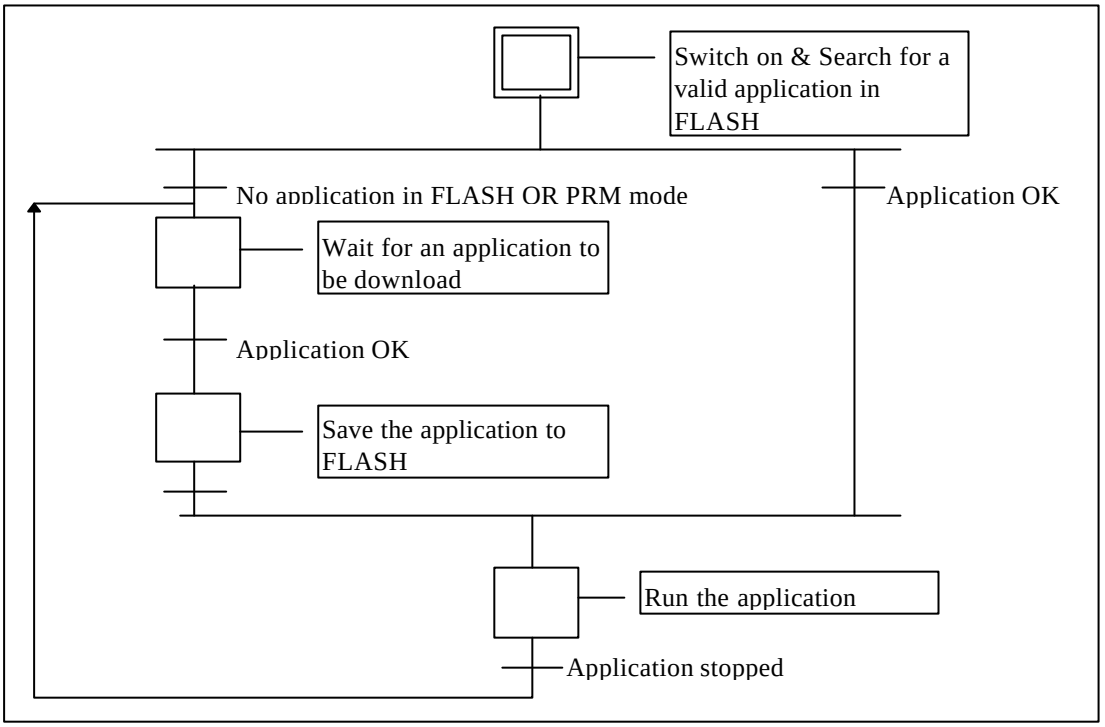


Figure 5: LT ISaGRAF: Theory of Operation

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

To connect to the LT, proceed as follows:

- connect the RS232 cable delivered with the kit between a COM on the PC and **com1 of Terminal Block 1** on the LT,
- create and generate a new application on the workbench. An application is generated by selecting "**Make application**" in the "**Make**" menu. In the list of "**Compiler options**", only select "**TIC code for INTEL**" *.
- 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.

*** TIC** = Target Independent Code: code dedicated to ISaGRAF kernels.

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.16 "**Debugging**" of the ISaGRAF User's Guide" for details concerning the correct use of the debugger.

If no connection has been made between the LT and the workbench, the workbench can be forced not to retrieve an application from the FLASH memory: this is the **LT parameter setting mode**.

II.6. Downloading an Application

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

Important: it is not possible to retrieve an application in the LT ISaGRAF.

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.

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 the LT. 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).

II.7. 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 the LT 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.

III. The CPU board

III.1. CPU board configuration

There's two versions of the CPU board : **cpu3xx** and **cpueth.**. It must always be located in the **first slot** of the I/O wiring editor.

The cpu3xx : cpu board of a LT80 or LT160 without an Ethernet port

The cpueth : cpu board of a LT80 or LT160 with an Ethernet port

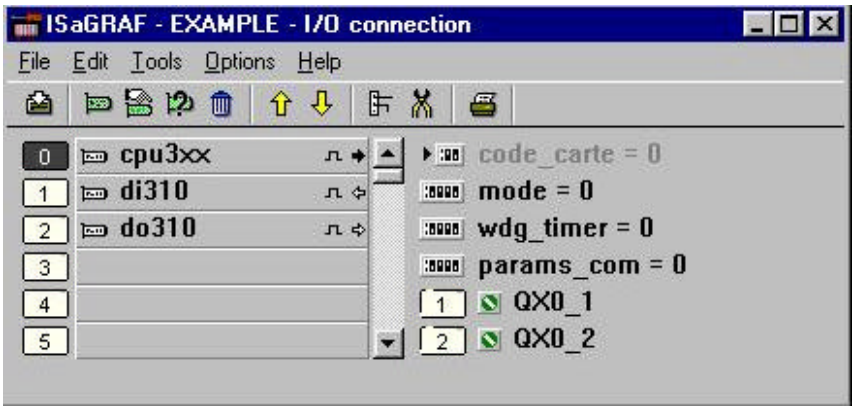
Mutual parameters of those two versions :

- **code_carte**: board code 0 for cpu3xx, board code 256 for cpueth
- **mode**: used to configure the different operating modes of the kernel (Word type, 0 by default),
- **wdg_timer**: used to limit the LT cycle time (in milliseconds). (Word type, 0 by default),
- **params_com**: console link communication parameters (Long Hexa type, 0 by default).

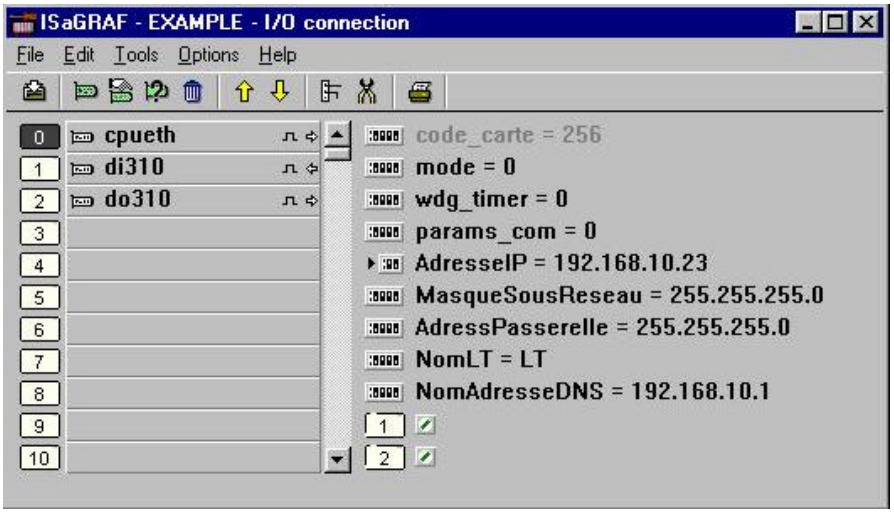
Those boards also has two Boolean outputs representing:

- the power supply **WDG output**. Driving the WDG output inhibits I/O management and refresh,
- the **Lamp Test output** is used to check whether the LEDs of the I/O boards operate correctly. Driving this output lights up all the I/O board LEDs (except NC100).

Example of the cpu3xx configuration :



Example of the cpueth configuration :



Particular parameters for the « cpueth » board :

- **IP address:** this identifies the network and the device (LT PLC) on a TCP/IP network. By default, the IP address is 255.255.255.255. In this case, the LT ignores the other parameters and uses a BOOTP address server, which will send a free IP address to the LT.
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, the LT 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 the LT 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 the LT itself (not the gateway).
Format : xxx.xxx.xxx.xxx where xxx [0..255]

- **LT Name:** LT 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]

III.1.1. Available Operating Modes (mode parameter)

- bit 0: **WDG mode**
  - set to 1: the WDG is managed by the ISaGRAF application (TIC)
  - set to 0: the WDG is managed by the kernel (default).
- bit 1: **Secure mode**
  - set to 1: all boards in the rack must be located on the wiring page; if this is not the case, an IO error is reported and the WDG is enabled.
  - set to 0: no "board present" check (default).
- bit 2: **FreeIO mode**
  - set to 1: an I/O board located on the wiring page may be missing from the rack without generating an IO error.
  - set to 0: an I/O board located on the wiring page is missing from the rack and generates an IO error (default).

III.1.2. Console Link Communication Parameters

The console link is on the com1 of terminal block 1 by default for the two versions cpu3xx and cpueth (params_com=0)

the communication parameters are:

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

To **modify communication parameters** the value of params_com must be changed. Each parameter is encoded on a 4-bit byte of params_com, 4-bit byte no. 1 being the least significant:

T. Block	Com	Slave number	Data	StopBits	Parity	Rate
4-bit byte 8	4 bit byte 7	4 bit byte 6	4 bit byte 5	4 bit byte 4	4 bit byte 3	4 bit byte 2	4 bit byte 1

Default parameter setting (params_com = 0) also corresponds to **params_com = 1101810A**

III.1.2.1. Console Link on RS232

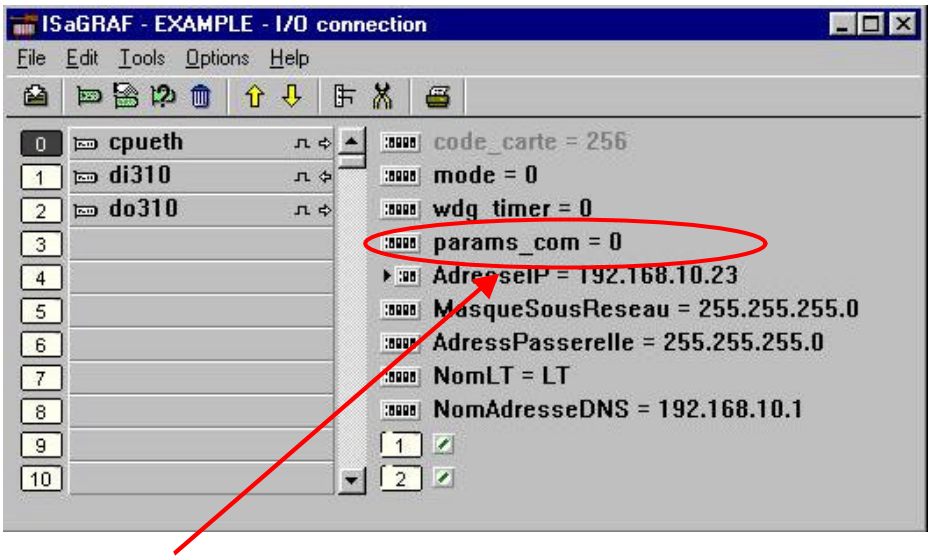
Parameter setting of params_com :

4 bit byte number	4 bit byte value	parameter value
4 bit byte 1 (rate)	5	600 Bauds
	6	1200 Bauds
	7	2400 Bauds
	8	4800 Bauds
	9	9600 Bauds
	A	19200 Bauds
4 bit byte 2 -(parity)	0	None
	1	even
	2	Odd
4 bit byte 3 (Stop bits)	1	1
	2	2
4 bit byte 4 (Data)	7	7 bits
	8	8 bits
4 bit byte 5 and 6 (Slave number)	[0..FF]	[0..255]
4 bit byte 7 (Com)	0	Com0
	1	Com1
4 bit byte 8 (T. Block)	1	Block 1
	2	Block 2
	3	Block 3
	4	Block 4

III.1.2.2. Console Link on the Ethernet port

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

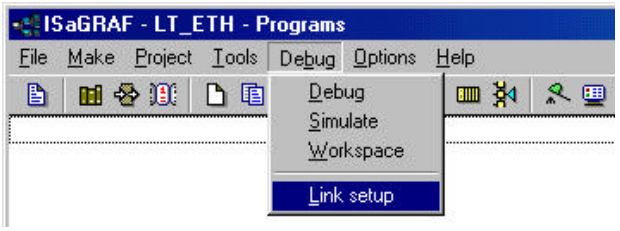
In order to achieve this you just need to modify the params_com in the cabling window as indicated below :



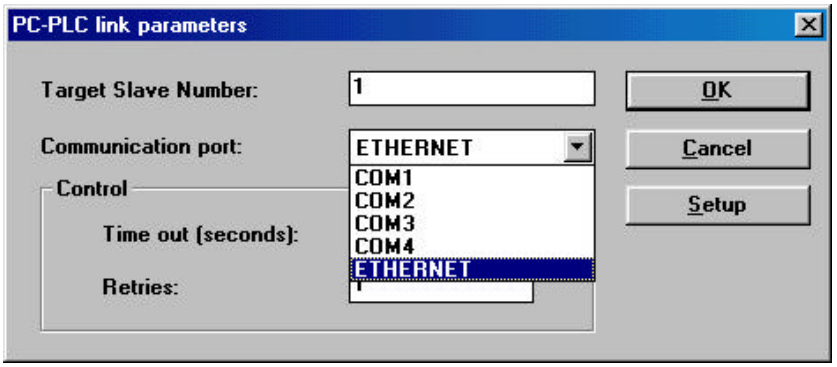
params_com=10000000 designates com0 of terminal block 1 (Ethernet port) as the console link .

params_com designates the port supporting the console link as well as the slave number of the port and its communication format. In the case of an Ethernet port the slave number is replaced by the IP address of the LT and the communication format is imposed by the IEEE 802.3 (10Base-T) standard.

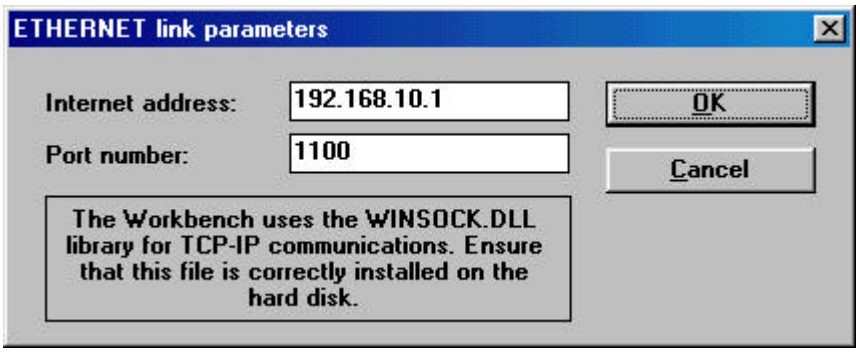
After having modified the params_com, which acts on the LT, generated and loaded the application to the PLC you must select the corresponding port in the ISaGRAF workbench. Click on « **Link setup** » in the « **Debug** » menu in the « Programs » window to do this.



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 LT in the « **Internet address** » field.



The Com0 terminal block 1 led on the CPU module of the Ethernet LT displays the monitoring of the connection with the ISaGRAF workbench. When it is permanently lit the ISaGRAF workbench is connected to the Ethernet LT, it will flash, after a 5 second timeout, when the connection is broken.

III.2. Variable back up :

III.2.1. LT EEPROM Access

The LT160 is equipped with two EEPROMs, while the LT80 is equipped with only one. In the CPU, these components are accessed via a serial communication line using the I2C protocol. Each component can be addressed by its number, 4 and 5 respectively. EEPROMs have a capacity of 1 Kwords each (1024 words). They can be accessed in read or write mode using integers (16-bit words).

Example of a write then read of a word in EEPROM no. 4:

- *Status:= E2p_W(4, 2, ValueToWrite);*
- *ValueRead = E2p_R(4, 2*);

Status is a Boolean variable, *ValueToWrite* and *ValueRead* are analog variables (only the 16 LSBs are meaningful).

Note: the written analog variable will be limited to [-8000h..7FFFh].

Function	**E2p_R**
ACTION	Reads 1 word in an LT EEPROM
SYNTAX	*analog Data E2p_R(analog NumE2p, analog ReadAddr);*
PARAMETERS	NumE2p: [4, 5] on the LT80 the no. is 4 ReadAddr: [0..1023]
RETURNED VALUE	Data: analog value read [0..FFFFh] if error: 10000h
EXAMPLE	*Data:= E2p_R(5, 3);*

Function	**E2p_W**
ACTION	Writes 1 word in an LT EEPROM
SYNTAX	*Boolean Status E2p_W(analog NumE2p, analog WriteAddr, analog Value);*
PARAMETERS	NumE2p: [4,5] on the LT80 the no. is 4 WriteAddr: [0..1023] Value: [0..FFFFh]
RETURNED VALUE	Status: [TRUE, FALSE]
EXAMPLE	*Data:= E2p_W(5, 3, 16#0F0F);*

III.2.2. Non Volatile Variables

The LT is equipped with a **2048 byte backed-up memory**. To back up a variable in the event of an LT power failure, simply **check the "non volatile" box** when declaring the variable in the Dictionary. With an LT, the execution parameters of an ISaGRAF application do not have to be configured as specified in the ISaGRAF User's Guide.

Of the 2048 bytes of the backed-up memory, **1980 are reserved for saving non volatile data.** 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.

The following diagram illustrates the procedure for saving and retrieving a backed-up variable:

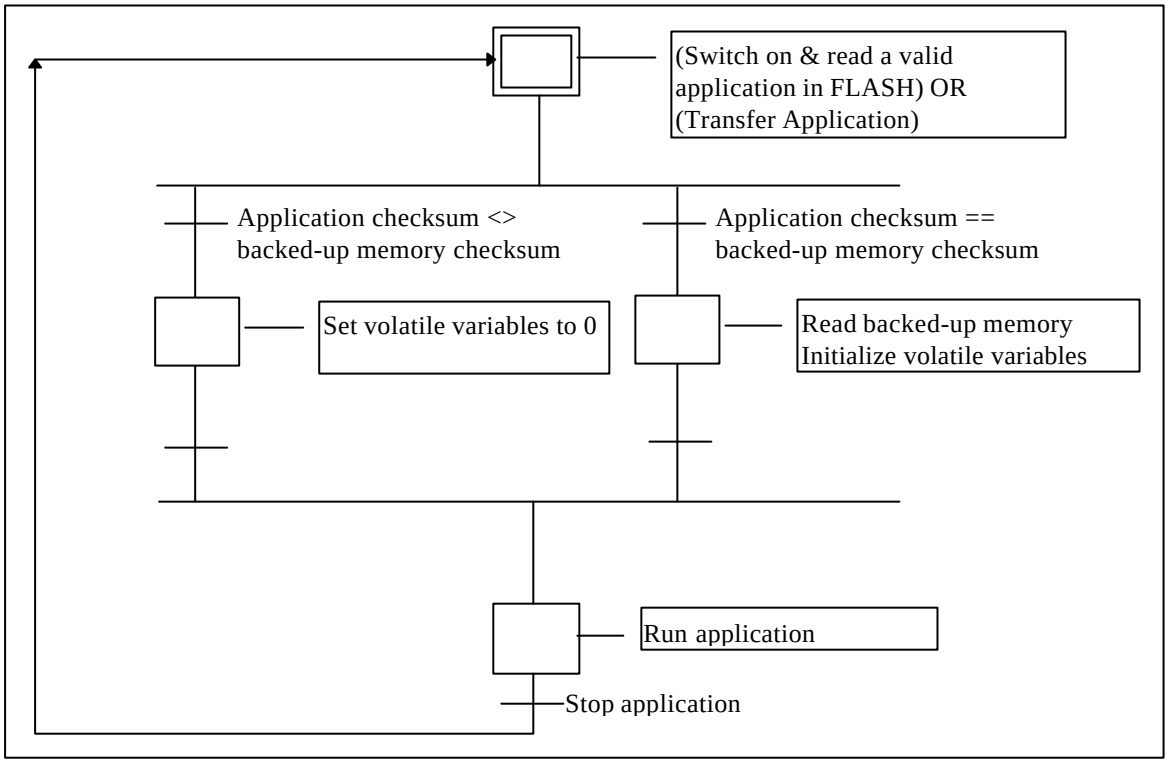


Figure 6: Processing Non Volatile Variables

III.2.3. Backed-up Clock

The LT 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:

- ***DayTim_O()***: initializes access to the clock on the LT,
- ***DayTim_W()***: is used to write and update the clock on the LT,
- ***Day_Time()***: is used to read the clock on the LT.

Day_time() is a "standard" function of the ISaGRAF workbench while the other two functions are supplied by LAI.

Before reading or writing the clock on the LT, the clock must be initialized using the *DayTim_O()* function.

The read or write functions can be used to read or write data in the following format:

- **date: YYYY/MM/DD** --> year, month, day
- **time: HH:MM:SS.hh** --> hour, minute, second, hundredths of a second
- **day of week: [Sunday to Saturday]**

This data is of the **Message type**: ISaGRAF character strings.

Example:

Initializing clock access: *Boolean0:=DayTim_O();*

Reading the date, time and day of the week and chaining in the ClockMess message:

```
ClockMess:= Day_Time(0) + '  ' + Day_Time(1) + '  ' + Day_Time(2);
ClockMess can contain, for example: "1998/10/02  12:31:26.08  Friday"
```

Writing or updating a date, time and day of the week. For example, Friday December 31 1999 at 23 hours 58 minutes 10 seconds and 02 hundredths of a second:

```
Boolean1:= DayTim_W(0, '1999/12/31');
Boolean2:= DayTim_W(1, '23:58:10.02');
Boolean3:= DayTim_W(2, '5');
```

Important: The LT ISaGRAF time date system is Y2K compatible. The year is encoded on four digits.

Function	**DayTim_O**
ACTION	Initializes access to the clock on the LT
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 the LT 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 **Important:** The LT ISaGRAF date system is Y2K compatible. The year is encoded on four digits.
EXAMPLE	**Write or update** a date, time and day of the week. For example, Friday December 31 1999 at 23 hours 58 minutes 10 seconds and 02 hundredths of a second: Boolean1:= DayTim_W(0, '1999/12/31'); 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** The LT 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**

IV. Serial Communication Management

LT ISaGRAF is equipped with **2 to 8 serial links** for the **LT160** and **2 serial links** for the **LT80**. Each LT160 must have 2, 4, 6 or 8 links: an odd number of links is not authorized.

com1 of terminal block 1 (com1 of terminal blocks Com301 and Com302) **is the console link by default**. This console link can, however, be connected to another port. The console link is used to exchange data with the ISaGRAF workbench (downloading, debugging, etc.) or to gain access to dictionary variables via a Modbus/Jbus master (see next section). **com1 of terminal block 1 only supports an RS232 link.** A communication channel must be dedicated to the console link.

com0 of terminal block Com303 is an Ethernet link and cannot be configured in any other way. It is the only Ethernet link possible. It cannot support the console link.

The other RS232 or RS485 links (link 1 of Com301, links 0 and 1 of terminal blocks Com311 and Com312) can support either a **master or slave JBUS** protocol, or a specific **byte transmission / reception** protocol, or the **console link** if necessary.

Provision is made for 4 communication terminal blocks on the LT160. If FIP is available, then 6 serial links + 1 console link can be configured. If FIP is not available, 7 serial links + 1 console link can be configured.

The different types of terminal block are as follows:

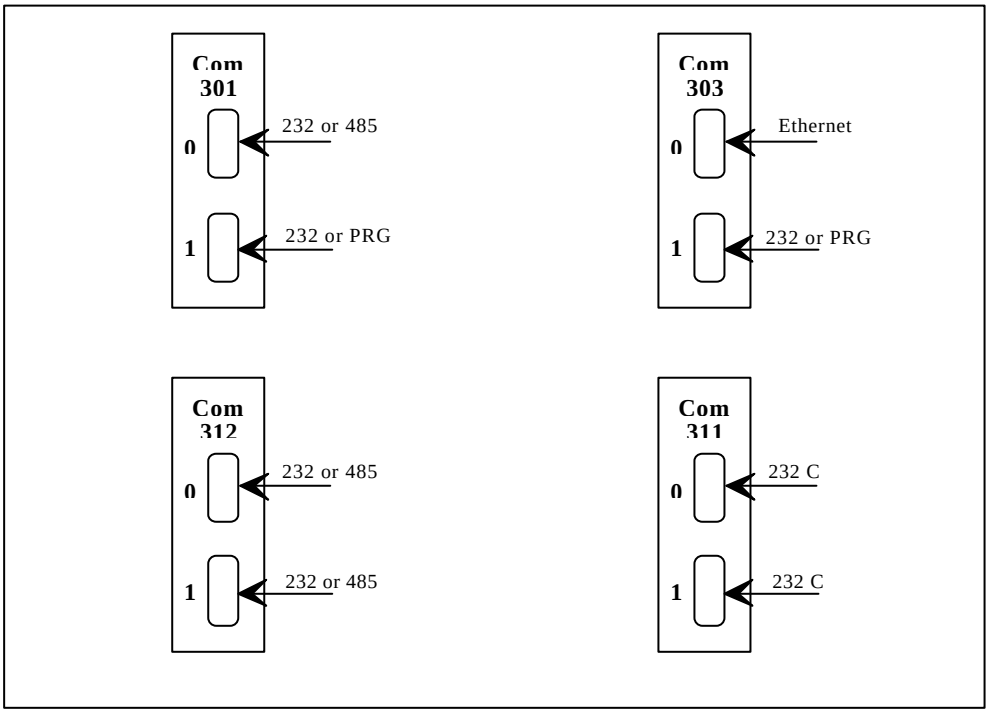


Figure 7: LT Communication Terminal Blocks

There must always be a Com301 or Com302 on terminal block 1. A single Com311 may be used; it must be located on terminal block 2.

Communication on the LT 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 the LT. 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 : the LT SNMP variables are read/write accessible for a SNMP manager.
- SMTP : the LT can send emails.

IV.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 LT port as follows:

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

This table will be refreshed by frames coming from a master or slave. 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:

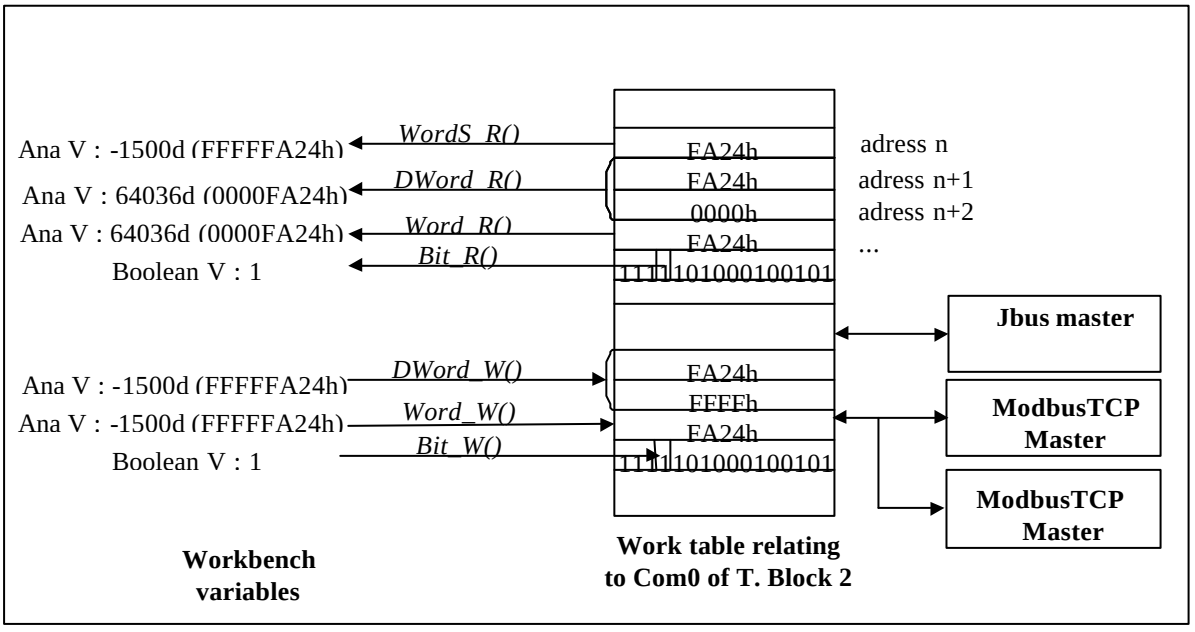


Figure 8: Slave Jbus: Theory of Communication

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:

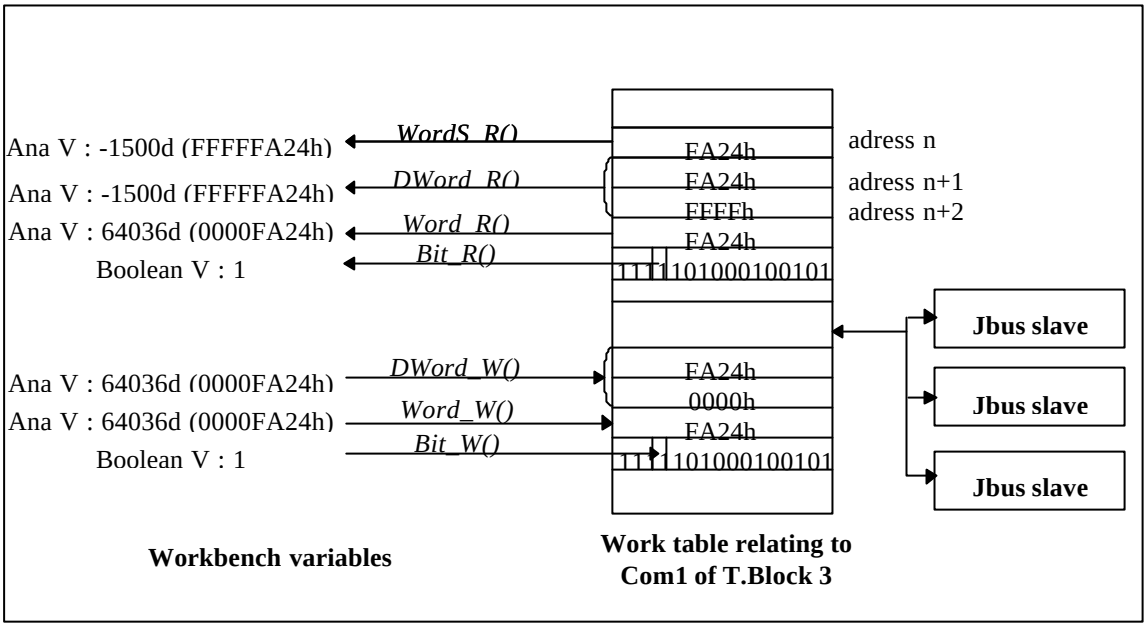


Figure 9: Master Jbus: Theory of Communication

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 LT 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) The LT does not distinguish between output bits and input bits
(2) The LT 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 LT 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 LT 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 LT 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 LT 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 LT 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 LT 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 LT 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 *)

IV.2. Protocols on RS232 and RS485 network :

IV.2.1. Slave Jbus Protocol

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

- **JbusS_O():** opens a Slave Jbus communication sequence on the selected port. The parameters are: TerminalBlock, Com, SlaveNum, TableNum, TableLength,
- **JbusS_P():** configures a Slave Jbus communication sequence on the selected port. The parameters are: TerminalBlock, Com, Speed, Parity, StopBit, Data, Access, Silence, NetworkAddr,
- **JBusS_C():** closes a Slave Jbus communication sequence on the selected port. The parameters are: TerminalBlock, Com. "StopApplication" closes all the communication ports which are open.

These functions return a Boolean status signal (True if there is no error).

Example: declaration of a slave Jbus communication port on com0 of terminal block 2. Communication parameters are:

- terminal block: 2,
- Com: 0,
- slave no.: 12,
- table no.: 3
- table of 100 words associated with this port,
- speed: 9600 bauds,
- parity: odd,
- 1 stop bit,
- 8 data bits,
- read access only
- silent period (3 characters by default): 0
- Modbus/Jbus network address: 0.

Use in an ISaGRAF project:

- Open the communication port: JbusS_O(2, 0, 12, 3, 100),
- Configure the communication parameters: JbusS_P(2, 0, 9, 0, 1, 8, 2, 0, 0),
- Use a Jbus master to read and write data in this table,
- Use the appropriate C functions to refresh workbench variables,
- Close the communication port by ending the program: JbusS_C(2,0).

A C functional block is used to read the **diagnostics counters** of a Slave Jbus port: **JbusS_D()**:

Simply declare a functional block instance in the dictionary and call this instance every cycle. The diagnostics counters are in the returned values of the functional block. A Boolean status signal shows whether the operation has been correctly performed. The call parameters are TerminalBlock and Com.

The **console link (com1 of terminal block 1 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.

These parameters can be modified using the **params_com** parameter **of the cpu3xx board** (see III.1.2).

Function	**JbusS_O**
ACTION	Opens a Slave Jbus port on an LT
SYNTAX	*Boolean JbusS_O(analog TerminalBlock, analog Com, analog SlaveNum, analog TableNum, analog TableLength);*
PARAMETER	TerminalBlock: [1..4] Com: [0,1] 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(2, 0, 12, 3, 100);* (* declares com0 of terminal block 2 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 LT.
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..4] Com: [0,1] 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 com1 terminal block 1) • 115kbauds=13, (only on com1 terminal block 1) • 200Bauds=14. (only on com1 terminal block 1) 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(2,0, 9, 0, 1, 8, 2, 0, 1000);* (* modifies Slave Jbus parameters on com0 of terminal block 2. 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 LT.
SYNTAX	*Boolean JbusS_C(analog TerminalBlock, analog Com);*
PARAMETERS	TerminalBlock: [1..4] Com: [0,1]
RETURNED VALUE	FALSE: Not closed. TRUE: Correctly closed.
EXAMPLE	*Status:= JbusS_C(2, 0);* (* closes the Slave Jbus port on com0 of TerminalBlock 2 *)

Functional Block	**JbusS_D**	
ACTION	Reads the diagnostics counters of a Slave Jbus port on an LT.	
SYNTAX	*Instance.JbusS_D(analog TerminalBlock, analog Com);*	
PARAMETERS	TerminalBlock: [1..4] Com: [0,1]	
RETURNED VALUES	analog RecCorrectCrc	Number of frames received with crc16 correct.
	analog RecBadCrc	Number of frames received with a crc16 error.
	analog Exception	Number of exception answers received.
	analog Broadcast	Number of broadcast requests received.
	analog NoAck	Number of "no acknowledge" exception answers transmitted.
	analog Busy	Number of "slave busy" exception answers transmitted.
	analog CharRecError	Number of wrong characters received.
	analog OverrunRecError	Number of overwrite errors received.
	analog ParityRecError	Number of parity errors received.
	analog FramingRecError	Number of format errors received.
	analog ReceiveBreak	Number of break signals received.
	analog Event	Event counter.
	boolean FbdStatus	(TRUE, FALSE)
DESCRIPTION	This port must first be declared as a Slave Jbus Access to returned values is gained in 2 steps as follows: - declare the JBusS_D type instance in the dictionary and call the instance with the parameters in the program, - read the values returned by the previously declared instance	
EXAMPLE	Instance(2, 0); Data:= Instance.Answer; (* reads the diagnostics counters on com0 of TerminalBlock 2. The number of answers transmitted is written in the Data variable. Each diagnostics counter can be accessed in the returned values *)	

IV.2.2. Master Jbus Protocol

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

- **JbusM_O():**opens a Master Jbus communication sequence on the selected port. The parameters are: TerminalBlock, Com, TableNum, TableLength, TimeOut, NbrTests,
- **JbusM_P():** configures Master Jbus communication parameters on the selected port. The parameters are: TerminalBlock, Com, Speed, Parity, StopBit, RecData TOut, BrdTOut, BusyRet.
- **JbusM_T():** sends a Master Jbus frame to a port and retrieves the communication status. The parameters are: TerminalBlock, Com, SlaveNum, FunctionCode, SlaveAddress, Length, DataAddress, SendFrame.
- **JBusM_C():** closes a Master Jbus communication sequence on the selected port. The parameters are: TerminalBlock, Com. "StopApplication" closes all the communication ports which are open.

JbusM_O(), JbusM_P() and ***JbusM_C()*** return a Boolean status signal (True if there is no error).

JbusM_T() sends a frame if the SendFrame parameter is True. If this parameter is False, ***JbusM_T()*** retrieves the status of the last frame sent to this port (see table below).

Example: declaration of a Master Jbus communication port to com0 of terminal block 2. The communication parameters are:

- Terminal Block: 2,
- Com: 0,
- TableNum: 1
- table of 100 words associated with this port,
- TimeOut: 500 milliseconds.
- NbrTests: 3
- Speed: 19200 bauds,
- Parity: odd,
- 1 stop bit,
- 8 data bits,
- silent period in addition to the 3 end-of-frame detection characters: 0,
- silent period in addition to the 3 end-of-frame characters on broadcast: 5,
- number of retries following a "slave busy" answer: 1.

Use in an ISaGRAF project:

- Open the communication port: JbusM_O(2, 0, 1, 100, 500, 3),
- Configure the communication parameters: JbusM_P(2, 0, 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(2, 0, 6, 3, 5, 1, 12, TRUE); sends the read frame,
  - JbusM_T(2 ,0, 6, 3, 5, 1, 12, FALSE); returns the communication status.
- Close the communication port by ending the program: JbusM_C(2, 0).

A C functional block is used to read the **diagnostics counters** of a Master Jbus: **JbusM_D()**:

Simply declare a functional block instance in the dictionary and call this instance every cycle. The diagnostics counters are in the returned value of the functional block. A Boolean status shows whether the operation has been correctly performed. The call parameters are TerminalBlock and Com.

Function	**JbusM_O**
ACTION	Opens a Master Jbus port on an LT
SYNTAX	*Boolean JbusM_O(analog TerminalBlock, analog Com, analog TableNum analog TableLength, analog TimeOut, analog NbrTests);*
PARAMETERS	TerminalBlock: [1..4] Com: [0,1] 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	FALSE: Not opened.

VALUE	TRUE: Correctly opened.
EXAMPLE	*Status:= JbusM_O(2, 0, 1, 100, 500, 3);* (* declares the com0 of TerminalBlock 2 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 LT
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..4] Com: [0,1] 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 com1 of TerminalBlock 1) • 115kbauds=13, (only on com1 of TerminalBlock 1) • 200Bauds=14. (only on com1 of TerminalBlock 1) 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(2, 0, 9, 0, 1, 8, 5, 0, 1);* (* modifies the Master Jbus parameters on com0 of TerminalBlock 2. 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 LT 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..4] Com: [0,1] 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 3
EXAMPLE	(* On com0 of TerminalBlock 2 *) (* read 1 word at address 5 on slave 1, store at address 2 *) *Status:= JbusM_T(2, 0, 1, 3, 5, 1, 2, TRUE);* (* read status of previous frame *) *Status:= JbusM_T(2, 0, 1, 3, 5, 1, 2, FALSE);*

Function	JbusM_C
ACTION	Closes a Master Jbus port on an LT
SYNTAX	*Boolean JbusM_C(analog TerminalBlock, analog Com);*
PARAMETERS	TerminalBlock: [1..4] Com: [0,1]
RETURNED VALUE	FALSE: Not closed. TRUE: Correctly closed.
EXAMPLE	*Status:= JbusM_C(2, 0);* (* closes the Master Jbus port on com0 of TerminalBlock 2 *)

Functional Block	JbusM_D	
ACTION	Reads the diagnostics counters on a Master Jbus port on an LT	
SYNTAX	*Instance.JbusM_D(analog TerminalBlock, analog Com);*	
PARAMETERS	TerminalBlock: [1..4] Com: [0,1]	
RETURNED VALUES	analog RecCorrectCrc	Number of frames received with crc16 correct.
	analog RecBadCrc	Number of frames received with a crc16 error.
	analog Exception	Number of exception answers received.
	analog Question	Number of questions transmitted.
	analog Broadcast	Number of broadcast requests received.
	analog NoAck	Number of "no acknowledge" exception answers transmitted.
	analog Busy	Number of "slave busy" exception answers transmitted.

	analog CharRecError	Number of wrong characters received.
	analog OverrunRecError	Number of overwrite errors received.
	analog ParityRecError	Number of parity errors received.
	analog FramingRecError	Number of format errors received.
	analog ReceiveBreak	Number of break signals received.
	analog TimeOut	Number of silent periods in transmission mode overrun.
	analog TransmitError	Number of wrong transmission requests.
	analog FrameError	Number of wrong frames received (but with crc16 correct).
	Boolean FbdStatus	(TRUE, FALSE)
DESCRIPTION	This port must first be declared as a Master Jbus Access to returned values is gained in 2 steps as follows: - declare the JBusM_D type instance in the dictionary and call the instance with the parameters in the program, - read the values returned by the previously declared instance	
EXAMPLE	Instance(2, 0); Data:= Instance.Question; (*reads the diagnostics counters on com0 of TerminalBlock 2. The number of answers transmitted is written in the Data variable. Each diagnostics counter can be accessed in the returned values *)	

IV.2.3. Byte Transmission/Reception Protocol

LT 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 the LT 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 the LT ISaGRAF during an interrupt.

The following C functions are provided:

- **NulPro_0():** opens a simple communication sequence on an LT port.
- **NulPro_P():** configures a simple communication sequence on an LT port.
- **NulPro_S():** writes in the transmission queue on an LT port.
- **NulPro_R():** reads in the reception queue on an LT port.
- **NulPro_N():** reads the number of characters located in the reception queue on an LT port.
- **NulPro_C():** closes a simple communication sequence on an LT 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(2, 0, 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(2, 0, 16#20, 12);

- Write the characters of the message located at network address 20h of the dictionary to the transmission file: NulPro_S(2, 0, 16#20, 0);
- Close the communication port: NulPro_C(2, 0);

Example : Printer Management

The byte transmission/reception functions on a serial link can be used for simple control of a **serial printer**. Any LT 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 com0 of terminal block 2:

- Open the communication port: NulPro_O(2, 0, 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(2,0, 16#20, 0);

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

Function	**NulPro_O**
ACTION	Opens a simple communication sequence on an LT port
SYNTAX	*Boolean NulPro_O(analog TerminalBlock, analog Com, analog RecTabLength, analog TransmitTabLength, analog Mode);*
PARAMETERS	TerminalBlock: [1..4] Com: [0,1] 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(2, 0, 1020, 510, 1);* (* declares a simple communication sequence on com0 of TerminalBlock 2 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 LT port
SYNTAX	*Boolean NulPro_S(analog TerminalBlock, analog Com, analog MsgNetworkAddr, analog NrbChar);*
PARAMETERS	TerminalBlock: [1..4] Com: [0,1] 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(2, 0, 16#A0, 0); (* sends the message located at address 16#A0 on the transmission queue associated with com0 of TerminalBlock 2 *)

Function	NulPro_R
ACTION	Reads in the reception queue on an LT port
SYNTAX	*analog NulPro_R(analog TerminalBlock, analog Com, analog MsgNetworkAddr, analog NbrChar);*
PARAMETERS	TerminalBlock: [1..4] Com: [0,1] 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(2, 0, 16#A0, 20); (* reads 20 characters in the reception queue associated with com0 of TerminalBlock 2: 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 LT port.
SYNTAX	*analog NulPro_N(analog TerminalBlock, analog Com);*
PARAMETERS	TerminalBlock: [1..4] Com: [0,1]
RETURNED VALUE	NbrChar: number of characters in the reception queue
EXAMPLE	NbrChar:= NulPro_N(2, 0); (* reads the number of characters located in the reception queue associated with com0 of TerminalBlock 2 *)

Function	NulPro_P
ACTION	Configures a simple communication sequence on an LT port.
SYNTAX	*Boolean NulPro_C(analog TerminalBlock, analog Com, analog Speed, analog Parity, analog StopBit, analog Data);*
PARAMETERS	TerminalBlock: [1..4] Com: [0,1] 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 com 1 of terminal block 1) • 115kbauds=13, (only on com 1 of terminal block 1) • 200Bauds=14. (only on com 1 of terminal block 1)

	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(2, 0, 9, 0, 1, 8); (* modifies the communication parameters on com0 of TerminalBlock 2 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 LT port.
SYNTAX	*Boolean NulPro_C(analog TerminalBlock, analog Com);*
PARAMETERS	TerminalBlock: [1..4] Com: [0,1]
RETURNED VALUE	FALSE: Not closed. TRUE: Correctly closed.
EXAMPLE	StatusPro_C(2, 0); (* Closes a simple communication sequence on com0 of TerminalBlock 2 *)

IV.2.4. RS232 Link Control Signals

Apart from the default console link (terminal block 1, com1), the LT has two types of RS232 links:

- an RS232 link including RTS and CTS control signals,
- an RS232C link which also includes DTR, DSR and DCD control signals.

These signals can be driven using two read or write C functions:

- **RS232_R():** reads a signal on the RS232 link
- **RS232_W():** drives (writes) a signal on the RS232 link.

These signals have a high state of approximately 10 V and a low state of approximately -10V.

Example of an RTS signal write:

Status:= RS232S_W(2, 0, 1, TRUE); (* enables the RTS wire on com0 of terminal block 2 *)

Example of a DSR signal read:

Status:= RS232S_R(2, 0, 1); (* reads the DSR wire on com0 of terminal block 2 *)

Note: in half-duplex mode, RTS is managed automatically (enabled upon transmission then disabled).

Function	**RS232_R**
ACTION	Reads an RS232 link signal
SYNTAX	*analog Status RS232_R(analog TerminalBlock, analog Com, analog SignalType);*
PARAMETERS	TerminalBlock: [1..4] Com: [0, 1]

	SignalType: 0=CTS (wire 8) 1=DSR (wire 6) 2=DCD (wire 1)
RETURNED VALUE	Status: 1=high state (+10V) 0=low state (-10V) -1=read or parameter setting error
EXAMPLE	Status:= RS232S_R(2, 0, 1); (* reads the DSR wire on com0 of TerminalBlock 2 *)

Function	**RS232_W**
ACTION	Drives (writes) an RS232 link signal
SYNTAX	*Boolean Status RS232_W(analog TerminalBlock, analog Com, analog SignalType, Boolean status);*
PARAMETERS	TerminalBlock: [1..4] Com: [0, 1] Type Signal: 0=RTS (wire 7) 1=DTR (wire 4) Status:TRUE=high state (+10V) FALSE=low state (-10V)
RETURNED VALUE	Status: TRUE= signal correctly written FALSE= error
EXAMPLE	Status:= RS232S_W(2, 0, 1, TRUE); (*enables the RTS wire on com0 of TerminalBlock 2*)

Example : Modem Management

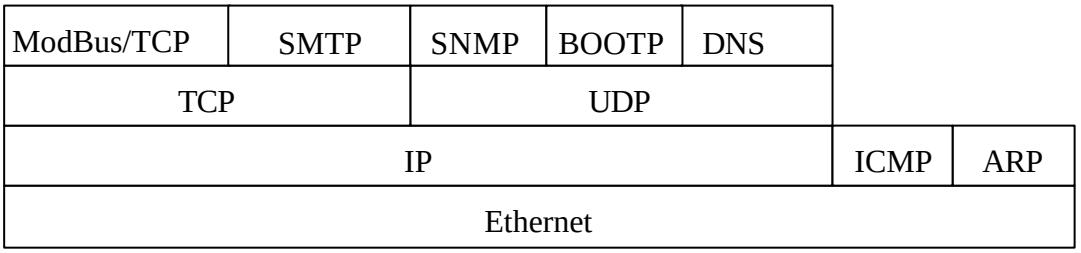
A modem can be controlled in the same way as a serial printer.

The modem can be selected by an RS232C link signal and the connection procedure carried out using the simple protocol. Data can then be transmitted either using the simple protocol or the master Modbus/Jbus protocol.

IV.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 the LT on Ethernet :

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

IV.3.1. The 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 » channels on asynchronous links of the product. The differences with the Modbus protocol over asynchronous channel 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. LT can open four simultaneous channels with numerous masters and slaves on the network. Leroy Automation sets the limit, to 4 masters, and 4 slaves.
- No diffusion available.

IV.3.1.1. The Modbus/TCP slave protocol

Two C functions and a C functional block are available for using this protocol :

- **tcpmbs_o) :** Opens a Modbus/TCP Slave channel on an Ethernet LT
- **tcpmbs_s() :** Monitors the presence of a modbus/TCP master
- **tcpmbs_d() :** Reads the diagnostics counters on a Modbus/TCP Slave channel.

Usage in an ISaGRAF project:

Function	**TCPMbS_O**
ACTION	Opens 4 Modbus/TCP Slave channel on an Ethernet LT **WARNING : the Modbus/TCP slave channel must be opened 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 LT 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 the LT IP address. Note: Four masters can access this table "simultaneously" *)

Function	**TCPMbS_S**
ACTION	Monitors the presence of 1 to 4 masters on the ModBus/TCP slave channel
SYNTAX	Boolean tcpmbs_S(analog MasterIndex, message IPAddress, analog TimeOut);
PARAMETERS	MasterIndex : [1..4] IPAddress : message type variable containing the IP address TimeOut : [0..FFFF] TimeOut in milliseconds
RETURNED VALUE	FALSE : Master absent. TRUE : Master present.
EXAMPLE	Status = Omodbs_S(1, address_M, 5000); (* monitor the presence of the master whose IP address is contained in the « address_M » message type variable. This master will have an index of 1. This index will be used to identify the master who will no longer be monitored once a new master connects even through 4 masters were already being monitored. If this master is absent for at least 5 seconds, the status will switch from TRUE to FALSE *)

Functional Block	**TCPMbS_D**	
ACTION	Reads the diagnostics counters on a Modbus/TCP Slave channel.	
SYNTAX	*Instance();* *Data:=Instance.Return value;*	
PARAMETERS		
RETURNED VALUES	analog AdresseIP_[1..4]	IP address of the last master currently connected.
	analog RecCorrectCrc_[1..4]	Number of frames received with crc16 correct.
	analog RecBadCrc_[1..4]	Number of frames received with a crc16 error.
	analog Event_[1..4]	Number of answers transmitted.
	analog Busy_[1..4]	Number of "slave busy" exception answers transmitted.
	analog Exception_[1..4]	Number of exception answers received.
	Boolean FbdStatus	(TRUE, FALSE)
DESCRIPTION	This channel must first be declared as a Modbus/TCP Slave Access to returned values is gained in 2 steps as follows: - declare the TCPMBS_D type instance in the dictionary and call the instance with the parameters in the program, - read the values returned by the previously declared instance - xxxx_i represents the diagnostics counters of the master connected number i	
EXAMPLE	Instance(); Data:= Instance.Event_1; (* reads the diagnostics counters on the Ethernet channel. The number of events sent to master no. 1 is written in the Data variable. Each diagnostic counter is accessible in the returned values *)	

IV.3.1.2. The Modbus/TCP master protocol

Two C functions and a C functional block are available for using this protocol :

- **tcpmbm_o() :** Opens a Modbus/TCP Master channel on an Ethernet LT
- **tcpmbm_t() :** Sends a master modbus/TCP frame on an Ethernet LT port
- **tcpmbm_d() :** Reads the diagnostics counters on a Modbus/TCP Master channel.

Usage in an ISaGRAF project :

Function	**TCPMbM_O**
ACTION	Opens one Modbus/TCP Master channel on an Ethernet LT **WARNING : only four Modbus/TCP master channel can be opened simultaneously.**
SYNTAX	Boolean tcpmbs_O(analog LineNum, analog TableNum, analog TableLength, analog Access);
PARAMETERS	LineNum: [1..4] 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 LT 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 the LT 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..4] 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 annex 3 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*)

Functional Block	TCPMbM_D	
ACTION	Reads the diagnostics counters on a Modbus/TCP master channel.	
SYNTAX	*Instance(*analog LineNum, message SlaveNum*);* *Data:=Instance.Return value;*	
PARAMETERS	LineNum : [1..4] SlaveNum : IP address or DNS Address of the modbus/TCP slave	
RETURNED VALUES	analog RetryCounter	Number of retry
	analog Question	Number of Questions
	analog ExceptionAnswer	Number of frames received with a crc16 error.
	analog TimeOutAnswer	Number of answers transmitted.
	analog Busy	Number of "slave busy" exception answers transmitted.
	analog NoAck	Number of exception answers received.
	analog GoodAnswer	Number of break signals received.
	analog FrameError	IP address of the last master currently connected.
	analog HeaderError	Number of frames received with crc16 correct.
	analog TimeOutDnsRequest	Number of frames received with a crc16 error.
	analog DnsRequest	Number of answers transmitted.
	analog TimeOutTcpOpen	Number of exception answers received.
DESCRIPTION	This channel must first be declared as a Modbus/TCP Master Access to returned values is gained in 2 steps as follows: - declare the TCPMBM_D type instance in the dictionary and call the instance with the parameters in the program, - read the values returned by the previously declared instance - xxxx_i represents the diagnostics counters of the master connected number I	
EXAMPLE	Instance(1, "192.10.5.2"); Data:= Instance.Question; (* reads the diagnostics counters on the Ethernet port. The number of questions sent to slave no."192.10.5.2" is written in the Data variable. Each diagnostic counter is accessible in the returned values *)	

IV.3.2. The SNMP Protocol

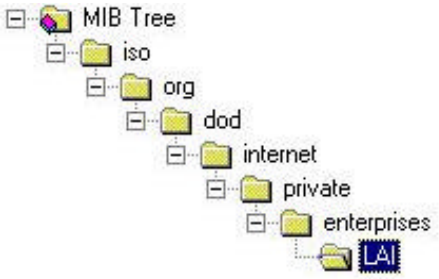
- **SNMP**: *Simple Network Management Protocol*: Standard protocol used on Internet for the administration of hosts, routers and other devices on the network.

The LT is an SNMP agent. The LT SNMP variables are read/write accessible for a SNMP manager. ISaGRAF does not allow a limitation of this access.

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 SNMP protocol enables access to LT 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.entrées and its identifier is written as *1.3.6.1.4.1.4273.1.0* (0 being the instance of the variable with this name).

4273 is the identifier of LAI PLCs

MIB additions with LT variables defined by ISaGRAF in ASN1 :

```
LAI DEFINITIONS ::= BEGIN

    IMPORTS
        enterprises, OBJECT-TYPE
            FROM RFC1155-SMI
        Integer32
            FROM RFC1213-MIB;

lai     OBJECT IDENTIFIER ::= { enterprises 4273 }

    input OBJECT-TYPE
        SYNTAX  Boolean
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
            "Boolean Input"
        ::= { lai 1 }

    output OBJECT-TYPE
        SYNTAX  Integer32
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
            "Block DO 310 : 32 boolean output"
        ::= { lai 2 }
```

my_message OBJECT-TYPE
SYNTAX Integer32
ACCESS read-only
STATUS mandatory
DESCRIPTION
Internal message : 10 Charakters"
::= { lai 3 }
END

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

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

Function	**SnmpVB_C**
ACTION	Defines the SNMP order, in the 1.3.6.1.4.1.4273 branch of a Boolean variable in the ISaGRAF dictionary
SYNTAX	Boolean Status:= SnmpVB_C(analog Flag, analog OID, analog AddrBooVar);
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 AddrAnaVar: [0..FFFF] dictionary network address (hex) of the Boolean variable
RETURNED VALUE	FALSE: Operation not carried out. TRUE: Operation correctly carried out.
EXAMPLE	Status = SnmpVB_C(0, 1, 16#12); (* the Boolean variable with address 1 in the dictionary can be accessed by the SNMP; its identifier is the following 1.3.6.1.4.1.4273.**1**.0*)

Function	**SnmpVA_C**
ACTION	Defines the SNMP order, in the 1.3.6.1.4.1.4273 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, 2, 16#20); (* the analog variable with address 16#20in the dictionary can be accessed by SNMP; its identifier is the following 1.3.6.1.4.1.4273.**2**.0*)

Function	**SnmpVM_C**
ACTION	Defines the SNMP order, in the 1.3.6.1.4.1.4273 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, 7, 16#20); (* 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.**3**.0*)

ATTENTION : Writing to an SNMP variable which has not been defined in ISaGRAF belonging to the LT branch (1.3.6.1.4.1.4273) will initiate a reset of the LT.

IV.3.3. Sending electronic mail

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

Number of mail transmission attempts : unlimited as long as the LT 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 the LT is the following: « LT Message number X » where X=LT 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	Initialise the SMTP server address. This address may be an IP address or a server name (ex : smtp.anydomain) **ATTENTION: The mail server must be unique ; it is therefore prohibited initialise 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 succeded.
EXAMPLE	Status = EMail_I(Address_Server); (* The address of the outgoing mail server (SMTP) is initialised. 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 *)

V. Input/Output Boards

The following input/output boards can be used on the LT from the ISaGRAF workbench:

- DI310: 32 digital inputs
- DI410: 64 digital inputs
- DO310: 32 digital outputs
- AI110: 8 analog inputs
- AO120:8 analog outputs
- AI210: 16 analog inputs
- AO220:16 analog outputs.

The following special features can also be used:

- DIO210: 16 digital inputs and 8 digital outputs
- AIO320: 8 analog inputs and 4 analog outputs
- DI312: 32 digital safety inputs
- NC100: numerical controller for single-axis motor in "stepper" mode
- DIO130: secure digital inputs/outputs
- DI130: secure digital

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

The **Flt LED** of each input/output board is managed by the board electronics. If this is not refreshed **within 1 second, this output is driven by a monostable.** This monostable remains enabled until the board has been initialized.

Note: this monostable is equal to 100 milliseconds on DIO130 models and 200 milliseconds on DI130.

V.1. DI310 Board

The DI310 board is made up of 32 Boolean inputs.

V.2. DI410 Board

The DI410 board is made up of 64 Boolean inputs.

V.3. DO310 Board

The DO310 board is made up of 32 Boolean outputs.

V.4. AI110 Board

The AI110 board is made up of 8 analog inputs (16 signed bits).

V.5. AO120 Board

The AO120 board is made up of 8 analog outputs (16 signed bits).

V.6. AI210 Board

The AI210 board is made up of 16 analog inputs (16 signed bits).

V.7. AO220 Board

The AO220 board is made up of 16 analog outputs (16 signed bits).

Current-to-Voltage Conversion Table <=> Number of Points	
Non isolated current inputs	±21.1mA => ±32767 points.
Non isolated voltage inputs	±10.25V => ±32767 points.
Isolated voltage inputs	0 to 10.25V => 32767 points.
Isolated current inputs	0 to 21.1mA => 32767 points.
Current outputs	0/32767 points => 4/20mA
Voltage outputs	±32767 points => ±10V

V.8. DI312 Module

The DI312 module is made up of 2 boards:

- Inputs: 32 Boolean inputs
- Fault: 32 faults relating to inputs.

DI312 modules 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.

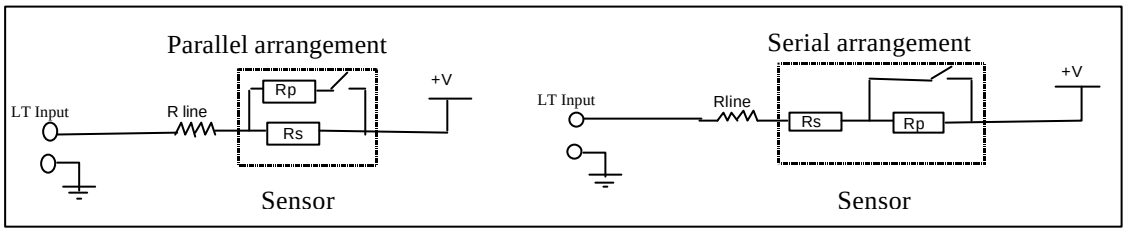


Figure 10: Wiring of Safety Inputs

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 a **DI312 board** and is therefore the same for all the channels of a single DI312 module.

DI312 board parameters are as follows:

- 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.
- RCNO: only one value for all inputs.
- RCNF: only one value for all inputs.
- Rline: only one value for all inputs.

For each channel, the status bit and alarm bit encode 4 possible states:

Input Status (Green LED)	Alarm (Red LED)	Description
0 (OFF)	0 (OFF)	Sensor normally open
1 (ON)	0 (OFF)	Sensor normally closed
0 (OFF)	1 (ON)	Input not connected or short-circuit at 0V
1 (ON)	1 (ON)	Short-circuit at +V

Resistance constraints:

- - Resistors must have a tolerance of no more than 1%.
- - 0.7 kΩ < **Rcno** < 22 kΩ
- - Rcno induces the current in the measuring device: I (mA) = 22 (V)/(1+ Rcno (kΩ)) given that I must be between 1 mA and 9.96 mA . If the calculated value of I is more than 9.96 mA, saturate it at 9.96 mA.
- - (2.95(V)/I (mA)) < **Rcnf** (kΩ) < Rcno (kΩ) - Rline (kΩ) - 1.95 (V)/I (mA)
- - **Rline** < 0.2 kΩ.

V.9. DIO210 Module

The DIO210 module is made up of 2 boards:

- 16 Boolean inputs
- 8 Boolean outputs.

V.10. AIO320 Module

The AIO320 module is made up of 3 boards:

- 8 analog inputs
- 16 Boolean inputs (inputs [1..8] and [9..16] are respectively dedicated to high-threshold and low-threshold overruns of the 8 analog inputs
- 4 analog outputs.

Current-to-Voltage Conversion Table <=> Number of Points on AIO320	
Non isolated current inputs	±20mA => ±32767 points.
Non isolated voltage inputs	±10V => ±32767 points.
Current outputs	0/32767 points => 4/20mA
Voltage outputs	±32767 points => ±10V
PT100 sensors inputs	-50°C=>-500 points +350°C=>+3500 points

V.11. NC100 Module

The NC100 module is made up of 7 boards:

- Status: 8 analog inputs (16 bits)
  - STCN board status
  - LCMD last command received
  - STNA last exchange status
  - MACH machine status word
  - STER error status word
  - STIO input/output status
  - reserved
  - reserved
- Command: 8 analog outputs (16 bits)
  - reserved
  - reserved
  - CMIO output command
  - MOUV movement command
  - MACX extended parameters
  - MODE basic modes of movement
  - IXOR polarity reversal of input bits
  - reserved
- Control: 6 analog outputs (32 bits)
  - APOS absolute position
  - RPOS relative position
  - VEL linear velocity
  - JVEL jogging velocity
  - ACC acceleration
  - DEC deceleration
- Cmd: 1 analog output exchange block command

- Bloc_R: 1 analog input data read in exchange block
- Bloc_W: 1 analog outputdata to write in exchange block
- Status: 1 analog input result of exchange block command

Using an N100 block involves reading and writing data in command and control status boards. The exchange block provides access to all NC100 addresses.

The startup and advanced-level use of an NC100 block are described in the **"NC100 - Stepper Version" Startup Manual** supplied by **Socitec**.
This manual is needed to implement an NC100 block and is supplied to all purchasers of an NC100 block.

In order to carry out a simple movement, simply start the demo project supplied in the LT ISaGRAF libraries: D_NC100.

V.12. DIO130 Module

The DIO130 module is made up of 3 boards:

- OUTPUT RELAYS : 16 Boolean outputs (remote controls)
- SIGNAL : 4 Boolean outputs (LEDs)
- INPUT : 8 Boolean inputs (remote signaling)

V.13. DI130 Module

The DI130 module is made up of 2 boards:

- I: 16 Boolean inputs (remote signals)
- Iinv: 16 Boolean inputs (reverse remote signals)

VI. LT monitoring and diagnostic

VI.1. CPU Board and Input/Output Board Status Read

The ***IOStatus(BoardOrder)*** function is used to read the status of the CPU board.
Meaning of CPU board status bits:

Function	**IOStatus**
ACTION	Reads the status of CPU and I/O boards
SYNTAX	*analog Status IOStatus(analog BoardOrder);*
PARAMETERS	BoardOrder [0..15] where 0=CPU, 1=1st I/O board ... , 15=15th I/O board
RETURNED VALUE	Data: analog value read [0..FFFFh]
DESCRIPTION	Only the status of correctly initialized boards is transferred Status = 0: board parameters set in the workbench but board inaccessible on the I/O bus Status = -1: Wrong BoardOrder or board parameters not set in the workbench Status composition: see "LT ISaGRAF" doc, section III
EXAMPLE	*Status:= IOStatus(1);* (* reads the status of the first I/O board *)

• **bit 0**	set to 1 at end of each cycle
• **bit 1**	set to 1 if WDG LED lit up
• **bit 2**	set to 0 if I/O fault (I/O LED lit up)
• **bit 3**	set to 1 if initializations completed (set to 1 at end of each cycle)
• **bit 4**	set to 0 if Test error (Test LED lit up)
• **bit 5**	set to 1 if PRM detected
• **bits 6-15**	reserved

The ***IOStatus(BoardOrder)*** function is used to read the status of each board (*BoardOrder*[1..15]).

Meaning of input/output board status bits:

• **bits 0-7**	**Board code [0..FFh] bit 6 not significant**
• **bit 8**	set to 1 if internal power supply of board is correct
• **bit 9**	see table below
• **bit 10**	WDG: set to 0 if WDG is set to 1 (LED lit up)
• **bit 11**	see table below
• **bits 12-15**	**Position of board on bus [0..15]**

Input/output board codes:

Board	**Status Bit 11**	**Status Bit 9**	**Board Code [0..FFh] Mask Bit 6: BFh**
DI310	1	Al Ext	03h or 43h
DI312	NS	Al Ext	14h
DI410	1	Al Ext	06h
DO310	Fault	Al Ext	05h or 45h
DIO210	Monostable	VRel	16h

AI110	0	0	80h
AI210	0	0	81h
AO120	0	0	84h
AO220	0	0	85h
AIO320	Monostable	0	83h
NC100	1	1	30h to 37h
DIO130	Monostable	VRel	58h
DI130	Monostable	0	59h

Where:

- NS: not significant
- AI Ext: set to 1 if the external power supply at the terminal blocks is in the Valim ±20% range.
- (*) AI ext of DI312: External power supply equal to 24V±10%
- Fault: set to 0 in the event of overload on a digital output channel
- Monostable: set to 1 if board is correctly refreshed; for 4TCD/TSD operation is reversed.
- Vrel: set to 1 if relays are correctly powered.

IOInits(BoardOrder) Function allows you to read the Initialization Counter of CPU and I/O Boards

Function	**IOInits**
ACTION	Reads the initialization counter of I/O boards
SYNTAX	*analog Status IOInits(analog BoardOrder);*
PARAMETERS	BoardOrder[1..15] where 1=1st I/O board... , 15=15th I/O board
RETURNED VALUE	Data: analog value read [0..FFFFh]
DESCRIPTION	Only the counter of correctly initialized boards is transferred The initialization counter has a value of 1 if board initialization is correct (only one)
EXAMPLE	*Status:= IOInits(1);* (* reads the initialization counter of the first I/O board *)

VI.2. 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: LT dynamic memory damaged, LT automatic reboot.
- 130: illegal board added in the case of an LT80 (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: LT 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.

VI.3. Switching the LT to Parameter Setting Mode

For an LT ISaGRAF, switching to parameter setting mode means running only the ISaGRAF kernel with no TIC (Target Independent Code) application downloaded.

This mode, called **PRM**, is symbolized by a LED of the same name on the CPU.

To switch to PRM mode from MSDOS:

- switch off the LT,
- connect the LT (Terminal Block 1, com1) and the PC (com1 or com2) using an RS232 cable,
- run the "**LTPRM.EXE**" program included on the "Libraries" diskette under DOS. This program waits until it recognizes an LT. It displays the message "**Waiting for connection on com x**". The wait may be interrupted by pressing a key,
- the Com can be entered as an LTPRM parameter: comx (com1 by default),
- switch on the LT,
- in its initialization sequence, the LT switches to PRM mode and its PRM LED lights up without flashing. The PC displays the message "**Success, PRM mode command sent**."
- the LT ISaGRAF is now in PRM mode.

To switch to PRM mode from Windows95:

- switch off the LT,
- connect the LT (Terminal Block 1, com1) and the PC (com1 or com2) using an RS232 cable,
- run the "**LTPRMW95.EXE**" program included on the "Libraries" diskette under Windows95. This program waits until it recognizes an LT.
- select the communication port on the PC [1..4],
- select "**Start**",
- it displays the message "**Wait time overrun. Waiting for acknowledge character**".
- the wait may be interrupted by pressing "**Stop**",
- switch on the LT,
- in its initialization sequence, the LT switches to PRM mode and its PRM LED lights up without flashing. The PC displays the message "**LT set to PRM by default**"
- The LT ISaGRAF is now in PRM mode.

The only way to be sure that the switch to PRM mode has succeeded is to check that the PRM LED lights up without flashing.

VI.4. LT ISaGRAF LEDs : Power Supply, CPU and I/O boards

VI.4.1. Power Supply LEDs

If the power supply is present and correct, the corresponding green LED lights up without flashing.

VI.4.2. CPU LEDs

- **Green RUN LED**:
  - ◊ flashes slowly (1 s) if the TIC application (ISaGRAF) is run correctly.
  - ◊ flashes 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.

- **Red TEST LED**:
  - ◊ OFF if operation is correct.
  - ◊ lights up without flashing if the program read in the Flash memory is not correct or if the I/O board is not correctly inserted.
- **RED I/O LED**:
  - ◊ OFF if operation is correct.
  - ◊ lights up without flashing if the board is not correctly inserted or if at least one I/O board status is incorrect while the program is running.
- **Green PRM LED**:
  - ◊ lights up without flashing if the equipment is in PRM mode when the LT is booted. Does not turn off until the LT is rebooted without PRM mode.
  - ◊ in all other cases, this LED is OFF.
- **Green PRG LED**: not managed by ISaGRAF.
  - ◊ lights up without flashing if the equipment is in PRG mode when the LT is booted: bridge between pins 5 and 6 of com 1.1. Does not turn off until the LT is rebooted without PRG mode.
  - ◊ in all other cases, this LED is OFF.
- **Red WDG LED**:
  - ◊ by default, lights up without flashing.
  - ◊ turns off as soon as the ISaGRAF kernel is active AND the program read in the Flash memory is correct. This is the case if the WDG is managed automatically by the kernel (see III.5).
- **Green comx LEDs**:
  - ◊ the comx LED of the console link lights up without flashing.
  - ◊ other LEDs: as required by user. E.g. lights up without flashing as soon as a com is correctly initialized.
- **Green FIP LED**:
  - ◊ as required by the user. E.g. lights up without flashing as soon as the LOAD and START STATION commands have been correctly executed.
    - ◊ turns off in the event of a STOP_STATION command.

The user can only intervene on the comx and FIP LEDs from the ISaGRAF workbench. The ***LEDCPU(LedOrder, LedStatus)*** function is implemented in the workbench.

Function	**LedCPU**
ACTION	Controls the comx and FIP LEDs of the CPU
SYNTAX	*Boolean LedCPU (analog LedOrder, Boolean Status);*
PARAMETERS	LedOrder: from 6,8 to 14 Status: FALSE = OFF, TRUE = ON.
RETURNED VALUE	FALSE: LED not controlled. TRUE: OK.
DESCRIPTION	Only LEDs 6,8 to 14 can be controlled.
NOTES	LED order [6,8..14]: • 0 : Led Run: inaccessible • 1 : Led Test: inaccessible • 2 : Led I/O : inaccessible • 3 : Led Prm: inaccessible • 4 : Led Prg : inaccessible • 5 : Led Wdg: inaccessible • 6 : Led com 4.1 • 7 : inaccessible • ---------- • 8 : Led com 1.0 • 9 : Led com 1.1 • 10: Led com 2.0 • 11: Led com 2.1 • 12: Led com 3.0 • 13: Led com 3.1 • 14: Led com 4.0 • 15: inaccessible

	A "Stop Application" on the ISaGRAF debugger does not turn off the com LEDs managed by the user (6, 8..14).
EXAMPLE	*Status:= LedCPU(9, TRUE);* (* lights up the LED of com1 on the CPU *)

VI.4.3. LED Control on Input/Output Boards

Input/output LEDs are automatically refreshed by the kernel. However, the workbench provides functions to control the status of these LEDs differently (e.g. reversing the logic of digital I/Os). There is one function per board:

- *LEDDI310(RangCarte, Leds_1_32);*
- *LEDDI410(RangCarte, Leds_1_32, Leds_33_64);*
- *LEDDI312(RangCarte, Leds_1_32, Leds_33_64);*
- *LEDDO310(RangCarte, Leds_1_32);*
- *LEDDIO210(RangCarte, LedsI_1_16, LedsO_1_8);*
- *LEDAI110(RangCarte, LedsV_1_8, LedsR_1_8);*
- *LEDAI210(RangCarte, LedsV_1_16, LedsR_1_16);*
- *LEDAO120(RangCarte, Leds_1_8);*
- *LEDAO220(RangCarte, Leds_1_16);*
- *LEDAIO320(RangCarte, Leds_1_8);*

Caution: The LED command must be rewritten every cycle, otherwise it is overwritten by the LT kernel: digital output command.

Function	**LedDI310**
ACTION	Controls the LEDs of a DI310 board
SYNTAX	*Boolean LedDI310(analog BoardOrder, analog Leds_1_32)*
PARAMETERS	BoardOrder: from 1 to 15. Leds_1_32: status of the first 32 LEDs (0 = OFF, 1 = ON).
RETURNED VALUE	FALSE: Not refreshed. TRUE: Correctly refreshed.
DESCRIPTION	The status of each LED is represented by a bit.
EXAMPLE	*Status:= LedDI310(1, 16#FFFFFFFF);* (* lights up all the LEDs of a DI310 in slot 1 *)

Function	**LedDI410**
ACTION	Controls the LEDs of a DI410 board
SYNTAX	*Boolean LedDI410(analog BoardOrder, analog Leds_1_32; analog Leds_33_64)*
PARAMETERS	BoardOrder: from 1 to 15. Leds_1_32: status of the first 32 LEDs (0 = OFF, 1 = ON). Leds_33_64: status of the last 32 LEDs (0 = OFF, 1 = ON).
RETURNED VALUE	FALSE: Not refreshed. TRUE: Correctly refreshed.
DESCRIPTION	The status of each LED is represented by a bit.
EXAMPLE	*Status:= LedDI410(1, 16#FFFFFFFF, 16#FFFFFFFF);* (*lights up all the LEDs of a DI410 in slot 1 *)

Function	**LedDI312**
ACTION	Controls the LEDS of a DI312 board
SYNTAX	*Boolean LedDI312(analog BoardOrder, analog Leds_1_32; analog Leds_33_64)*
PARAMETERS	BoardOrder: from 1 to 15. Leds_1_32: status of the 32 green LEDs (0 = OFF, 1 = ON). Leds_33_64: status of the 32 red LEDs (0 = OFF, 1 = ON).
RETURNED VALUE	FALSE: Not refreshed. TRUE: Correctly refreshed.
DESCRIPTION	The status of each LED is represented by a bit.
EXAMPLE	*Status:= LedDI312(1, 16#FFFFFFFF, 16#FFFFFFFF);* (*lights up all the LEDs of a DI312 in slot 1 *)

Function	**LedDO310**

ACTION	Controls the LEDs of a DO310 board
SYNTAX	*Boolean LedDO310(analog BoardOrder, analog Leds_1_32)*
PARAMETERS	BoardOrder: from 1 to 15 Leds_1_32: status of the 32 red LEDs (0 = OFF, 1 = ON).
RETURNED VALUE	FALSE: Not refreshed. TRUE: Correctly refreshed.
DESCRIPTION	The status of each LED is represented by a bit.
EXAMPLE	*Status:= LedDO310(1, 16#FFFFFFFF);* (*lights up all the LEDs of a DO310 in slot 1 *)

Function	**LedDIO21**
ACTION	Controls the LEDs of a DIO210 board
SYNTAX	*Boolean LedDIO21(analog BoardOrder, analog Ledsi_1_16, analog Ledso_1_8)*
PARAMETERS	BoardOrder: from 1 to 15 Ledsi_1_16: status of the 16 green LEDs (0 = OFF, 1 = ON). Ledso_1_8: status of the 8 red LEDs (0 = OFF, 1 = ON).
RETURNED VALUE	FALSE: Not refreshed. TRUE: Correctly refreshed.
DESCRIPTION	The status of each LED is represented by a bit.
EXAMPLE	*Status:= LedDIO21(1, 16#FFFF, 16#FF);* (*lights up all the LEDs of a DIO210 in slot 1 *)

Function	**LedAI110**
ACTION	Controls the LEDs of an AI110 board
SYNTAX	*Boolean LedAI110(analog BoardOrder, analog LedsV_1_8; analog LedsR_1_8)*
PARAMETERS	BoardOrder: from 1 to 15. LedsV_1_8: status of the 8 green LEDs (0 = OFF, 1 = ON). LedsR_1_8: status of the 8 red LEDs (0 = OFF, 1 = ON).
RETURNED VALUE	FALSE: Not refreshed. TRUE: Correctly refreshed.
DESCRIPTION	The status of each LED is represented by a bit.
EXAMPLE	*Status:= LedAI110(1, 16#FF, 16#FF);* (*lights up all the LEDs of an AI110 in slot 1 *)

Function	**LedAI210**
ACTION	Controls the LEDs of an AI210 board
SYNTAX	*Boolean LedAI210(analog BoardOrder, analog LedsV_1_16; analog LedsR_1_16)*
PARAMETERS	BoardOrder: from 1 to 15. LedsV_1_16: status of the 16 green LEDs (0 = OFF, 1 = ON). LedsR_1_16: status of the 16 red LEDs (0 = OFF, 1 = ON).
RETURNED VALUE	FALSE: Not refreshed. TRUE: Correctly refreshed.
DESCRIPTION	The status of each LED is represented by a bit.
EXAMPLE	*Status:= LedAI210(1, 16#FFFF, 16#FFFF);* (*lights up all the LEDs of an AI210 in slot 1 *)

Function	**LedAO120**
ACTION	Controls the LEDs of an AO120 board
SYNTAX	*Boolean LedAO120(analog BoardOrder, analog Leds_1_8)*
PARAMETERS	BoardOrder: from 1 to 15. Leds_1_8: status of the 8 green LEDs (0 = OFF, 1 = ON).
RETURNED VALUE	FALSE: Not refreshed. TRUE: Correctly refreshed.
DESCRIPTION	The status of each LED is represented by a bit.
EXAMPLE	*Status:= LedAO120(1, 16#FF);* (*lights up all the LEDs of an AO120 in slot 1 *)

Function	**LedAO220**

ACTION	Controls the LEDs of an AO220 board
SYNTAX	*Boolean LedAO220(analog BoardOrder, analog Leds_1_16)*
PARAMETERS	BoardOrder: from 1 to 15. Leds_1_16: status of the 16 green LEDs (0 = OFF, 1 = ON).
RETURNED VALUE	FALSE: Not refreshed. TRUE: Correctly refreshed.
DESCRIPTION	The status of each LED is represented by a bit.
EXAMPLE	*Status:= LedAO220(1, 16#FFFF);* (* lights up all the LEDs of an AO220 in slot 1 *)

Function	**LedAIO32**
ACTION	Controls the LEDs of an AIO320 board
SYNTAX	*Boolean LedAIO32(analog BoardOrder, analog Leds_1_8)*
PARAMETERS	BoardOrder: from 1 to 15. Leds_1_8: status of the 8 amber LEDs (0 = OFF, 1 = ON).
RETURNED VALUE	FALSE: Not refreshed. TRUE: Correctly refreshed.
DESCRIPTION	The status of each LED is represented by a bit.
EXAMPLE	*Status:= LedAIO32(1, 16#FF);* (*lights up all the LEDs of an AIO320 in slot 1 *)

VI.5. LT Identification Functions

The **version of the embedded software** (kernel) supported by the LT can be read by the ***LTVer();*** function.
The **CPU type** of the LT can be checked by the ***LTType();*** function
The **serial number** of the LT can be checked by the ***LTSerial();*** function

VI.5.1. Version of Kernel Embedded on the LT Target

Function	**LTVer**
ACTION	Reads the version of the kernel embedded on the target
SYNTAX	*analog Version := LTVer();*
PARAMETERS	None
RETURNED VALUE	version 1.04: 0104h PF = 1 and pf =4 version 1.1: 0110h PF = 1 and pf =10h
EXAMPLE	*VersionLT:= LTVer();*

VI.5.2. Type of CPU Installed on the LT

Function	**LTType**
ACTION	Reads the type of CPU installed on the LT as well as the type of daughterboard.
SYNTAX	*analog TypeLT:= LTType();*
PARAMETERS	None
RETURNED VALUE	**Low-order byte: type of LT** 2,3: LT160 4,5: ACS21 6,7: LT80 other: unidentified CPU **High-order byte: type of daughterboard** 13 : Only 1 communication terminal block: Com301 11 : Com301 + 1 communication terminal block 7 : Com301 + 2 to 3 communication terminal blocks 6 : Only 1 communication terminal block: Com302 (FIP) 3 : Com302 + 1 communication terminal block 1 : Com302 + 2 to 3 communication terminal blocks
EXAMPLE	*TypeLT:= LTType();*

VI.5.3. Serial number of the LT

Function	**LTSerial**
ACTION	Reads the LT serial number
SYNTAX	*analog SerialNum:= LTSerial();*
PARAMETERS	None
RETURNED VALUE	Serial number: [0..FFFFh]
EXAMPLE	*SerialNum:= LTSerial();*

APPENDIX 1

LT without CPU identification

Some LT CPUs (386) may not contain a specific identification code. In such cases, you will need to know and specify the component used in conjunction with the serial port. This is the case of ACS21 (SNCF) PLCs. In the ISaGRAF C functions used to manage serial communication, this parameter replaces the TerminalBlock parameter. In this cases, the Com parameter is inoperative but must have a value of 0 or 1.

The codes used to identify components are as follows:

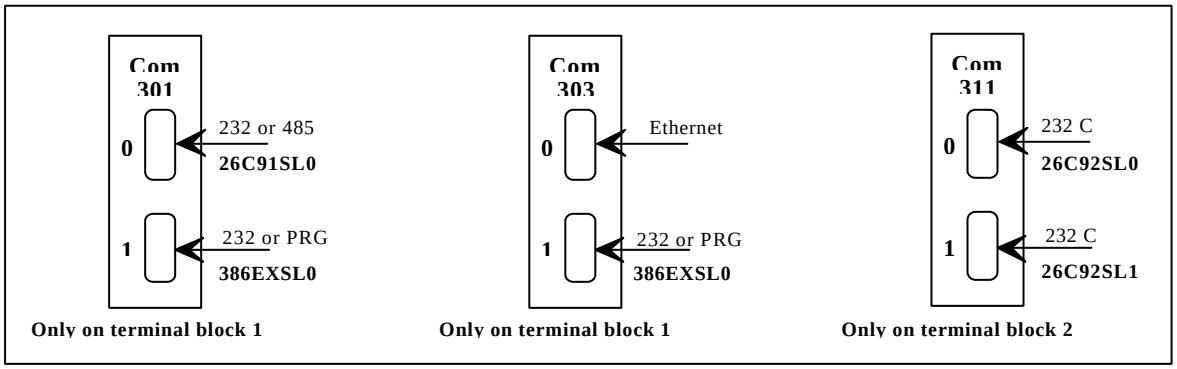
Code	C Function Parameter	TerminalBlock	Com
26C91SL0	**16h**	1	0
386EXSL0	**1000h**	1	1
26C92SL0	**10h**	2	0
26C92SL1	**11h**	2	1
26C94SL0	**12h**	3	0
26C94SL1	**13h**	3	1
26C94SL2	**14h**	4	0
26C94SL3	**15h**	4	1

Example: *Status = JbusS_C(2, 0);*

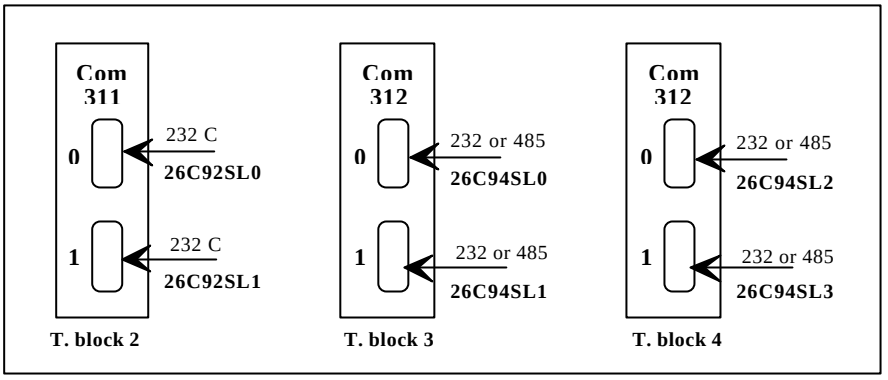
The previous example applied to an ACS21 becomes: *Status = JbusS_C(16#10, 0);*

Equivalence between codes and communication ports:

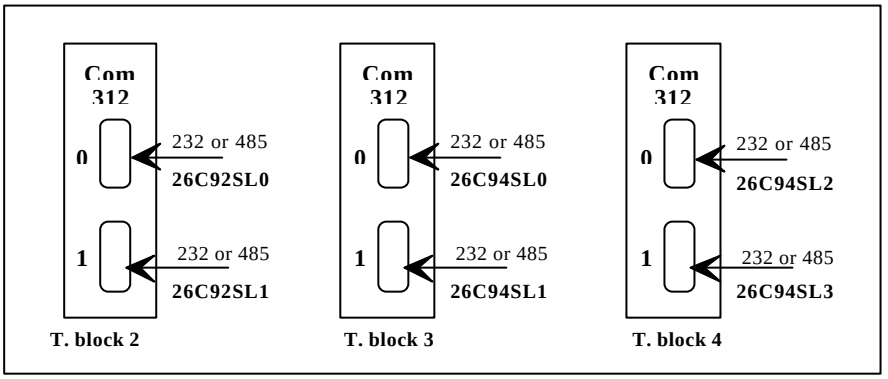
1- Terminal block 1 (for Com 301 and Com 303) and terminal block 2 (for Com 311):



2- Terminal block 2 3 and 4 (with Com 311 on terminal block 2 and Com 312 on terminal block 3 and 4):

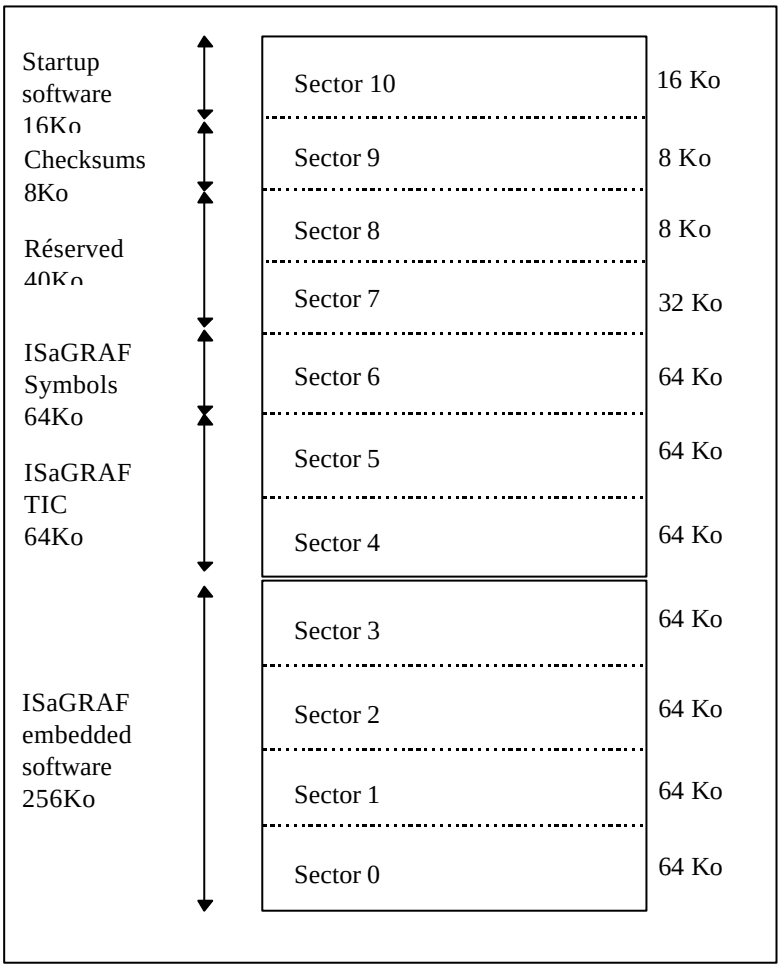


3- Terminal block 2 3 and 4 (with Com 312 on the 3 terminal blocks):



APPENDIX 2

FLASH Memory Map of the LT ISaGRAF



APPENDIX 3

Modbus/Jbus asynchronous and Modbus/TCP error 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

FUNCTIONS C INDEX