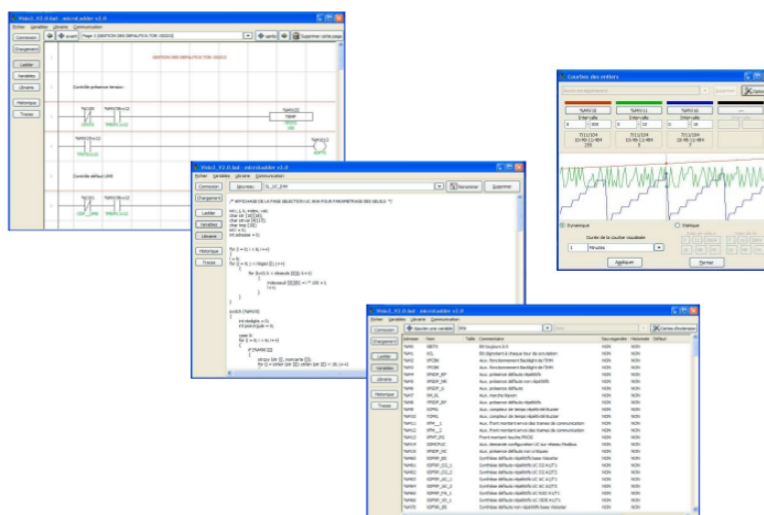


User Manual



Date	Modification
18/05/16	Start writing the original version
25/08/17	End of writing of the original version
16/10/17	Adding %SW91 to %SW94 (Wi-Fi DNS server)
05/12/17	Added precision for Linux installation Add LogGetVariable function The LogFindVariable function only works on mnemonics
04/01/18	Added additional information about the initial values of the variables Adding library import information Adding RF port setting information (LORA) Added system code options "Optimize code size" and "RF" Creation of Version 14 : new layout following SIREA's graphic chart

Table of contents

1 INTRODUCTION AND INSTALLATION.....	7
1.1 System requirements.....	7
1.2 Compiler and system code.....	7
1.3 MicroLADDER.....	7
1.4 MicroDRIVER.....	7
1.5 MicroHMI.....	7
2 PROGRAMMING IN MicroLADDER.....	7
2.1 Getting started.....	7
2.1.1 Start and quit MicroLADDER.....	8
2.1.2 Using the HMI.....	8
2.1.1 The programming languages.....	14
2.1.2 Importing firmware.....	15
2.1.3 Connecting with the PLC.....	15
2.1.4 Using pages.....	17
2.1.5 Creating a page.....	18
2.1.6 Call of a page.....	18
2.1.7 Variable editor.....	19
2.1.1 Variable editor tool bar.....	19
2.1.2 Variable properties.....	21
2.1.3 Type of variables.....	24
2.1.1 Using variables in a program.....	26
2.1.1 Variable multi-edition.....	26
2.2 Objects available in Ladder.....	27
2.3 Creation of a program.....	29
2.3.1 First program in ladder.....	29
2.3.2 First program in C.....	31
2.3.1 Combining Ladder and C (including an example).....	31
2.3.2 Compilation and loading to the PLC.....	33
2.4 Creation of a function.....	33
2.4.1 Defining variables.....	34
2.4.2 Use of a function.....	34
2.4.2.1 Ladder.....	34
2.4.2.2 C code.....	35
2.5 Using timer.....	35
2.5.1 How to use a timer.....	36
2.5.2 Examples of the use of timer.....	36
2.6 Using a GUI (HMI).....	37
2.7 MicroHMI.....	37
2.8 How to insert a GUI.....	37
2.9 How to configure a GUI.....	37
2.9.1 Example.....	38

3 SOFTWARE ENVIRONMENT.....	39
3.1 System architecture.....	39
3.1.1 Monitor software.....	39
3.1.1.1 LED showing operational stat of the PLC.....	40
3.1.1.2 Monitor mode.....	40
3.1.1.3 Transferring the program from the SD card to the PLC memory.....	40
3.1.1.4 STOP mode.....	40
3.1.1.5 RUN mode.....	40
3.1.1.6 Incompatible versions between the monitor and the application.....	40
3.1.2 "MAIN.CFG" file.....	40
3.1.3 Loadmain file.....	41
3.1.4 Loading an application.....	41
3.1.4.1 MicroCONTROL.....	42
3.1.4.2 SD Card.....	42
3.2 Type of data.....	43
3.2.1 DIGITAL inputs.....	43
3.2.2 ANALOG inputs.....	43
3.2.3 DIGITAL outputs.....	43
3.2.4 ANALOG outputs.....	43
3.2.5 PWM outputs.....	43
3.2.6 Boolean.....	44
3.2.7 Integer.....	44
3.2.8 Long.....	44
3.2.9 Float.....	44
3.2.10 String.....	44
3.2.11 System bits.....	45
3.2.12 System words.....	47
3.2.13 Modbus Detail.....	55
3.2.14 Radio Frequency details.....	57
3.2.15 Edge management.....	57
3.3 Importing the variables by overwriting the present variables.....	57
3.3.1 General.....	57
3.3.2 Programming.....	58
3.3.3 Input / Output.....	59
3.3.4 Communication.....	59
3.3.5 Logging.....	59
3.3.6 Remote server.....	60
3.3.7 Display.....	60
3.4 Application implementation.....	61
3.4.1 Fonction.....	61
3.4.2 Fonction implementation.....	61
3.4.3 Using a function.....	62
3.4.4 Use in Ladder.....	62
3.4.5 Using library.....	62

3.4.6 Calling a page.....	62
3.4.7 Temporization.....	62
3.4.8 Global variables.....	63
3.4.9 Available RAM size.....	64
3.5 Saved variables.....	64
3.5.1 Saved RAM.....	64
3.5.2 EEPROM or FRAM.....	65
3.5.3 System words.....	65
3.6 Watchdog.....	65
3.6.1 Cycle Time Exceeded.....	65
3.6.2 Watchdog soft.....	65
3.7 System functions/Internal functions.....	65
3.7.1 Configuration of the system code.....	65
3.7.1.1 Optimize the code size.....	65
3.7.1.2 AUTO_STOP.....	65
3.7.2 DHCP.....	66
3.7.3 DNS.....	66
3.7.4 GFX.....	66
3.7.5 HTTP.....	66
3.7.6 LCD.....	66
3.7.7 RF.....	66
3.8 Communicating without protocol.....	66
3.8.1 Setting.....	66
3.8.1.1 ComSetCharTimeout (Parameter1, Parameter2).....	66
3.8.2 Transfer.....	67
3.8.2.1 ComPush (parameter1, parameter2, parameter3).....	67
3.8.2.2 ComPushByte (parameter1, parameter2).....	67
3.8.2.3 ComSend (parameter1).....	68
3.8.2.4 ComFlushOutput (parameter1).....	68
3.8.3 Reception.....	69
3.8.3.1 Return = ComGetFrameLength (parameter1).....	69
3.8.3.2 Return = ComGetFrame (parameter1).....	69
3.8.3.3 ComFlushInput (parameter1).....	70
3.8.4 Modbus.....	70
3.8.4.1 Slave Modbus and slave Modbus TCP.....	70
3.8.4.2 Master Modbus.....	70
3.8.4.3 Return = ModbusRead (parameter1 to parameter7).....	70
3.8.4.4 Return = ModbusWrite (parameter1 to parameter8).....	71
3.8.4.5 Master TCP Modbus.....	71
3.8.4.6 ComConnect (parameter1, parameter2, parameter3).....	71
3.8.4.7 ComClose (parameter1).....	72
3.8.4.8 Return = ComGetSockState(parameter1).....	72
3.9 Files management.....	72
3.9.1 Structure of file names.....	72

3.10 Structure "FSFile "	73
3.10.1 Return = FSOpen (parameter1, parameter2, parameter3).....	73
3.10.2 FSClose (parameter1).....	73
3.10.3 Return = FSSeek (parameter1, parameter2).....	73
3.10.4 Return = FSDelete (parameter1).....	74
3.10.5 Return = FSWrite (parameter1, parameter2, parameter3).....	74
3.10.6 Return = FSRead (parameter1, parameter2, parameter3).....	74
3.10.7 Return = FSReadLine (parameter1, parameter2, parameter3, parameter4)	74
3.10.8 Return = FSWriteCSVRow (parameter1, parameter2, parameter3).....	75
3.10.9 Return = FSReadCSVRow (parameter1 to parameter4).....	75
3.10.10 Return = FSMove (parameter1, parameter2).....	76
3.10.11 Return : FSCreateFolder (parameter1).....	76
3.10.12 Return: FSCopy (parameter1, parameter2).....	76
3.11 Log Management.....	76
3.11.1 Time stamp format.....	76
3.11.2 "Date" structure.....	77
3.11.3 Return = dateToTime (parameter1).....	77
3.11.4 Return = timeToDate (parameter1).....	77
3.11.5 LogValue Format.....	78
3.11.6 Return = LogAlQuery (parameter1, parameter2, parameter3).....	78
3.11.7 Return = LogEvQuery (parameter1, parameter2, parameter3).....	78
3.11.8 Return = LogTrQuery (parameter1 to parameter4).....	79
3.11.9 Return = LogFetch (parameter1).....	79
3.11.10 Return = LogAlSave (parameter1, parameter2, parameter3).....	79
3.11.11 Return = LogEvSave (parameter1, parameter2, parameter3).....	80
3.11.12 Return = LogTrSave (parameter1 to parameter4).....	80
3.11.13 Return = LogSave (parameter1).....	80
3.11.14 LogPurge ().....	81
3.11.15 LogEvPurge ().....	81
3.11.16 Return = LogTrPurge (parameter1).....	81
4 IO Bus.....	81
4.1 Declaration of slave Equipment.....	81
4.2 Declaration of variables.....	82
4.3 State of communication with equipment.....	82
5 HTTP Protocol.....	83
6 Wi-Fi.....	83
6.1 Mode.....	83
6.2 WSocketSSID (parameter 1).....	84
6.3 WSocketSecKey (parameter 1).....	84
6.4 WSocketSecType (parameter 1).....	84
6.5 WSocketKey (parameter 1, parameter 2).....	84
7 Connection to a server.....	85

7.1 Setting.....	85
7.2 Setting by programming.....	85
8 History Management.....	86
8.1 Alarm.....	86
8.2 Event.....	87
8.3 Curve.....	87
8.4 Return = LogFindVariable (parameter1).....	87
8.5 Return = LogGetVariable (parameter 1).....	87
9 Functions for character strings.....	88
9.1 Return = StrToNum (parameter1).....	88
9.2 Sprintf (parameter1, parameter2, parameter 3).....	88
9.3 Upper (parameter1).....	88
9.4 Lower (parameter1).....	88
9.5 StrSet (parameter1, parameter2, parameter3).....	89
9.6 StrToLower (parameter1).....	89
9.7 StrToUpper (parameter1).....	89
9.8 Return = StrGetChar (parameter1, parameter2).....	89
9.9 Return = StrSetChar (parameter1, parameter2, parameter3).....	89

All Rights Reserved

No part of this document or any of its contents may be reproduced, copied, modified or adapted, without the prior written consent of the author, unless otherwise indicated for stand-alone materials.

1 INTRODUCTION AND INSTALLATION

This document provides a guide for the use of MicroLADDER V14 software and associated software, for the programming of PLCs (Programmable Logic Controllers) of mArm7 family. Specific features of each PLC are described within a separated document.

1.1 System requirements

MicroLADDER can be installed either on Windows or Linux (Debian) operating systems.

1.2 Compiler and system code

Both compiler and system code are present online and work automatically as soon as you have an internet connection. They will transform your MicroLADDER file to make them readable by your PLC. The version of the system code is automatically selected based on the version of the MicroLADDER used. However, it is important to be careful that these versions are compatible with the boot installed on the card.

1.3 MicroLADDER

This software can be used to create the application, compiling it, doing dynamic visualization and forcing variables.

For the installation of MicroLADDER on Windows, execute the file "setup-mladder-**", where ** is the version number. For Linux, you can install it with the command "sudo dpkg -i mladder-x.x.deb". If the installation doesn't work by lack of dependency it is necessary to execute the command "sudo apt-get -f install".

1.4 MicroDRIVER

This software is used by MicroCONTROL and MicroLADDER for the communication with the PLC. Version 5.3 or a more recent one needs to be installed. For the installation of MicroDRIVER on Windows, execute the file "setup-mdriver-**.exe", where ** is the version number. On Linux, execute the file "mdriver-**.deb", where ** is the version number. On Linux as well, if the installation doesn't work by lack of dependency it is necessary to execute the command "sudo apt-get -f install".

1.5 MicroHMI

This software is complementary to MicroLADDER and used to create HMI (Human-Machine interface) for some PLCs that include a touchscreen (specifically, MicroARMA2, MicroARMA8 and MicroARMA9). For the installation of MicroHMI on Windows, execute the file "setup-mhmi-**.exe", where ** is the version number. On Linux, execute the file "mhmi-**.deb", where ** is the version number. On Linux as well, if the installation doesn't work by lack of dependency it is necessary to execute the command "sudo apt-get -f install".

2 PROGRAMMING IN MicroLADDER

2.1 Getting started

2.1.1 Start and quit MicroLADDER

After installation of the MicroLADDER software, the access to the program can be done from the Start Menu on Windows operating systems or through the Dash on some Linux operating systems, just searching for MicroLADDER in the Search Box. No shortcuts are created on the Desktop by default. Once the program is open, the MicroLADDER main window pops up.

To leave the program just go to the menu bar and do File>Quit. You will be asked to save changes or to discard them before closing MicroLADDER. Temporary files are not saved in MicroLADDER so make sure of saving them manually. You can also leave MicroLADDER by clicking on the Close Button located in the upper right part of the main window on Windows operating systems or upper left hand part of the main window on Linux operating systems.

2.1.2 Using the HMI

When opening the MicroLADDER, the main window of the software pops up (See Figure 1). The following elements can be found on the main window:

Title bar: on the title bar you can find the version of the MicroLADDER installed. In the example of Figure 1, you can see "MicroLADDER v14.5". It tells as well if you are using the online (remote) or offline firmware version.

Tool bar: the toolbar includes the following items:

Page selection.

Add button.

Remove button.

Find button.

Find and replace button.

Connect.

Show variables.

Programming window: this window provides the user the space for programming applications of functions. The user can move through this window by using the scroll bar located on the right side.

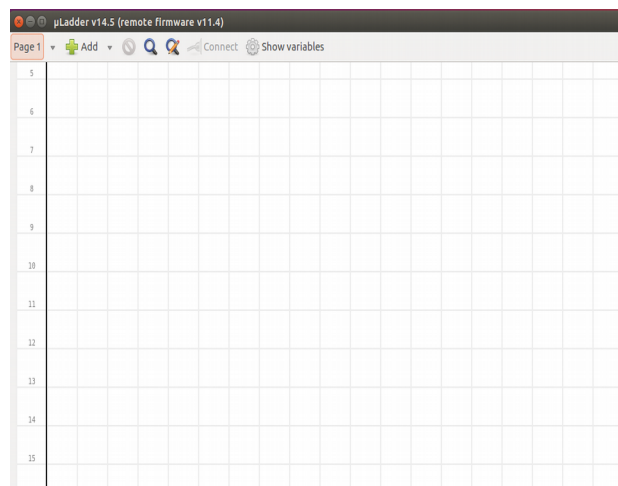


Figure 1 : MicroLADDER main window

2.1.1.1 Tool bar

The tool bar in MicroLADDER appears at the top of the programming window. It includes the following items, shown in Figure 2.



Figure 2 : Toolbar of MicroLADDER

The following items are included in the tool bar:

Page. Select the page to work with from the drop-down list.

Add. Click on this option to include a new item to the current ladder page. Select the item desired from the drop-down menu and place it with the mouse cursor at the desired position by a single left-click. You can also add items from the right-click menu or from the Ladder menu located on the menu bar. When accessing through the right-click menu, make sure you first place the mouse cursor at the desired location before right-clicking, as the item will be placed directly with no need of an additional left-click.

Remove. This button is enabled only when having active an item selected in a ladder page. Once the item(s) you want to delete selected, click on this option. You can also delete the item(s) by selecting this option from the right-click menu or from the Ladder menu located on the menu bar.

Find. Click on this option from a C code page to activate the research bar for code searching. This bar will appear at the bottom of the window. See Figure 4. Write the text to be found on the research bar and activate Match case for case-sensitive search and Wrap around if you want all the matches in the page to be marked.

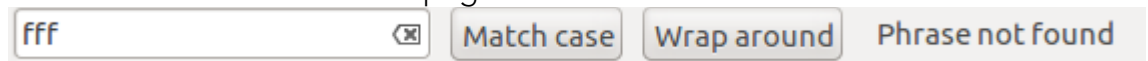


Figure 4: Bar appearing when clicking on "Find".

Find and replace. Click on this option from a C code page and a pop up window will appear. Specify the research criteria and the replacing text. You can either replace a text manually match by match or replace it all. In the case of manual replacing, you can also do it forwards or backwards, by activating or deactivating the option Search backwards.

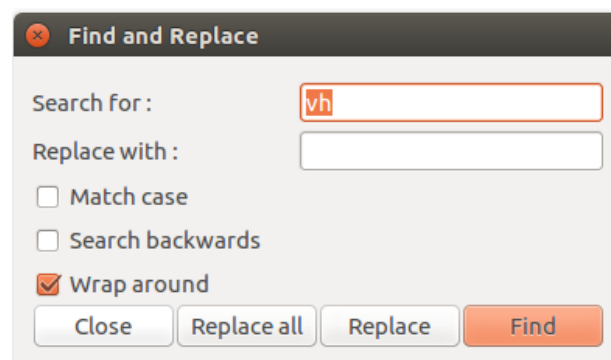


Figure 5: Find and replace window

Connect. Before this step, you need to select the PLC you will use. To do this, go on the title bar, click on Program, then Set program type and select your PLC. You can then click

on the Connect button to connect to a PLC when disconnected or to disconnect it when connected. When connecting, a new window will pop up (see figure 6). Define appropriate settings for the Slave number, Channel and MicroDRIVER's address in order to connect to your PLC. To disconnect, just click on this option while the connection with the PLC is active. You can also access the Connection to MicroDRIVER window from the Communication menu located on the menu bar.

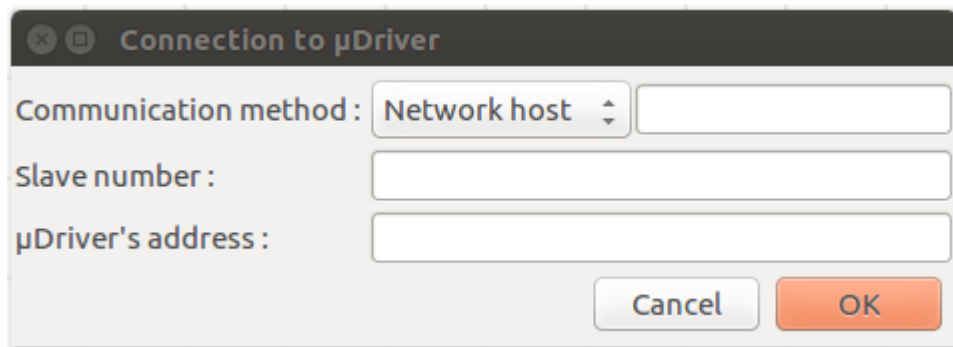


Figure 6: Connection to MicroDRIVER window.

Show variables. Click on this option to open the *variable editor* window. This window contains all the information regarding variables of the program, both system variables and user variables. Use it also to check real time value of variables when communicating with the PLC. You can also access this option from the Program menu located on the menu bar.

2.1.1.2 Menu bar

The MicroLADDER menu bar contains a series of drop down menus that can be used to access the various tools and configuration utilities of the software (See Figure 3)

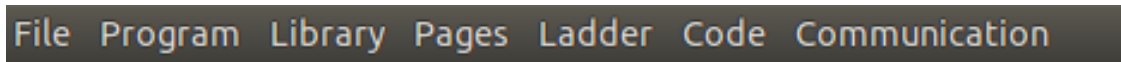


Figure 7. MicroLADDER menu bar.

The following drop menus are included in the menu bar:

File (Alt+F). Common functions are included in the file menu:

New (Ctrl+N). Create a new project. If you already have a project open you will be asked to save or not changes.

Open (Ctrl+O). Open an already existing project. Just look for it in the explorer window that will pop up when clicking.

Save (Ctrl+S). Save changes of the open project with the predefined name and location. With new projects saved as first time, the Save As option will be open. A .lad file will be then saved at the specified location. An asterisk located at the title bar next to the firmware version indicates if the last changes performed on the current project have already been saved (without asterisk) or not (with asterisk).

Save As (Ctrl+A). Save changes of the project choosing the name and location.

Recent projects. The five most recent projects will be additionally shown in this menu, for an easy access.

Quit (Ctrl+C). Exit MicroLADDER. If you have not saved changes manually, you will be

asked for it before leaving the program.

Program (Alt+P).

Compile. Compile a program or bloc function already created. After compilation you will be asked to choose a location where to save it. To compile, the system code must be imported first in the project or be available online.

Set program type. Select the program type to be developed, either a bloc function or a specific PLC program. Be aware that the firmware has to be imported to see all options of program types to be created. If the firmware is not imported, you will only have the Bloc function type available.

Set custom icon. Associate an icon to a function. This icon will appear when imported in the final program, inside the object. It is an asset only for the function block: it displays an image on the block when it is imported in a project and called on a ladder page.

Devices. Gives the device list, splitted in two kinds :

- Main device : Create a list of settings that will allow to configure remounting data to the server.

- Add remote device : Also called a slave. This option allows to set the slave(s) used to create a Modbus network (see section 4 IoBus).

Show variables. Click on this option to open the *variable editor* window. This window contains all the information regarding variables of the program, both system variables and user variables. Use it also to check real time value of variables when communicating with the PLC. You can also access this option by using the *Show Variables* button located on the tool bar.

Note: Setup / Import / Export firmware are options only available when using integrated firmware and not online firmware.

Setup firmware : Configure the options for compilation. Click on this option to change the firmware settings into AUTO-STOP (by default), HTTP, LOG or SNMP.

Import firmware. Before creating a new program, the firmware has to be imported. Click on this option and select the mArm.sys file on the explorer window. Once the firmware is correctly imported, the different options for the program type will be available.

If you have an Internet connection, you don't need to import the firmware as MicroLadder will use online firmware.

Export firmware. Click on this option to export firmware currently imported in the project, to be used in further projects. You may want to use this option to develop a new program using the same firmware, or just to know the firmware version of a PLC.

Remove firmware. Click on this option to remove firmware previously imported.

Setup HMI. Configure the options for HMI.

Import HMI. Click on this option if you have created a HMI with MicroHMI that you want to include in your current project.

Export HMI. Click on this option to export a HMI previously imported.

Remove HMI. Click on this option to remove a HMI previously imported.

Library (Alt+L).

Import function. Import a function to the current project by selecting it from the explorer window that pops up when clicking this option.

Export function. Export a function to the current project by selecting it from the new

window that pops up when clicking on this option.

Remove function. Remove a function from the current project by selecting it from the new window that pops up when clicking on this option.

Pages (Alt+G).

Add Page. Add a new page to your current project. You may choose a Label (otherwise the label will be automatically created by giving a number to the page), the programming language of the new page (Ladder or C) and the options "Call on interrupt" and the use of a Timer (in ms) can be considered as well.

Edit Page. Click on this option if you want to change the properties (label, language, call on interrupt and timer) of a specific page already created.

Copy Page. Activate the page you want to copy and click on this option. Both properties and code will be kept in this new page. The label will be automatically created by assigning the next page number available to the new page.

Move Page. Activate the page you want to move and click on this option. Select in the new window that pops up the new position desired for the active page.

Remove Page. Activate the page you want to remove and click on this option. Be aware that this action will not be undoable.

Page 1 (and next). The different pages available in the project are directly accessible from this menu.

Ladder (Alt+D)

This drop menu will not be accessible from a C page.

Select All (Ctrl+A). Click on this option to select all the elements included in an active ladder page.

Invert selection (Ctrl+Shift+I). Click on this option if you want to invert the current selection in the active ladder page. If no item is selected, all elements will be then selected when clicking on this option.

Cut (Ctrl+X). Once selected the item(s) you want to cut, click on this option. Note that the item(s) selected will not disappear nor change its appearance to show it is (they are) being cut. It (They) will just disappear from its (their) previous position when pasting it (them). You can also cut the item(s) by selecting this option from the right-click menu.

Copy (Ctrl+C). Once selected the item(s) you want to copy, click on this option. You can also copy the item(s) by selecting this option from the right-click menu.

Paste (Ctrl+P). Position the mouse cursor where you want to paste the item(s) previously cut or copied. You can also paste the item(s) by selecting this option from the right-click menu.

Delete (Supr). Once selected the item(s) you want to delete, click on this option. You can also delete the item(s) by selecting this option from the right-click menu or by using the Remove button placed at the Tool bar, which is enabled when having a selection active.

Properties. Click on this option to assign the variable or code associated to the item selected (only one item each time). Variable can be assigned both with the address and the mnemonic. You can also access the item properties by selecting this option from the right-click menu or by double-click on the item. Note that the double click only works if MicroLadder is not connected to the PLC. If connected, the double-click allow to force variable value.

Add. Click on this option to include a new item to the current ladder page. Select the item desired from the drop-down menu and place it with the mouse cursor at the desired position by a single left-click. You can also add items from the right-click menu or by using the Add button placed at the Tool bar. When accessing through the right-click menu, make sure you first place the mouse cursor at the desired location before right-clicking, as the item will be placed directly with no need of an additional left-click.

Code (Alt+O)

This drop menu will not be accessible from a ladder page.

Find (Ctrl+F). Click on this option from a C code page to activate the research bar for code searching. This bar will appear at the bottom of the window. See Figure 4.

Write the text to be found on the research bar and activate **Match case** for case-sensitive search and **Wrap around** if you want all the matches in the page to be marked.

Find and replace (Ctrl+R). Click on this option from a C code page and a pop up window will appear. Specify the research criteria and the replacing text. You can either replace a text manually match by match or replace all. In the case of manual replacing, you can also do it forwards or backwards, by activating or deactivating the option **Search backwards**.

Communication (Alt+C)

Connect. This option will be available when no connection has yet been established. A new window will pop up when clicking on this option. Define appropriate settings for the Communication method, Slave number, Channel and MicroDRIVER's address in order to connect to your PLC. You can also access the *Connection to MicroDRIVER* window by using the Connect button placed at the Tool menu.

Disconnect. This option will be available when a previous connection has already been established. Click on this option to stop communication with the PLC.

Read slave number : Only available when a connection with a PLC has been established. Used to obtain the slave number of a specific slave, when it is planned to connect several serial ones, on the same line.

Write slave number : Only available when one (and only one) connection with a PLC has been established. Used to modify its slave number.

Start program : To start a program when the PLC is on a STOP state.

Stop program : To stop a program when the PLC is on a RUN state.

Reset device : To reset the PLC to its BOOT state. Useful to test its reactions when turned on and off for example.

Reset variables : Does not leave the program, but put all the variables back to their initial value. Useful to simulate a behaviour.

Flash program : Send a program to a PLC by connecting it. It stops the running program to begins the one that has just been done. It is used for a loading by communication, but it is faster to load using a SD card if your PLC have a SD card reader. See section "2.3.2 Compilation and loading to the PLC".

Note:

- It is possible to learn the corresponding key associated to shortcuts Alt+key for the direct access to each drop menu on the menu bar by clicking on Alt while having the menu

bar active or visible. Letters associated to each key will appear underlined.

- When using Unity in Ubuntu operating systems, some problems may be experienced in the appearance of the menus. Menus that should appear blocked (Ladder drop menu when programming in a C code page and Code drop menu when programming in a ladder code page) can appear active. However, their functions are deactivated even if it seems possible to click on them.

2.1.1.3 Programming window

The programming window is the big blank space located under the tool bar used for the development of programs. Two types of programming windows can be displayed: the ladder window and the C window (see Figure 8).

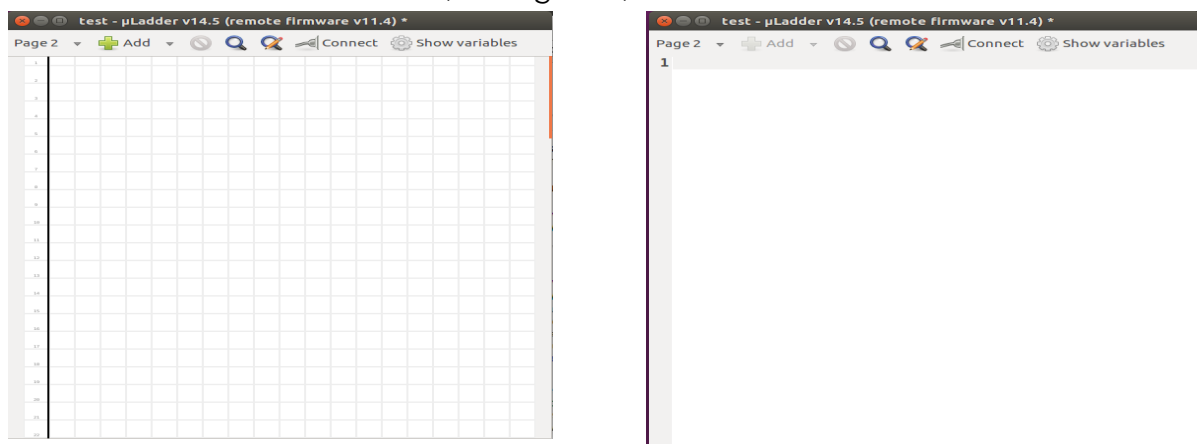


Figure 8. MicroLADDER programming windows: ladder (left) and C (right).

2.1.1 The programming languages

2.1.1.1 Ladder

Ladder logic language (or just **ladder language**) is a programming language that represents a program by a graphical diagram based on the circuit diagrams of relay-based logic hardware. It is mainly used to develop software for PLCs used in control applications. Ladder logic is widely used to program PLCs, where sequential control of a process or manufacturing operation is required. Ladder logic is useful for simple but critical control systems or for reworking old hard-wired relay circuits.

Ladder logic can be thought of as a rule-based language rather than a procedural language. A "rung" in the ladder represents a rule. When implemented with relays and other electromechanical devices, the various rules "execute" simultaneously and immediately. When implemented in a programmable logic controller, the rules are typically executed sequentially by software, in a continuous loop (scan). By executing the loop fast enough, typically many times per second, the effect of simultaneous and immediate execution is achieved, if considering intervals greater than the "scan time" required to execute all the rungs of the program. Proper use of programmable controllers requires understanding the limitations of the execution order of rungs.

While ladder diagrams were once the only available notation for recording programmable

controller programs, today other forms are standardized in IEC 61131-3.

2.1.1.2 C

The **C programming language** was originally developed by Dennis Ritchie between 1969 and 1973 at Bell Laboratories. C is very suitable for actually writing system level programs because of the simplicity of expression, the compactness of the code and the wide range of applicability. It allows the programmer a wide range of operations from high level down to a very low level approaching the level of assembly language. The flexibility available is wide what makes it a perfect programming language for the development of PLC programs of high degree of complexity.

Users of MicroLADDER are expected to know basic concepts of this language. The point of this user manual is to help the user to create PLC programs with MicroLADDER, so no specific information is being provided regarding C language.

2.1.2 Importing firmware

With this version of MicroLADDER, the firmware is online and works automatically with your system. No import is needed, but an internet connection is.

2.1.3 Connecting with the PLC

Establishing a connection with the PLC is necessary when loading a program through a port and when monitoring variables.

2.1.3.1 MicroDRIVER

MicroDRIVER is the software in charge of establishing connection between the PLC and the different applications in the system. It can be considered as a “black box” of communications. It is independent to the communication protocol of the system.

The user will not be aware of its operation as it runs in the background. However, if MicroDRIVER is not installed, it would be impossible to establish connection with the PLC.

2.1.3.2 Establishing connection

To establish connection from MicroLADDER with the PLC, you have to click on the Connect option and establish connection through MicroDRIVER, as said before. Once selected this option, a connection window pops up (see Figure 9).

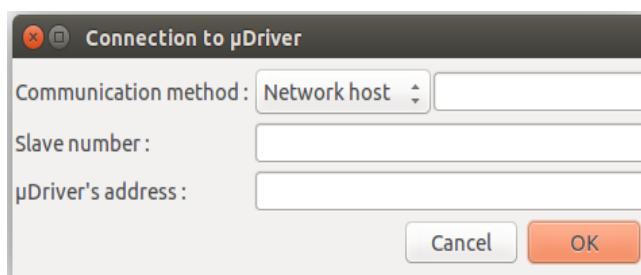


Figure 9. Connection to mDriver window.

For the connection with MicroDRIVER you have to determine the following items:

Communication method: select your option between:

- network host: connection is made by the network. Put the IP address or domain name
- serial port: choose a serial port in the list and its configuration
- channel: obsolete functionality

Slave number: this number identifies the PLC inside a network of PLCs. You do not need to set the slave number if you are programming a single PLC. It will be determined after the first connection.

MicroDRIVER's address: MicroDRIVER is the server application. It can run locally or on a remote computer. If it runs in a remote computer you have to specify the IP address of this computer.

The configuration for the connection to MicroDRIVER may differ depending of the way you are connecting with the PLC from your computer. Most typical configuration parameters can be seen in Table 1.

Table 1. Typical configuration parameters for connecting to MicroDRIVER.

Type of connection	Configuration	
PC – Ethernet - PLC	Communication method	Network host IP address of PLC specified in the SD card file main.cfg. Typically: 192.168.0.123
	Slave number	Typically 1. you don't need to change slave number when network host connection.
	MicroDRIVER's address	Nothing
PC – Serial – PLC	Communication method	Serial port Select your port number in the list. Typical configuration is "38400 bauds", "8N1"
	Slave number	Slave number of PLC port specified in the SD card file main.cfg. Typically 1. When having more than one PLC connected in serial to the same RS2485 port, the slave number must be specified.
	MicroDRIVER's address	Nothing
PC – Ethernet – µFox – Serial – PLC	Communication method	Channel Select channel number for µDriver (default 1:COM1, 2,COM2, etc). Typical configuration is "38400 bauds", "8N1"
	Slave number	See above
	MicroDRIVER's address	Address of the system communicating

		directly with the PLC. In the case of using a MicroFOX, the MicroDRIVER address is in this case the MicroFOX address. Make sure of checking this IP address.
--	--	--

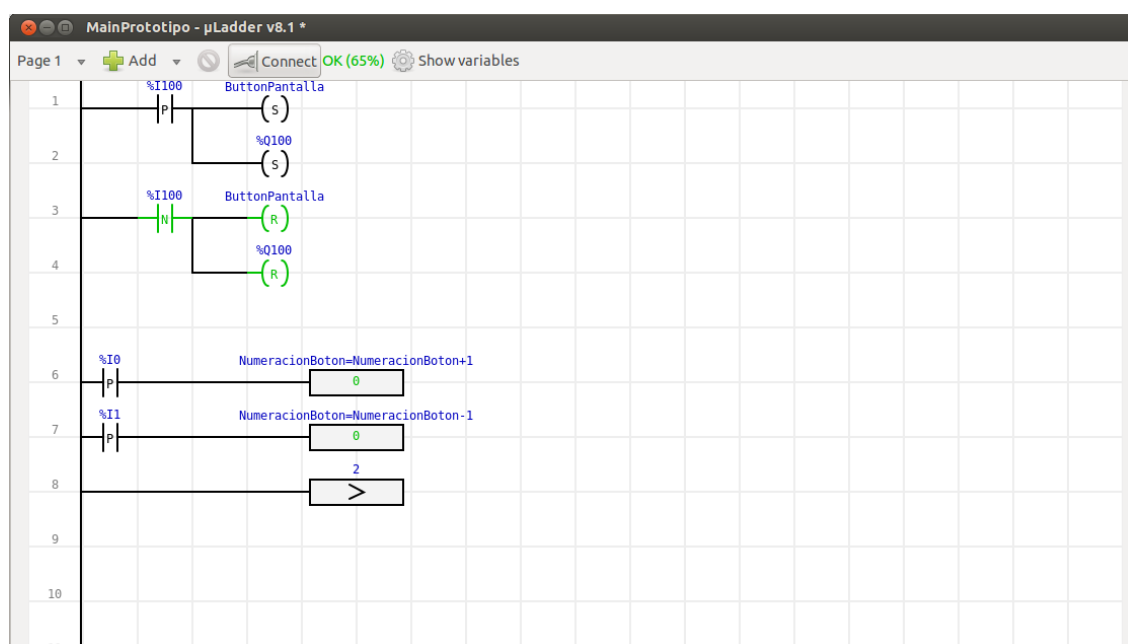


Figure 10. Screen capture of the MicroLADDER main window with the PLC connected and communicating.

Once the connection parameters correctly established, click on OK and the connection will be established. You can make sure that the connection has been established checking the following:

- The **Connect** button of the Tool bar is pressed. (If you click on it you directly disconnect again).
- The Connect option of the Communication drop-down menu of the menu bar is unable and the Disconnect option enabled.
- A green message "OK" has appeared on the tool bar.
- Active instructions of the program appears painted in green as well (See Figure 10).

Note:

A variable percentage is shown in green near the state of the connection. This is just additional information which shows the quality of the connection with the PLC: it represents the percentage of frames that have been correctly transmitted. The statistics calculation is made when MicroDRIVER starts.

2.1.4 Using pages

The number of pages able to be used in any MicroLADDER program is unlimited, being each ladder page limited to 100 lines (there is no line limitation in C pages though). The

limitation in the number of lines inside a ladder page forces the user to structure the program in different pages and/or making use of functions, what facilitates the interpretation of the code.

Page 1 is called at each PLC cycle. It must then call the other cyclic page. It is also possible to call the pages on interruption (period of call set in %SW25 for a precise and fast period), or on timer (settings in the property of the page for a slower and less precise period).

The user can decide if a specific page should be programmed either in ladder or in C. Pages of different programming languages can be called and combined easily. This provides the user a great flexibility and an easy structure interpretation when creating of applications.

Make sure that there is no code already generated in a page when changing it from C to ladder or vice-versa, as it will be lost.

2.1.5 Creating a page

A new MicroLADDER project includes by default a single page, Page 1, which is by default configured as ladder code. You can easily create an additional page by adding a new page or by copying an existing page.

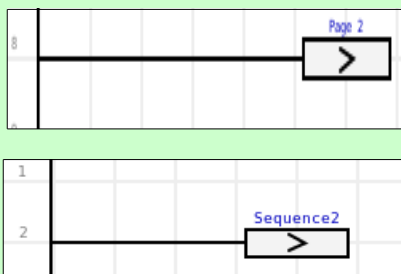
The number of pages to be used in a MicroLADDER project is unlimited, and pages can interact between them by using the *call* command. This command is an object in ladder code and a function in C code.

2.1.6 Call of a page

Page 1 is called automatically whereas the other pages must be called on interruption, timer, or with call orders in C or ladder.

A page can be called from another page inside a program, no matter the type of code used in each page. That means a ladder page can be called from a C page and vice-versa. For calling a page in ladder, just add a Call command either from the ladder drop-down menu located on the menu bar or by the right-click menu.

Inserting a Call command with ladder.



If the called page does not have a label, it would be called by its number. In this example, Page 2.

If the called page has a label, it can either be called by its number or by its label, but in the ladder diagram, the label will be shown. As it happens in this example, where the page called its named "Sequence2".

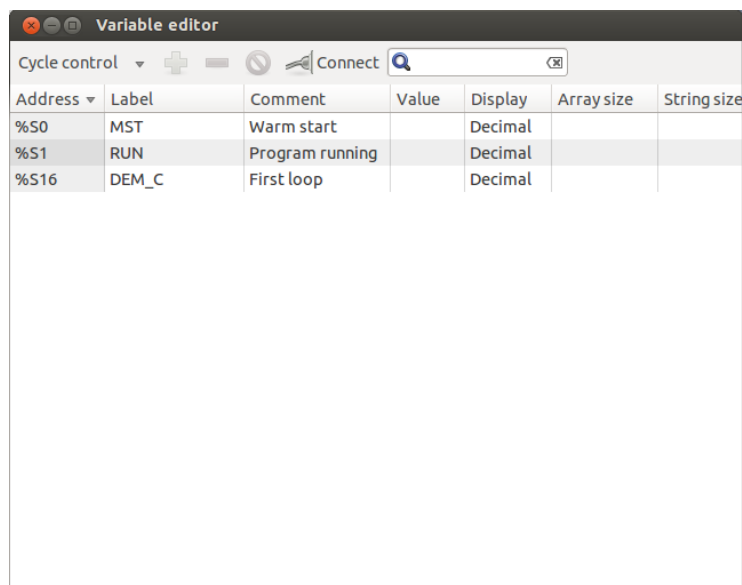
In a page written in C language, another page can be also called either by its number, or by a mnemonic.

```
Inserting a Call command with C.  
page_20; //For calling page number 2  
init() ; //For calling a page named "init".
```

2.1.7 Variable editor

Variables in MicroLADDER, as in ordinary computer programming, are storage locations with an associated symbolic name which contains some known or unknown quantity of information.

Variables available in a MicroLADDER program can be shown and edited from the variable editor (see Figure 11). You can access the variable editor by clicking on Show variables.



Address	Label	Comment	Value	Display	Array size	String size
%S0	MST	Warm start		Decimal		
%S1	RUN	Program running		Decimal		
%S16	DEM_C	First loop		Decimal		

Figure 11 : Variable editor table

2.1.1 Variable editor tool bar

The variable editor tool bar appears at the top of the variable editor window. It includes the following items, shown in Figure 12.

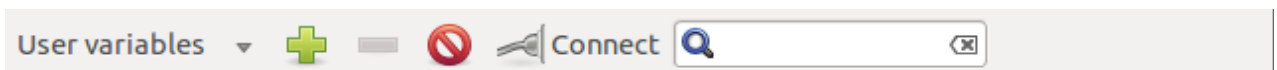


Figure 12. Variable editor tool bar.

The following items are included in the variable editor tool bar:

User variables/System variables/User Tables created. Select the type of variables you want to be shown from the drop-down list: system variables, user variables or one of the tables that you may have created.

Add. Click on this option to include a new variable to the current MicroLADDER project. Give this new variable a name (either an address or a label) and define its properties in the

new properties window that pops-up.

Remove from table. Remove a variable from a table, but not from the project.

Delete. Select a variable (or variables selecting them with Ctrl or Shift) and click on this option to remove them. Variables could not be deleted when being used inside the program.

Connect. Click on this option to connect to a PLC when disconnected or to disconnect it when connected. When connecting, a new window will pop up when clicking on this option. Define appropriate settings for the Communication method, the Slave number, and MicroDRIVER's address in order to connect to your PLC. For disconnecting, just click on this option while the connection with the PLC is active. You can also access the *Connection to MicroDRIVER* Window from the Communication menu located on the variable editor menu bar.

Find. Use this search bar for looking for a specific variable either by its address (as for example, "%S1") or by its label. The search is not case sensitive. Make sure you are in the correct type of variable (if you are looking for a cycle control variable and you have the user variables selected, the search will return no results).

2.1.1.1 Variable editor tool bar

The Variable Editor has its own menu bar with a series of drop-down menus included (See Figure 13).

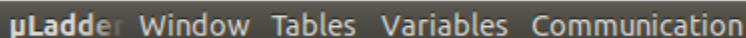
A screenshot of the variable editor menu bar. It is a dark grey horizontal bar with the text "µLadder Window Tables Variables Communication" in a light grey font. The "µLadder" part is slightly larger and more prominent than the others.

Figure 13. Variable editor menu.

These drop-down menus contain the following options:

Window

Close. You can close the Variable Editor by clicking on this option or just clicking on the upper right close button (on Windows operating systems) or on the upper left close button (on Linux operating systems).

Tables

Add table. In MicroLADDER, a category or group of variables is called table. By default MicroLADDER considers two tables: user variables and system variables. New tables can be defined by clicking on this option and giving a name.

Rename table. Click on this option to rename a Table. Default tables cannot be renamed.

Remove table. Click on this option to remove a Table. Default tables cannot be removed.

Import variable into table. Click on this option to import a CSV variable file to the current MicroLADDER project.

Export table of variables. Click on this option to export a table of variables into a CSV file from the current MicroLADDER project.

Import values into tables. Click on this option to import values into a table from the current MicroLADDER project.

Export table values. Click on this option to export values into a table from the current MicroLADDER project. It is only possible to export in a csv file.

System variables. A drop-down menu with the different system variable tables appears

when selecting this option. Variables included in each category selected will be shown in the variable editor main window.

User variables. Variables created by the user will be shown in the variable editor main window.

User tables. To sort the variables in the tables. Select the appropriate one in the list.

Variables

Add. Click on this option to include a new variable to the current MicroLADDER project. You can also add a variable through the tool bar.

Select unused variables. Click on this option to select automatically all the variables that are not used in the program. Gives a better visibility to delete them for example.

Properties: Click on this option to set-up the variables. Shortcut : double-clicking on a variable from the list when not connected.

Set value. Some values can be forced manually when connecting with the PLC. Use this option to set a variable to a specific value in real time. Shortcut : double-clicking on a variable from the list when connected.

Add to table. Select one or more (with Ctrl or Shift) variables and add it into one of the created tables.

Remove from table. Click on this option to remove one or more (with Ctrl or Shift) variables from a table, but not from the project.

Delete. Select one or more (with Ctrl or Shift) variables and click on this option to delete them. Variables could not be deleted when being used inside the program. You can also delete them from the tool bar.

Cross references : Click on this option to show all the spots where a specific variable is used.

Communication (Alt+C). This drop-down menu is similar to the Communication drop-down menu of the MicroLADDER main menu bar.

Connect. This option will be available when no connection has yet been established. A new window will pop up when clicking on this option. Define appropriate settings for the Slave number, Channel and MicroDRIVER's address in order to connect to your PLC. You can also access the Connection to MicroDRIVER Window by using the Connect button placed at the Tool menu.

Disconnect. This option will be available when a previous connection has already been established. Click on this option to stop communication with the PLC.

Read slave number. To know more about this option and the following ones, see 2.1.1.2 explaining the menu bar, see « Communications » tab.

Write slave number.

Start program.

Stop program.

Reset device.

Reset variables.

Flash program.

2.1.2 Variable properties


The following information is shown about available variables in the variable editor:

General tab

Sirea, a company specialized in the field of industrial automation and electrical energy.

Sirea, 1 rue Jean Perrin
ZI de Mélou - 81100 Castres

Phone : +33 (0)5 63 72 93 92

 www.sireagroup.com
 contact@sirea.fr

Follow us on
   

Type: Type of variable. For better understanding prefixes used for the address of variables, see section 2.1.3.

Address: is the memory address of the variable in the PLC. For better understanding prefixes used for the address of variables, see section 2.1.3.

Label: is the mnemonic given to the variable.

Comment: short explanation of the variable.

Value: current value of the variable.

Formatted value: the value with its unity (v as volt for example) and the number of digits after the desired coma on floating number (see "format" and "precision").

Array size: in the case of having an array, number of items allocated in the array. Otherwise, this information is empty.

String size: in the case of having a string, maximum number of characters allocated for the string. Otherwise, this information is empty.

Programming tab

Init value: value loaded on the variable when starting the application or under initialization request.

Saved: if the PLC has a saved memory, you can select this option and the value will be kept during power interrupt. The user can determine the instant for the backup to EEPROM or FRAM.

Timer (in ms): time period for the variable to be decreased from 1 to 0

Global: this property can be used with functions. It allows the variable keeping its value between two calls.

Parameter: used for the declaration of function variables. See section on 2.4 *Creation of a function* for further information. "Internal" is used for a main application variable.

Position: used to define order of input/output in a function.

Label: used to give a name to a variable when it appears in a function.

Input / Output tab

Configuration: parameter setting for the analog inputs

Bound variable: Only for input/output variable that correspond to connected elements. Associates a user variable to a material variable.

Invert state: Only available when using a boolean bound variable. Allows to associate a variable but reversing its value.

Scaling: Only available when using a no boolean bound variable.

It is possible to scale values, and to do so, specify a scale of raw value and associate it a scale of scaled value.

For example : if 0-20000 (for 0-20mA) \Leftrightarrow -20°C-+80°C, set Min raw value=0, Max raw value=20000, Min scaled value=-20, Max scaled value=80 and associate a bound variable with type MF.

Min. raw value: minimum raw value for analog input

Max. raw value: maximum raw value for analog input

Min. scaled value : minimum scaled value for bound variable

Max. scaled value : maximum scaled value for bound variable

Communication tab

Sirea, a company specialized in the field of industrial automation and electrical energy.

See section 4.3 for lobus communication setting

Remote access:

None : no communication for this variable

Read only : Communication cycle is a periodic read frame. The variable in the PLC is a copy of the variable in remote device.

Write only : Communication cycle is a periodic read frame. The variable in the remote device is a copy of the variable in PLC.

Read / write : Communication cycle is a periodic read frame and a write frame when necessary. Each time the variable change in PLC, the value is written into remote device. Each time the variable change in the remote device, the value is updated into the PLC.

Remote address: Address of variable in remote device. The address must specify the index of the remote device with ".index " syntaxe at the end. Example : "%MW2.3" read/write holding register n°2 in remote device n°3.

Invert state: Only available when the remote address is a boolean type. Allows to associate a variable but reversing its value.

Scaling: It is possible to scale values, and to do so, specify a scale of raw value and associate it a scale of scaled value.

For example : if 0-100 in remote device \Leftrightarrow 0-10 in PLC, set Min raw value=0, Max raw value=100, Min scaled value=0, Max scaled value=10. Thus, when value is 233 in remote device, value is 23.3 in PLC and vice versa.

Min. raw value: minimum raw value in remote device

Max. raw value: maximum raw value in remote device

Min. scaled value : minimum scaled value in PLC

Max. scaled value : maximum scaled value in PLC

Logging tab

Alarm condition : set if the condition for the alarm will be inferior or superior to the threshold.

Alarm threshold : define a condition with which an alarm will be turned ON. An alarm is a state, and an event happens and is created every time that a threshold is passed.

Alarm apparition label : text that will appear when the alarm condition is present.

Alarm disparition label : text that will appear when the alarm condition disappears.

Event condition : set if the condition for the event will be inferior or superior to the threshold.

Event threshold : define a condition with which an event will be turned ON.

Event label : text that will appear when the event condition is present.

Logging type : used to archive the value curves in time. There are two ways to do it :

- standard : takes a value with a regular interval
- averaged : takes an average of value in a given period of time

Logging delay (in seconds): the archive delay depends on the type of logging.

- standard : specified time, 1 min for example
- averaged : specified period of time

Logging threshold : threshold after which it will save a value.

Remote server tab

Access : to decide if the value must wind to the server. Three options:

Sirea, a company specialized in the field of industrial automation and electrical energy.

- private : value will not wind
- read only : value will wind
- read and write : value will wind and it is possible to modify it.

Label : only available when an "access" is set. Text that can be added to a name.

Display server tab

These parameters are used in GUI (MicroHMI), in the server (MicroServer) and in the display of the formatted value in MicroLADDER when connected to the PLC. Different options available below :

- **Format** : to set the format of the value displayed. "%v" means the value . Example : "%v°C" will display value followed by unit "°C".
- **Float precision**: when having float variables, the number of digit after the decimal point can be determined. -1 means no limit.
- **Min. display value** : minimum value to be displayed of a variable for synoptics, gauges, etc. in MicroHMI.
- **Max. display value** : maximum value to be displayed of a variable for synoptics, gauges, etc. in MicroHMI.

2.1.3 Type of variables

Variables considered in a MicroLADDER program can be divided into two main tables: system variables and user variables.

System variables

They are inherent to the PLC. Their properties cannot be modified, so the variable editor will only show their real time value, but no modifications of their properties can be performed.

These variables can be additionally classified into 9 tables:

System infos : %SW13,%SW14,%SW23,%SW24

Cycle control: %S0, %S1, %S16

Date&Time: %S5-%S8, %S24, %SW5-%SW12

Ethernet: %S25, %SW96-%SW137

Wi-Fi : %S26, %SW91-%SW95,%SW138-%SW175

Radio : %SW200-%SW211

I/O settings: %S11, %S12, %SW26-%SW32

Interrupt: %SW25

Serial ports: %SW34-%SW81

System infos: %SW13, %SW14

Variable setting: %S11, %S12, %SW26-%SW32

Watchdogs: %S15, %S20-%S22, %SW0-%SW2, %SW4

User variables

The user may need additional variables for a program created in MicroLADDER. If no variables have been created by the user yet, no variables will be shown when selecting **User variables** in the Variable Editor. Variables already created in the program (both ladder or C pages) are created by the MicroLADDER software and thus they will be shown

in the Variable Editor. However, a manual entry is needed for the declaration of arrays. See next section on creation of variables to see how to manually create variables from the Variable Editor.

Creating variables

Variables already entered in the program are automatically created by the MicroLADDER software, but for the declaration of arrays a manual entry is needed.

For creating variables manually click on Add, give the variable a name and determine its properties (see section 2.1.2) on the new window that pops-up (see Figure 14).

If the variable is declared alone, the properties of the variable have priority on the table ones. It is then better to use %MW10[1] over %MW11 which will declare a variable and thus properties that will have priority on the table ones.

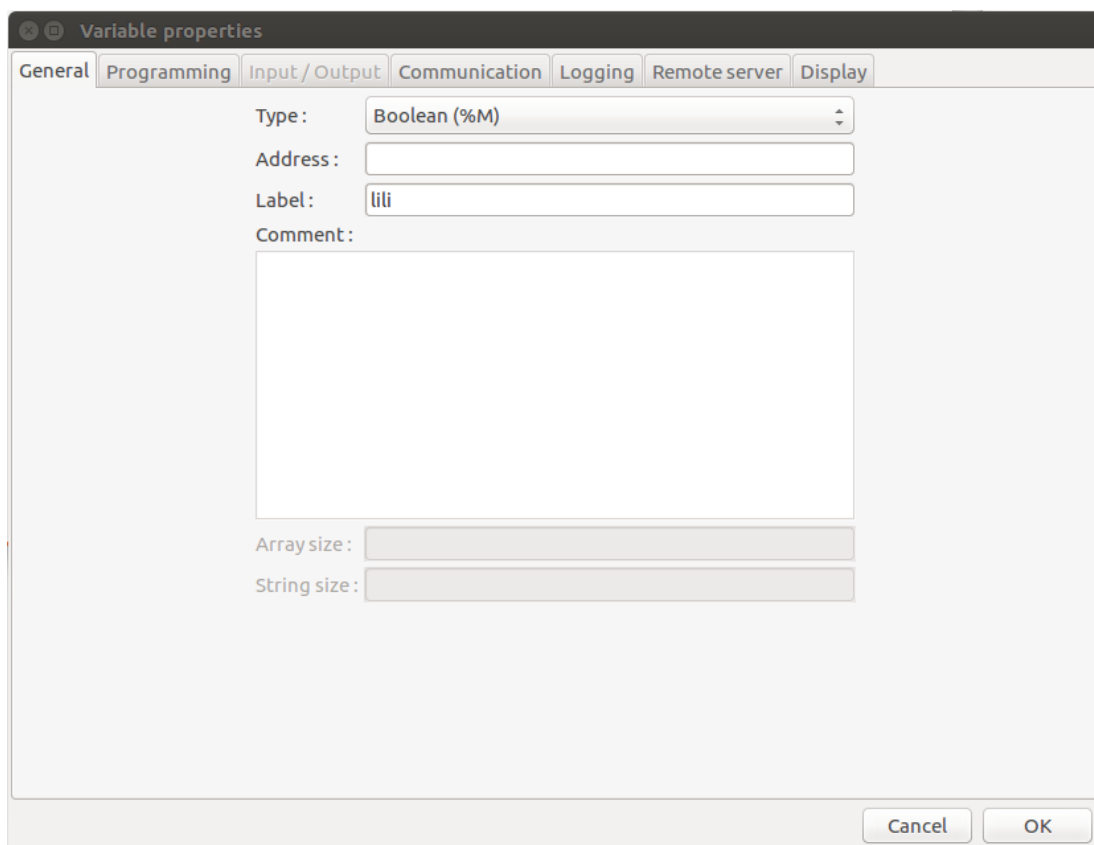


Figure 14.Variable properties window.

Variable identification:

Variable is identify by a combination of type and address.

For example if type is boolean and address is 10. Variable is identified as %M10.

- Type : Select a type between :
 - Boolean : variable going from 0 to 1
 - Short : variable going from 0 to 65535

- Long : variable going from about minus 2 billion to plus 2 billion
- Float : floating value that can be positive, negative and with coma
- String : of characters
- Binary input : input where there is current ON 0 or OFF 1
- Analog input : input where there is current with variable going from 0 to 65535
- Binary output : output that turns ON 0 or OFF 1
- Analog output : output that changes the state of an object (besides from ON or OFF, like an acceleration for engine for example)
- System bit : for a data out of the system, with a numeric or boolean value.
- System word idem : for a data out of the system, with written value.

- Address :

Address is the variable offset in the PLC memory. Address must be unique for each type. For example, it's not allowed to set 2 variables %M0 but a variable %M0 and an other %MW0 is allowed.

The definition of an address is not required. In that case a compilation will locate the variable at a free space within the variable area of the same type. It is necessary if a variable needs to go up to a server or if the PLC must be set to be a Modbus slave, to make an other device come and read/write this variable.

It is possible to export variables in a CSV file or to import from a CSV file. During these operations, a chat box allows to precise the structure of the CSV file.

2.1.1 Using variables in a program

Variables can be referred to in MicroLADDER program both by calling them through their address or label (=mnemonic). However, if they have a label, a ladder page will always identify items included in a program with the mnemonic.

Variables can be exported to a CSV file and imported from a CSV file for their use in further programs.

To export variables, go to Tables>Export table variables. A dialogue box allows to specify the structure of the CSV file during the execution of these processes. It is preferable not to change the default structure to avoid possible import problems. If you change it when exporting, make sure that you respect the same order when importing variables.

Variable import is performed by overwriting current variables, so make sure you do not have in your current project a variable with the same address or mnemonic, as it will be directly replaced.

2.1.1 Variable multi-edition

It's possible to edit properties of several variables together instead of editing each variable separately.

- Select variables with Ctrl or Shift. Then go to properties from the toolbar

Variables>Properties or the right-click menu.

- Properties that have same value on all variables appear selected and framed in blue while properties that have different value appear unselected and grey.
- Select properties you want to change and set the new value. All properties selected will be applied to all variables. Properties not selected will keep unchanged.
- Validate with OK.

Warning : don't change properties that must be unique (label, couple type/address)

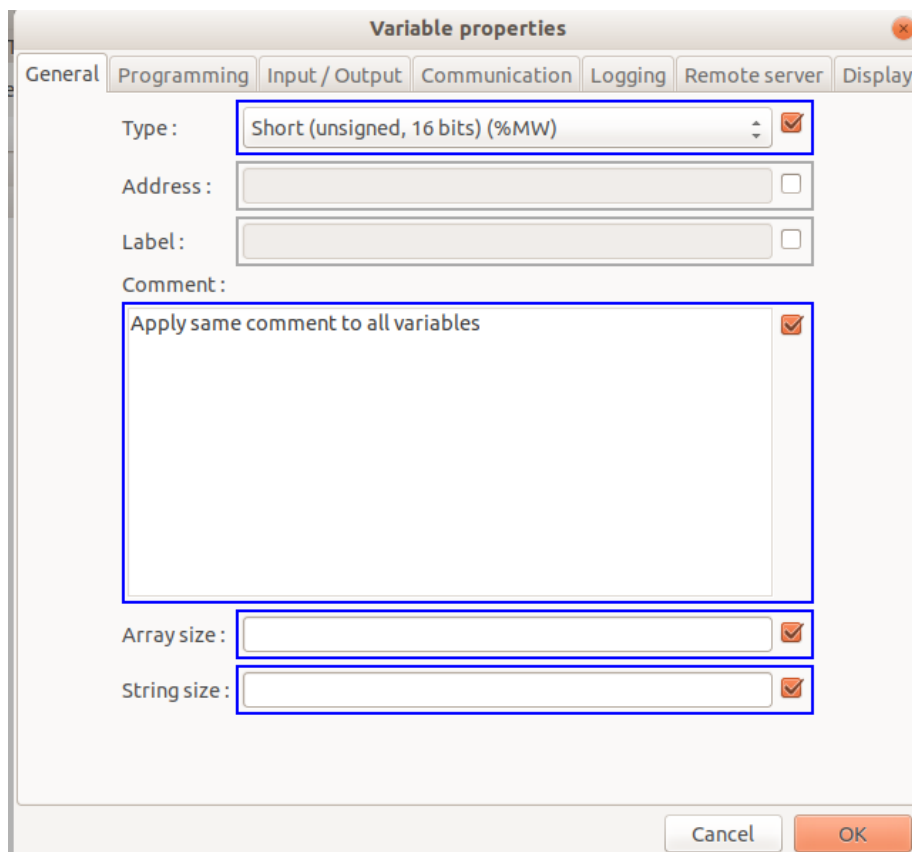




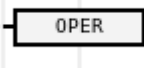






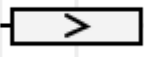

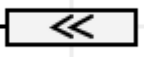

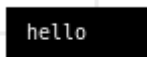
Figure 15. Multiple edition.

2.2 Objects available in Ladder

Basic elements or items available in MicroLADDER for the creation of programs are shown in Table 6. As a program written in ladder can be written as well in C, equivalence between ladder items are additionally shown in this table.

Table 6. Typical items available in ladder language.

Item: symbol and name	Description
 Open contact	<p>It is activated when the value of the variable represented (input, internal variable or system bit) is 1. It is deactivated when its the value is 0.</p> <p>It can be used either with a binary variable, or as a comparison of the type <code>%MW0>0</code> with integer, long or float variables. That means every value of the variable different to 0 will activate the contact.</p>
 Closed contact	<p>It is activated when the value of the variable represented (input, internal variable or system bit) is 0. It is deactivated when its the value is 1.</p> <p>Same remarks as those already done for open contact can be done.</p>
 Rising edge	<p>It is activated when there is a change in the value of the variable represented (input, internal variable or system bit) from 0 to 1.</p> <p>Each edge object is managed by an internal variable independent from the variable internally used:</p> <p>The edge is valid exactly during a complete cycle: if the variable moves after the edge, the edge is easily seen on the next cycle.</p> <p>Possible bug with indexed bits, if the index has varied from one cycle to the next cycle.</p> <p>1 byte is used by the edge object</p> <p>Complex expressions and word bits can be managed.</p>
 Falling edge	<p>It is activated when there is a change in the value of the variable represented (input, internal variable or system bit) from 1 to 0.</p> <p>Same remarks as those already done for rising edge can be done.</p>
FUNCTION	<p>When having a function imported to the current project it can be inserted as a ladder instruction. Just define input variables and connect the correct outputs.</p>
 Operation	<p>Operations between variables can be defined with the use of the Operation instruction.</p>
 On	<p>It is activated when the combination on the left results into 1. Its activation means that it has a value logic 1.</p>
 Off	<p>It is activated when the combination on the left results into 0. Its activation means that it has a value logic 0.</p>
 Set	<p>It sets the variable associated to 1. The variable can only be set to 0 with a Reset. It is usually used for bits storage.</p>

Item: symbol and name	Description
 Reset	It sets the variable associated to 0. It deactivates a variable previously activated with a Set.
 Call	It calls a page and executes its code. Then it turns back to the next line where this call was located.
 Jump	It enables to jump ahead some program instructions to a certain anchor.
 Back	It enables to jump back some program instructions to a certain anchor.
 Anchor	It serves as anchor for Jump and Back instructions (through the Anchor Index) and as program text comment.
 Comment	It allows to add a comment zone inside the project.
Insert blank lines	It adds blank lines to set space for other objects.
Remove blank lines	It takes off useless spaces.
Paste	It pastes objects copied previously.

For connecting items with each other, just double click in the background line of the ladder grid where a connection needs to be established or click and drag from one item to another (or to a line). For disconnecting items, right-click and select Disconnect or double click on the connection you want to delete.

2.3 Creation of a program

In order to better understand how to create programs with MicroLADDER, some examples are shown in the current section.

2.3.1 First program in ladder

For a better understanding of the programming in MicroLADDER, a simple exercise is being done. A PLC is going to be programmed with a very simple application. The system to be programmed consists of the following elements:

- PLC MicroARM-A1

- 1 digital input at the %I100 address: push button
- 1 digital output at the %Q100 address: a LED
- The back-light of the LED screen is lead by the %Q0 HMI input. For those addresses see MicroARM-A1 manual user.

The program to be created has to do the following things:

- The LED will only be ON when pushing the push button. Otherwise, it will always be OFF.

- When the LED is ON (that means the push button is pushed), the MicroARM-A1 screen has also the back-light ON and the text shown is the following:

- First line: "LED is ON" positioned on the left side of the screen
- Second line: "Screen is ON" positioned on the left side of the screen

- When the LED is OFF (that means the push button is released), the MicroARM-A1 screen has the back-light OFF and the text shown is the following:

- First line: "LED is OFF" positioned on the right side of the screen
- Second line: "Screen is OFF" positioned on the right side of the screen

For writing this program, the function WriteText is being used. Figure 16 shows this program already implemented in MicroLADDER.

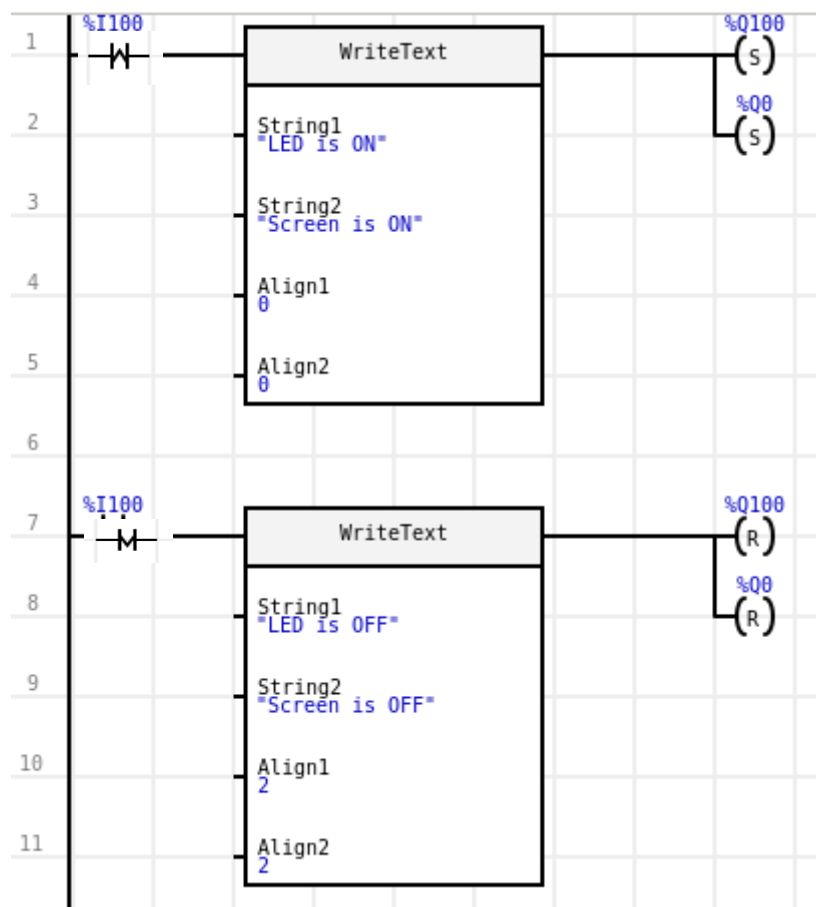


Figure 16. First program created with ladder.

Ladder elements considered for the program, can be described as follows.

Objective	Digital interpretation	Ladder
When pushing the push button...	%I100 changes from 0 to 1	Rising edge
When releasing the push button...	%I100 changes from 1 to 0	Falling edge
... LED or back-lightning ON.	%Q100 or %Q0 is set to 1	Set
... LED or back-lightning OFF.	%Q100 or %Q0 is set to 0	Reset

2.3.2 First program in C

The similar exercise is being performed in the current section, so the same context has to be analysed.

The C code expected would be the following:

```
if (%I100 == 1)
{
    %Q100=1;
    %Q0=1;
    WriteText("LED is ON","Screen is ON",0,0);
}
else
{
    %Q100=0;
    %Q0=0;
    WriteText("LED is OFF","Screen is OFF",2,2);
}
```

2.3.1 Combining Ladder and C (including an example)

As already mentioned in previous sections, it is also possible to combine a program with both ladder and C code by structuring it in different pages. The same exercise as the one explained before is being divided into two pages as follows:

- A first page includes the program that shows the text on the screen when pushing the push button.

- A second page includes the program that activates the LED and the back-light of the LED screen.

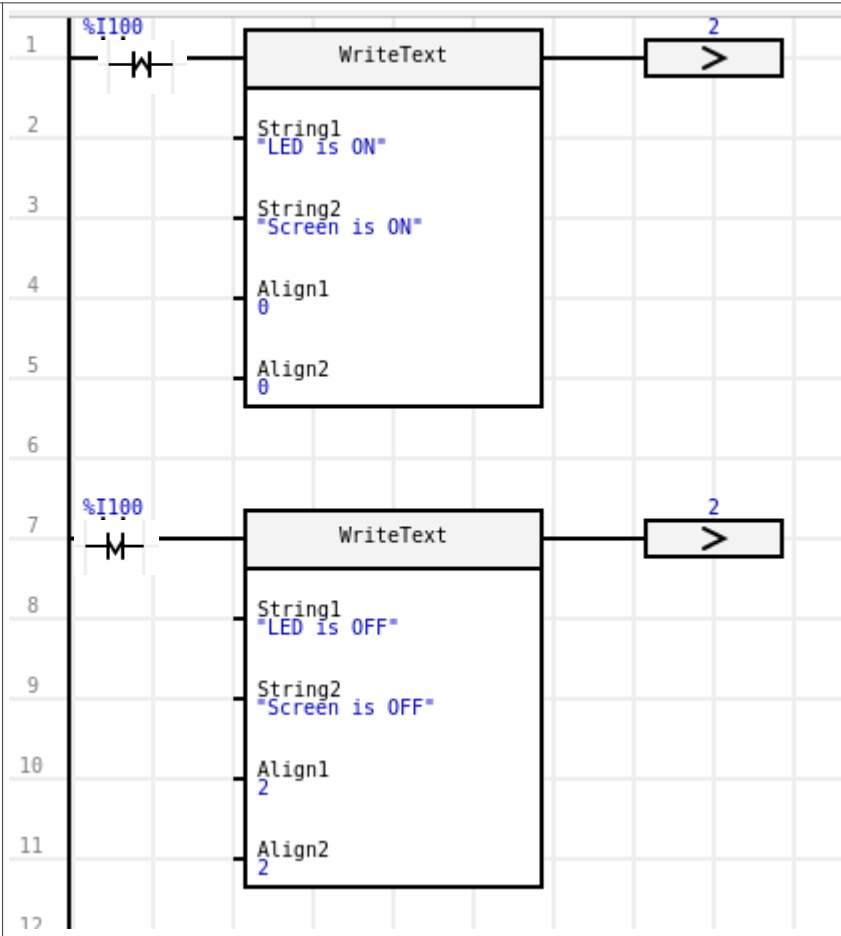
This is being done in two ways as well:

- First, the main program is being written in ladder and it calls a C page.

- Second, the main program is being written in C code and it calls a ladder page.

Taking into account these considerations, the new program will be as it is shown in Table 7 for the first case.

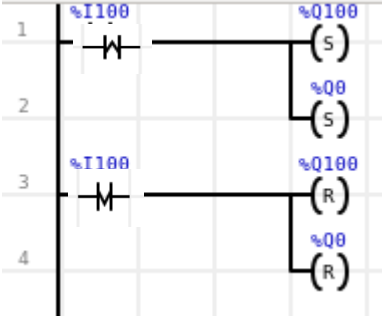
Table 7. Page structure and code for the combination of the first program in MicroLADDER combining ladder and C code.

Page 1	
Page 2	<pre>if (%I100==1) { %Q100=1; %Q0=1; } else { %Q100=0; %Q0=0; }</pre>

Taking into account these considerations again, the new program will be as it is shown in Table 8, for the second case.

Table 8. Page structure and code for the combination of the first program in MicroLADDER combining C and ladder code.

Page 1	<pre>if (%I100 == 1) { WriteText("LED is ON", "Screen is ON", 0,0); }</pre>
--------	---

	<pre> page_2(); } else { WriteText("LED is OFF","Screen is OFF", 0,0); page_2(); } </pre>
Page 2	

2.3.2 Compilation and loading to the PLC

After the program has been created, it has to be compiled before charging it into the PLC. It is advisable to save the project before compiling it by doing File>Save as from the menu bar. You can compile then the program selecting Program>Compile.

You may specify where do you want to save this new files:

- If your PLC includes a SD card reader and you wish to load the program by using this mean, copy files to the SD card directly. The program will be automatically loaded when inserting the SD card into the PLC.

- If you do not want to load the program with the SD card or your PLC does not allow this option, just select a folder in your PC. You will have the to load the .hex file to your PC via MicroCONTROL software.

If there are any errors in the program a warning message will pop up. Please check you program before compiling it again.

If the compiling is successful the files main.hex, main.cfg, loadmain and size.txt will have been created and a new window will pop up to confirm that the compilation was successful. For further information on files created see sections.

2.4 Creation of a function

A function is an application which can be imported inside another application. This can be useful when managing identical sub-parts inside a program or in several programs. A function can also call other functions.

For the creation of a function, an application has to be created and saved as a classical application, without declaring the type of PLC. For the declaration of variables, the field "Parameter" has to be defined, selecting one of the following options:

Internal: the variable is only valid inside the function. It is initialized at every call of a function, except in the case of having the "Global" property selected.

Input: the value given by the calling application is taken at the call of the function. It appears inside the function plot on the Ladder page.

Output: the value given by the calling application is taken at the call of the function. The value is resent to the calling application at the end of the function execution. It appears inside the function plot on the Ladder page.

External: the value given by the calling function is taken at the call of a function. The value is forwarded to the calling application at the end of the function execution. It does not appear inside the function design on the Ladder page.

Note:

- The Timer function can be used if the variable is declared with the Global property. The system is in charge of decreasing it with the correct period, even if the function is not regularly called.
- A function cannot access the system words bits (%S and %SW). They must be accessed as parameters.
- Each time a function is used, it arranges its own memory space. This allows to use the same function many times inside a program and save data to memory at each call, from current cycle to next cycle.

2.4.1 Defining variables

As said before, only array variables need to be defined manually. However, in the case of functions, it is necessary to manually define input and output variables of the function created.

2.4.2 Use of a function

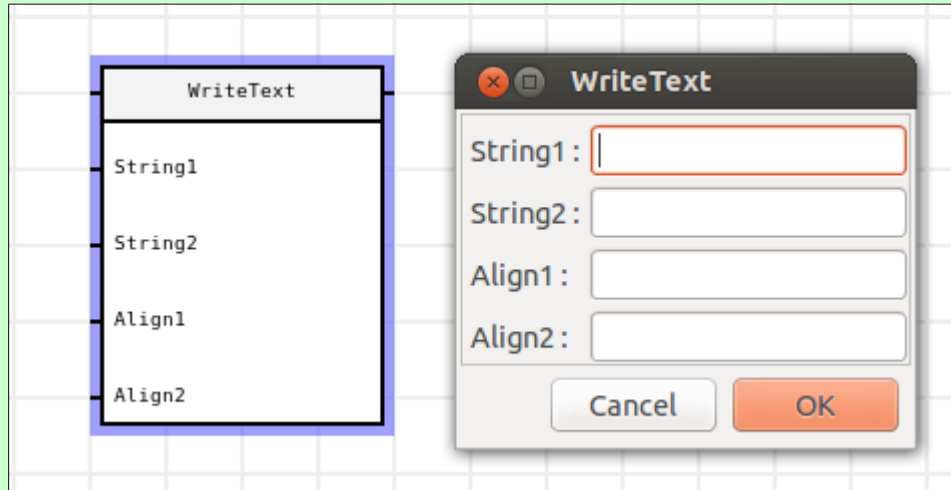
The function must be imported inside the application by launching the "Library/Import function" command. A single import is enough, even if the function is used many times.

2.4.2.1 Ladder

A "Function" object must be inserted into a program page by using Add/Function.

For the binary type data, contacts and coils can be connected to the function's connections. For the digital type data, double click on the function and define the exchanged variables. This last method can be also used for the binary type data.

WriteText function.



8.9.2.2 C code

The function must be called by its name and variables must be introduced as parameters in the same order that they were declared: first the input variables, then output variables and finally external variables. This is similar to the call of a C function, with output and external variables being refreshed, even if they are not in the call of the function.

WriteText function.

```
string1="Good morning";  
string2="Good night";  
align1=1;  
align2=1;  
WriteText (string1, string2, align1, align2);
```

2.5 Using timer

In general, the timer presents 2 different status: inactive timer and active timer:

The timer is inactive when its value is zero or lower.

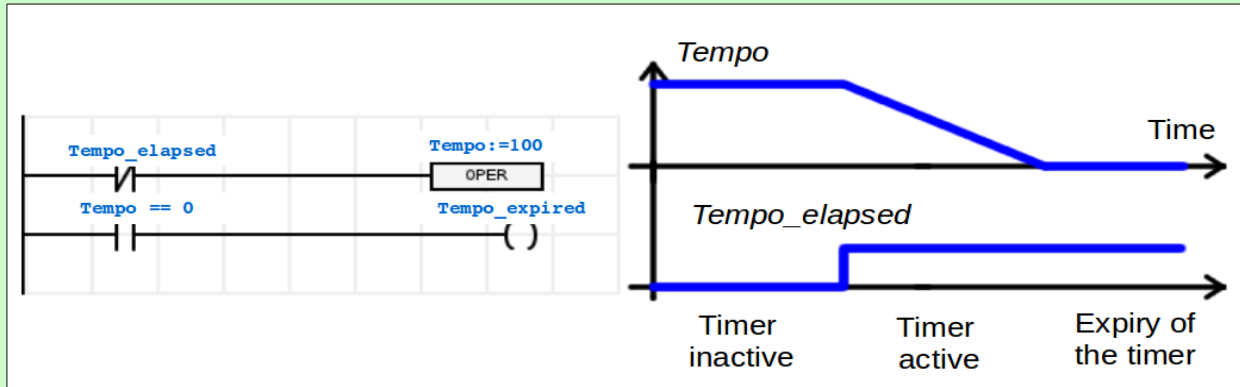
The timer is active while its value is higher than zero and lower than its countdown period value. In this case it decreases with the countdown period indicated in the variable.

Understanding the timer

At the first state there is no time elapsed (the binary variable `Tempo_elapsed` is set to 0), the value of the countdown period (100) is loaded in the `Tempo` variable. The status of the timer is inactive. While `Tempo_elapsed` is 0, the `Tempo` variable remains set to 100, so no countdown is performed.

At the second state, the binary variable `Tempo_elapsed` is set to 1. The value of the countdown period (100) is no longer written inside the `Tempo` variable and the value of `Tempo` decreases. The status of the timer is active.

At the third state, the value of Tempo drops off to 0 after the countdown. The timer has then expired, being inactive again.



Note: It is possible to use a bit to create a timed variable. In this case, the time base is equal to the timer value.

2.5.1 How to use a timer

For the use of the timer, the property "Timer" of a variable must be set. Just define the countdown period (in ms) inside the "Timer" property and this variable will be managed by the system as a time delay.

The time delay duration is equal to the variable value multiplied by the "Timer" property of the same variable.

The use of a timer in ladder or C is similar, as in both cases the property "Timer" of the variable used as timer has to be set.

2.5.2 Examples of the use of timer

To better understand how to use a timer in a program, the following example is being done. Considering again the PLC MicroARM A1, the program to create has to do the following things:

- When pushing the upper button (%I0) a counter is activated. This counter only counts the time when being pushed.
- When releasing the button, the counter resets to zero.
- After 3 seconds with the button being pushed, the backlight of the LCD screen turns on if it was off, or off if it was on.

This program can be created in ladder as it is shown in Figure 17.

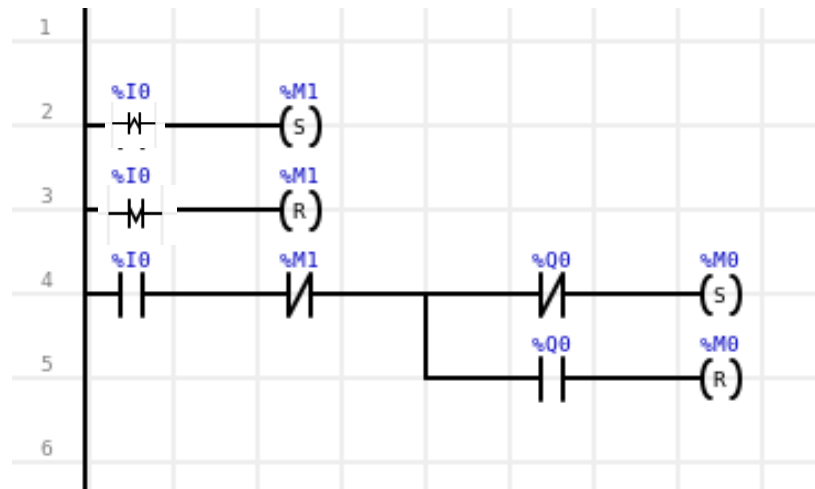


Figure 17. Example with the use of a timer in MicroLADDER.

The counter action has been assigned to the %M1 variable. For establishing the timer, just change the properties of this variable and set the value of 3 seconds. (Be aware that the unit considered in the properties is millisecond, so 3000ms should be written).

2.6 Using a GUI (HMI)

Some PLCs include a graphic screen, whose graphic user interface can be easily designed with the MicroHMI software. This software is complementary to MicroLADDER, and has been conceived for the development of Human-Machine Interface.

Adding a GUI to a MicroLADDER program provides an extra degree of flexibility in the PLC programming with MicroLADDER, as it is possible to easily interact between a ladder or C program and a GUI, by linking variables, functions and/or sequences of code.

2.7 MicroHMI

MicroHMI is the specific software created for the easy design of graphical HMI. This software allows to insert pictures from file, push button, toggle button, text areas and many other options in order to easily design the user interface of a PLC screen.

For further information regarding MicroHMI, please read the MicroHMI user guide.

2.8 How to insert a GUI

As previously introduced, the management of GUIs can be only considered when programming a graphic PLC. When programming this type of PLCs, it is possible to additionally include a GUI into a MicroLADDER application when programming. You can do it by selecting Program>Import HMI from the menu bar, and selecting a .vu file already created with MicroHMI.

2.9 How to configure a GUI

Once a GUI has been imported into a MicroLADDER program, both files, the MicroHMI .vu file and the .lad file can interact between them to relate functions or actions with graphical effects. This interaction can be done through variables defined in MicroLADDER and

assigned in both files.

It is important to know that there is no sign in MicroLADDER that warns whether a program contains a GUI imported or not.

2.9.1 Example

To understand better how to use a GUI in a program, the following example is being done. Considering the PLC MicroARM A2, the program to create has the following configuration: First of all, a simple GUI is being created with MicroHMI software. This GUI will contain the following objects:

- A text field that will show the value of a word %SW5, that means, the word containing the seconds of the current time.

- A dial that will show as well the value of the word %SW5, up to a maximum angle of 180°, with a fill color defined by a new user variable called "DialColor".

- A push button linked to variable %M0. When pushing the button, the variable turns into 1. When releasing the button, the variable turns back to 0.

The MicroLADDER program will be very simple as well, and will just include a single function: if the value of the variable %M0 is 1, then the DialColor turns into red, otherwise, the DialColor is blue.

For the creation of the .vu file with MicroHMI a new project has to be created. Then, the following objects are being added:

A text field. The size of this field can be manually adjusted with the mouse, and the appearance (color, border, etc.) can be defined as the user may want to. For this example, only the property "Text field" will be specified as dynamic with value %SW5 which means that the text will take value of the %SW5 variable.

A push button. For this example, the logo of Sirea has been chosen for the appearance of the push button. For this example, only the property "Main variable" will be determined as %SW5.

A dial. The main variable, it is the variable represented in the dial, will be the %SW5, and the color will be specified by the variable DialColor, so in the property Fill Color, this variable has to be specified as a dynamic with value *DialColor*.

We could additionally change the background color of the screen by double clicking on it and specifying the background color property, setting it, for example, to yellow #FF0.

The appearance of this interface can be similar to the shown in Figure 18.

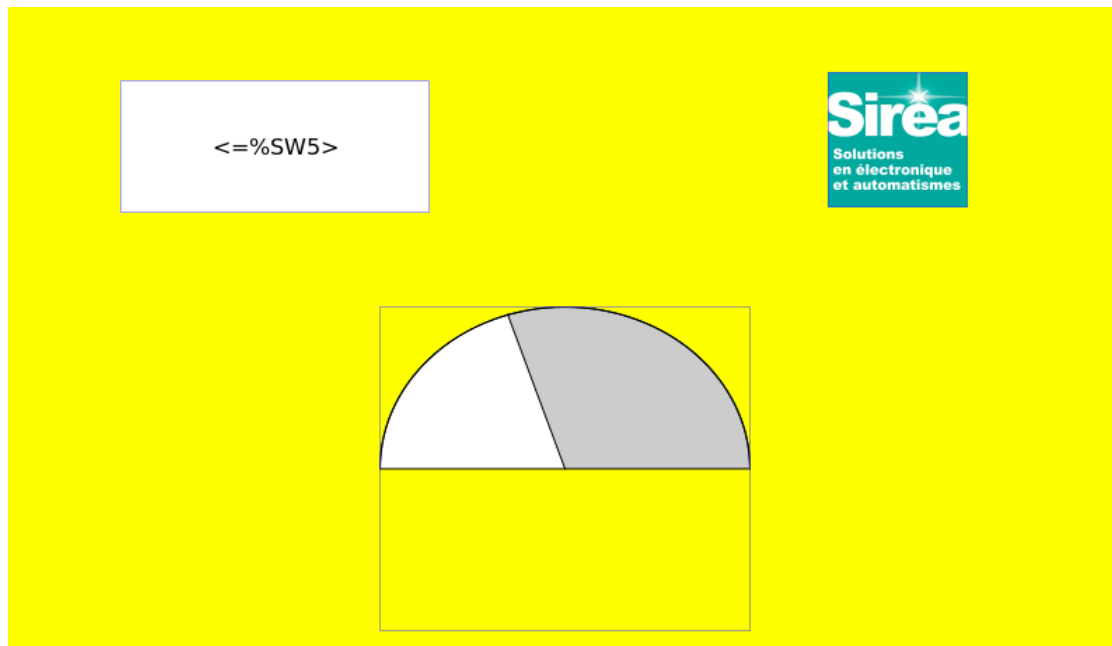


Figure 18. Example of the appearance of the interface created.

The next step is to create the program in MicroLADDER and import the GUI just created. For this purpose, create a new project in MicroLADDER, do Program>Import HMI and select the .vu file just created from the explorer window that pops up.

This program will just include a single C Page, with the following code:

```
if (%M0) DialColor="#F00";  
else DialColor="#00F";
```

With this code the DialColor will turn into red when the value of variable %M0 is 1, and will turn back to blue when its value is 0. And variable %M0 will turn into 1 when pressing the push button on the interface. That means that, when pressing the button, the dial will change into red color.

It is important not to forget to define variables %M0 and DialColor (%MS) in MicroLADDER. As %SW5 is a system word, there is no need to define it.

3 SOFTWARE ENVIRONMENT

3.1 System architecture

3.1.1 Monitor software

The monitor (or monitor software) is the base software layer, which is loaded by Sirea. It allows to load the software application. The user will not need to modify it unless a new software upload may require reloading the monitor software.

If an SD card is available when switching on, FORCE_MON and RUN parameters from file "MAIN.CFG" available on the SD card will determine the launch of the main loop and the application.

Parameters are saved at the same time on the SD card and on the EEPROM or FRAM. It is thus, possible to operate without the SD card. The monitor is loaded by Sirea.

3.1.1.1 LED showing operational stat of the PLC

This LED can be welded on the electronic card or accessible on a connector.

3.1.1.2 Monitor mode

The LED flashes slowly: 500ms ON, 500ms OFF. The PLC is under Modbus control. It can be controlled by MicroCONTROL.

3.1.1.3 Transferring the program from the SD card to the PLC memory

The LED flashes quickly: 100ms ON, 100ms OFF.

The transfer only lasts few seconds. It happens when MicroCONTROL has been transferred. You can transfer it by pushing the button or by going the SD card parameter.

3.1.1.4 STOP mode

The LED flashes slowly and stays mainly OFF: 100ms ON, 900ms OFF.

The PLC is not on the monitor anymore, the main loop begins, the application does not run.

3.1.1.5 RUN mode

The LED flashes slowly and stays mainly ON: 900ms ON, 100ms OFF.

The application runs.

3.1.1.6 Incompatible versions between the monitor and the application

The LED lights static red. This only happens when the application begins, the data structures saved by the monitor is different from the data structures saved by the application. This can create issues with the saved variables.

3.1.2 "MAIN.CFG" file

This file can be found on the SD card. It has the parameter setting. It is read and rewritten while switching on, while launching the application or while inserting the SD card. Therefore, if the file is incomplete, it is fully rewritten. If changes need to be made on this file, it is important to proceed carefully.

Some characteristics from this file can be changed through variables if it's needed (see Table 9).

Table 9. Description of the MAIN.CFG variables.

MAIN.CFG variables	Description
FORCE_MON	It forces to stay on the monitor and not to force the application if its value is 1
LOAD	It indicates the need of loading the program into the memory if its value is 1. It is then reset to 0 at the end of the loading process
FIRST_RUN	It indicates that variables must be initialized if its value is 1. It is

	not necessary to set the variable to 1 when loading, it is automatically done by the system. This characteristic can be controlled through the variable %S2.
RUN	It allows to jump from the application in STOP mode to the application in RUN mode, when its value is 1.
RESET_SW	It indicates that a cycle time violation (watchdog) has taken place thus blocking the PLC on the main loop, when its value is 1. This characteristic can be controlled through the variable %SW21.
RESET_CFG	It allows to reinitialize variables of the "MAIN.CFG" file if its value is 1.
LOAD_IO_CFG	It allows to reload the "var.csv" file and reinitialize the log of curves, alarms and events. This characteristic can be controlled through the variable %S19.
MODE_DST	It allows to memorize the management of the summer time. This characteristic can be controlled through the variable %SW11.
TIME_OFF	It allows to fix the difference with the UTC time in minute. It is useful for automatic time-setting. It is the variable %SW12.
BOOT_VER	It indicates the version of the boot installed on the PLC. It is the variable %SW23.
W_SSID	It allows to set the SSID (access point name) for the Wi-Fi. The standard allows the use of 32 characters maximum.
W_STYPE	It allows to set the type of encryption for the Wi-Fi. The possible values are "" (empty character string that uses automatic detection), OPEN (no encryption), WEP, WPA, WPAES, WPA2AES, WPA2TKIP, WPA2
W_SKEY	It allows to set the security key for the Wi-Fi

Example of Main.CFG file with default value if they don't exist. It shows that if there is no Main.CFG file on the SD card, the PLC will launch the main loop of the application present in memory. If there is no application, the system will be blocked.

```
FORCE_MON=0
LOAD=0
RUN=0
```

3.1.3 Loadmain file

When this file is present on the SD card, the application is reloaded and the file is deleted. This method is equivalent to setting the LOAD variable to 1, but it avoids changing the other variables in the "MAIN" file. CFG". The content of the file is not important, only its presence counts. So it is possible to create a file called "loadmain" on the SD card. The PLC detects this file and loads the file main.hex in its memory. The quickest method is to let MicroLADDER save the compiled files directly on the SD card, and then to put the SD card in the PLC.

3.1.4 Loading an application

Sirea, a company specialized in the field of industrial automation and electrical energy.

Having an SD card is not compulsory. If you have one it must be formatted into FAT32.

3.1.4.1 MicroCONTROL

MicroCONTROL software enables to load ".hex" files, that means, already compiled files. The transfer can be done using a serial port or the Ethernet port. It is thus possible to load it remotely from the application's code but not from the HMI code.

The file transfer takes place first to the SD card, if there is a SD card available. After the transfer process the file will be loaded into the PLC memory. The application will be then started by MicroCONTROL.

In the case of loading for the first time an application into the PLC, it is necessary to establish FORCE_MON=1 inside the MAIN.CFG file of the SD card. On the contrary, the PLC will try to launch the main loop of the application, being it in-existent.

3.1.4.2 SD Card

If the PLC includes a SD card you can just copy the file compiled by MicroLADDER ("main.hex") to the SD card and the file "MAIN.CFG" created that contains the following information:

FORCE_MON=0 LOAD=1 RUN=1

The SD card must be then inserted in the PLC. The operating status LED indicates the program transfer from the SD card to the PLC memory by means of a fast blinking. Once the program is transferred, the operating status LED indicates the execution of the application with a blinking, mainly with the status LED light on.

A second option is to use the programming button to load programs. There are specific cases where the loading of a program must be executed manually by pressing buttons provided in some PLCs:

- If the user saves previously the file "MAIN.HEX" on the SD card, but not the file "MAIN.CFG"
- If the user wants to reload an old .hex file (not a .hex file just generated).

In these cases, after inserting the SD card the reset button must be pressed, and before the end of the first second, the programming button must then be pressed. The operating status LED indicates the program transfer from the SD card to the PLC memory by means of a fast blinking. When loading is done, it indicates that that the PLC is on the main loop by blinking slowly, mainly with the status LED light off.

For starting the application, the loading button must be then pressed for 3 seconds. The operating status LED indicates the starting of the application with a blinking, mainly with the status LED light on. The method using the SD card allows as well to copy the management files of the HMI. It is possible to add the file "Loadmain " without changing the MAIN. CFG ".

The fastest way to load a program is to save the compiled file directly on the SD card when MicroLADDER tells where to save at the end of the compilation. Thus, MicroLADDER will automatically edit "MAIN.CFG" and "loadmain" files.

3.2 Type of data

It is important to note that MicroLADDER manages those PLC variables able to be used in ladder objects, that means, those accessible to real time monitoring through the software. However, it is sometimes different from variables that the user can declare inside C pages.

3.2.1 DIGITAL inputs

Value is 0 or 1. The number of DIGITAL inputs (binary inputs) depends on the PLC.

Prefix for DIGITAL inputs: %I (I: Input bool) Example: %I0.

These data are visible in standard Modbus by shifting, example %I0 \Rightarrow %M20000.

3.2.2 ANALOG inputs

Value range depends on the PLC used and on the input assignment. The number of ANALOG inputs also depends on the PLC.

These data are visible in standard Modbus by shifting. Example: %IW100.

Some PLCs allow the configuration of the analog inputs through the use of jumpers. The property "configuration" of analog variables must be then adjusted with MicroLADDER to meet the proper value range and calibration needs.

Table 10. Options available for the property "configuration" in analog inputs.

Property "Configuration"	Type of analog input
0	Default configuration. It corresponds to lower value for property configuration available in the PLC.
1	0-20mA. Value range*: 0-20000 points
2	0-10V. Value range*: 0-10000 points
3	PT100. Temperature in decimal of degree Celsius.

* In some specific cases this value range may vary.

3.2.3 DIGITAL outputs

Value is 0 or 1. The number of DIGITAL outputs depends on the PLC.

These data are visible in standard Modbus by shifting. Example %Q0 (Q: Output bool)..

3.2.4 ANALOG outputs

Value range depends on the PLC used. The number of ANALOG outputs also depends on the PLC.

These data are visible in standard Modbus by shifting. Example: %QW (QW: Output words), %QW100.

3.2.5 PWM outputs

Frequency is defined in Hz with %SW26. This value must be between 0 and 65535Hz (it has to be noted that, technically, electronics does not allow to follow frequencies above 10000Hz).

Cycle time is defined with %QW by PWM output. This value must be between 0 and 1000. For instance, to have a time-slot of 20% at 1 and 80% at 0, the %QW variable must be set to 200.

To deal with a binary output, %SW26 must be set to 0, and %QW must be set to 0 or 1000 for setting the BIN output to 0 or 1.

The default value of %SW26 and variables %QW is 0, what corresponds to an output of 0V. When the value of %SW26 is 0, the value of the frequency is equal to the maximum frequency value: 100000Hz. This allows to answer faster to changes in status in binary mode, as the current period necessarily ends before applying the new cycle time.

Note:

- It is possible to set a frequency up to 65535Hz. However, when the frequency reaches too high values the signal gets worse (change in transistor status), even becoming unusable. The maximum limit is approximately 10kHz depending on the load.
- To visualize a signal in the oscilloscope, a resistance of approximately 1k Ω must be connected to the output terminals.
- The output PWM signal varies between 0V and Vcc (supply voltage).

3.2.6 Boolean

Value is 0 or 1. Prefix for boolean: %M (M: Main bool). Example : %M0.

Note: Boolean are character bits. Therefore, 8 bits use 1byte in memory. Accessing a bit takes longer than accessing a byte or a word.

3.2.7 Integer

Value range from 0 to 65535. Prefix for integer: %MW (MW: Main Word). Example : %MW0.

3.2.8 Long

Value range from - 2 147 483 648 to 2 147 483 647. Example: %MD0. These data are visible in standard Modbus by shifting.

Example (%MD10: w0 \Rightarrow %MW20020 et %MD10: w1 \Rightarrow %MW20021).

3.2.9 Float

Value range from $3,4 \times 10^{-38}$ to $3,4 \times 10^{38}$, both positive and negative values. Example: %MF0. These data are visible in standard Modbus by shifting.

Example (%MF10:w0 \Rightarrow %MW30020 et %MF:w1 \Rightarrow %MD30021).

3.2.10 String

The length of the string must be indicated in the field "string size". Prefix for string: %MS (MS: Main String). Example: %MS0.

%MS20[5] represents the fifth element of a table of strings. It is equivalent to %MS25.

(%MS20)[5] represents the fifth character of the string %MS20.

It is preferable to use assignment instead of the C classic instruction "strcpy" when copying a string, since the assignment calls a specific function that controls the size of the string and avoids overflows.

These data are available in standard Modbus by adding a shift of 50000 on the %MW. Two characters are placed on a same word. The correspondence of the addresses is complex because it depends on the string's size, which is defined at the below addresses.

3.2.11 System bits

These data are available in standard Modbus by shifting (%S0 ⇒ %M22000).

Bit	Mnemonic	Description
%S0	MST	Bit set to 1 at the first cycle of the system program, followed by a switching on or a reloading of the program. Bit re-established to 0 automatically after executing the first cycle. This bit is slightly different from %S16 (DEM_C).
%S1	RUN	Bit set to 1; set to status 0 by the user program or through data writing mode via the serial link (RUN/STOP command). Bit to 1 : program run. Bit to 0 : program stop.
%S2	INIT	Bit set to 0, set to status 1 by the system at the starting of the program or in data writing mode via the serial link (INIT command). Bit to 1: loading of the initial values in the data values. No modifications on saved variables. Re-established to 0 automatically at the end of the initialization.
%S3		Spare
%S4		Spare
%S5	MSEC_10	Bits where the change in status is synchronized with the internal clock. They are asynchronous with respect to the cycle of the program. Example S5 : State 1: 5ms – State 0: 5ms.
%S6	MSEC_100	
%S7	SEC_1	
%S8	MIN_1	
%S9		Spare
%S10		Spare
%S11	INIT_Q	Bit set to 1; set to status 0 by the user program or via the serial link. Bit to 1: causes the setting of the outputs to 0 Bit to 0: outputs are updated by the user program
%S12	GEL_Q	Bit set to 1; set to status 0 by the user program or via the serial link. Bit to 1: causes the outputs to be frozen at their status Bit to 0: outputs are updated by the user program

%S13	MAJ_H	Bit to 0; set to status 1 by the user program or via the serial link. Bit to 1: Used to set time and date : copy the values from %SW5 to corresponding %SW10 in the system clock. Bit to 0: words from %SW5 to %SW10 are updated by the system to give current time and date. Since 29/06/12, this bit is useless. It is possible to change %SW5 to %SW10 at any time. The change is saved at the end of the PLC loop cycle.
%S14	RAZDEF	Spare
%S15	CDG	Bit set to 0; set to status 1 by the system when the program cycle time goes beyond the maximum cycle time defined by the system word SW1. It is automatically re-established to 0 through an INIT command, either at switching on (mode MST) or by the user program.
%S16	DEM_C	Bit to 1 at the first cycle of the user program, no matter the cause of the restart (switching on, reloading, starting through the console) Bit automatically set to 0 at the end of the 1st cycle. This bit is slightly different from %S0 (MST)
%S17		Spare
%S18	SAV_VARS	Bit set to 0; set to status 1 by the user program or via serial link. Variables declared as "Saved" are saved in the EEPROM or in the FRAM. Bit re-established to 0 after saving process. See section on Saved variables for details. Note: backup should not be performed permanently. The number of writings of the EEPROM is limited.
%S19	LOAD_IO_CFG	Bit normally set to 0, set to status 1 by the user program or via serial link. It causes the re-reading of "var.csv" file and the reset of the alarms and events history report.
%S20	RESET_HARD	Spare
%S21	RESET_SOFT	Bit normally set to 0, set to 1 by the system after a reset soft of the watchdog. This bit is re-established to 1 by the system when the program restarts. See Mx_CYC: %SW2 and section 3.6 on Watchdog for further information.
%S22	RESET_ERROR	Spare
%S23		Spare
%S24	ALM_BAT_RTC	Set normally set to 0, set to 1 by the system when the backup battery level is low.
%S25	DHCP	Use DHCP to set Ethernet port. Socket 3 is used for the initial request and for periodical requests. It means that DHCP temporarily takes over the socket at the expense of the use planned by the application.

%S26	W_DHCP	Use DHCP to set Wi-Fi port. On the opposite of the Ethernet, the Wi-Fi module uses an internal socket that doesn't interfere with the application.
%S27 to %S49		Spare

3.2.12 System words

These data are available on standard Modbus by shifting (%SW0 ⇒ %MW42000).

%SW0	T_CYC	PLC current cycle runtime in ms. Only read access.
%SW1	S_CYC	PLC cycle time surveillance Reference value allowing the system to control cycle time overrun (SW0>SW1) and positioning the %S15 bit (CDG). Application is not stopped. Read and write access.
%SW2	Mx_CYC	PLC cycle time-out value. Reference value allowing the system to control a maximum cycle time overrun. In case of overrun (SW0>SW2), Saved variables are saved by the system, BIN and ANA outputs are forced to 0 and the PLC is switched to STOP mode. The %S21 bit (RESET_SOFT) is set to 1. Read and write access.
%SW3		Spare
%SW4	WD_HAR D	Spare
%SW5 %SW6 %SW7 %SW8 %SW9 %SW10	SECONDE MINUTE HEURE JOUR MOIS ANNEE	Date and time function. Words containing current values for date and time. These words are accessible for both permanent reading and writing regardless of the situation of %S13 (MAJ_H).
%SW11	MODE_DS T	Automatic management of summer time. Value 0: change to summer time is automatically managed by the system according to French conventions. Value different to 0: automatic change to summer time not managed. The material component (RTC) always keeps the winter time. Change to summer time considered by system words %SW5 to %SW10 usable value.
%SW12	TIME_OFF	Lag with UTC time (in minutes). For France, put 60.
%SW13	VERSION_ SYS	System code version. Only read access.

%SW14	VERSION_APP	Application version. This word is available to the user for the record of the application version number.
%SW15	SAV1	These 8 words are available to the user for word backup. See section on Saved variables for further details.
%SW16	SAV2	
%SW17	SAV3	
%SW18	SAV4	
%SW19	SAV5	
%SW20	SAV6	
%SW21	SAV7	
%SW22	SAV8	
%SW23	VERSION_BOOT	Boot version.
%SW24	VERSION_MLADDER	Version of MicroLADDER that was used for compilation.
%SW25	FREQ_TIMER	Calling period of interrupting functions in μ s. Functions to be called need to have the "Call on interrupt" property confirmed. Value 0: no calling Value different to 0: calling period in μ s. This value is not saved. It must be initialized by the program. During the execution of the program, the value can be modified if needed. Too weak values could block the PLC.
%SW26	FREQ_PWM	PWM output frequency.
%SW27	FET1	Reading mask. Allows to unabled the inputs refreshing. Each input is linked to a bit. HMI inputs are not affected. 1st and 2nd input bytes 3rd and 4th input bytes 5th and 6th input bytes 7th and 8th input bytes 9th and 10th input bytes 11th and 12th input bytes Note : a 16 bits input card straddles 2 system words.
%SW28	FET2	
%SW29	FET3	
%SW30	FET4	
%SW31	FET5	
%SW32	FET6	
%SW33		Spare
%SW34	SER0	Port configuration (see Modbus details below)
%SW35	NESC_SER0	Slave number (variable saved on the SD card) result of the exchange (see Modbus details below)
%SW36	RES_SER0	Speed code (see Modbus details below)
%SW37	SPD_SER0	Communication format (see Modbus details below)
%SW38	FOR_SER0	For the MASTER Modbus, maximum time value in ms between the sending of a frame and the reception of the answer. Default value 3000ms.
%SW39	TIMEOUT_SER0	

%SW40	SER1	Port configuration (see Modbus details below)
%SW41	NESC_SER1	Slave number (variable saved on the SD card) result of the exchange (see Modbus details below)
%SW42	RES_SER1	Speed code (see Modbus details below)
%SW43	SPD_SER1	Communication format (see Modbus details below)
%SW44	FOR_SER1	For the MASTER Modbus, maximum time value in ms between the sending of a frame and the reception of the answer. Default value 3000ms.
%SW45	TIMEOUT_SER1	
%SW46	SER2	Port configuration (see Modbus details below)
%SW47	NESC_SER2	Slave number (variable saved on the SD card) result of the exchange (see Modbus details below)
%SW48	RES_SER2	Speed code (see Modbus details below)
%SW49	SPD_SER2	Communication format (see Modbus details below)
%SW50	FOR_SER2	For the MASTER Modbus, maximum time value in ms between the sending of a frame and the reception of the answer. Default value 3000ms.
%SW51	TIMEOUT_SER2	
%SW52	SER3	Port configuration (see Modbus details below)
%SW53	NESC_SER3	Slave number (variable saved on the SD card) result of the exchange (see Modbus details below)
%SW54	RES_SER3	Speed code (see Modbus details below)
%SW55	SPD_SER3	Communication format (see Modbus details below)
%SW56	FOR_SER3	For the MASTER Modbus, maximum time value in ms between the sending of a frame and the reception of the answer. Default value 3000ms.
%SW57	TIMEOUT_SER3	
%SW58	SER4	Port configuration (see Modbus details below)
%SW59	NESC_SER4	Slave number (variable saved on the SD card) result of the exchange (see Modbus details below)
%SW60	RES_SER4	Speed code (see Modbus details below)
%SW61	SPD_SER4	Communication format (see Modbus details below)
%SW62	FOR_SER4	For the MASTER Modbus, maximum time value in ms between the sending of a frame and the reception of the answer. Default value 3000ms.
%SW63	TIMEOUT_SER4	
%SW64	SER5	Port configuration (see Modbus details below)
%SW65	NESC_SER5	Slave number (variable saved on the SD card) result of the exchange (see Modbus details below)
%SW66	RES_SER5	Speed code (see Modbus details below)
%SW67	SPD_SER5	Communication format (see Modbus details below)
%SW68	FOR_SER5	For the MASTER Modbus, maximum time value in ms between the sending of a frame and the reception of the answer. Default value 3000ms.
%SW69	TIMEOUT_SER5	

%SW70	SER6	Port configuration (see Modbus details below)
%SW71	NESC_SER6	Slave number (variable saved on the SD card) result of the exchange (see Modbus details below)
%SW72	RES_SER6	Speed code (see Modbus details below)
%SW73	SPD_SER6	Communication format (see Modbus details below)
%SW74	FOR_SER6	For the MASTER Modbus, maximum time value in ms between the sending of a frame and the reception of the answer. Default value 3000ms.
%SW75	TIMEOUT_SER6	
%SW76	SER7	Port configuration (see Modbus details below)
%SW77	NESC_SER7	Slave number (variable saved on the SD card) result of the exchange (see Modbus details below)
%SW78	RES_SER7	Speed code (see Modbus details below)
%SW79	SPD_SER7	Communication format (see Modbus details below)
%SW80	FOR_SER7	For the MASTER Modbus, maximum time value in ms between the sending of a frame and the reception of the answer. Default value 3000ms.
%SW81	TIMEOUT_SER7	
%SW91	W_DNS_A	Address of the DNS server for Wi-Fi port.
%SW92	DDR1	
%SW93	W_DNS_A	
%SW94	DDR2	
	W_DNS_A	
	DDR3	
	W_DNS_A	
	DDR4	
%SW95	W_MODE	Operating mode of the Wi-Fi port.
%SW96	DNS_ADR	Address of the DNS server for the Ethernet port and Wi-Fi port.
%SW97	1	From MicroLADDER 10.0 and system code 13.0 these words are only available for the Ethernet as there are now words for the Wi-Fi.
%SW98	DNS_ADR	
%SW99	2	
	DNS_ADR	
	3	
	DNS_ADR	
	4	
%SW100	MAC_AD	Ethernet connector MAC address.
%SW101	DR1	
%SW102	MAC_AD	
%SW103	DR2	
%SW104	MAC_AD	
%SW105	DR3	
	MAC_AD	
	DR4	
	MAC_AD	
	DR5	
	MAC_AD	
	DR6	

%SW106 %SW107 %SW108 %SW109	IP_ADDR1 IP_ADDR2 IP_ADDR3 IP_ADDR4	Ethernet connector IP address.
%SW110 %SW111 %SW112 %SW113	SUBNET_MASK1 SUBNET_MASK2 SUBNET_MASK3 SUBNET_MASK4	Ethernet connector subnet mask.
%SW114 %SW115 %SW116 %SW117	GATEWAY1 GATEWAY2 GATEWAY3 GATEWAY4	Ethernet connector gateway.
%SW118 %SW119 %SW120 %SW121 %SW122	SOCK0 PORT_SOCK0 NESC_SOCK0 RES_SOCK0 TIMEOUT_SOCK0	Socket communication protocol (see Modbus details below) Socket port number in slave mode (502 in Modbus, 10000 in VNC, 80 in HTTP, 161 in SNMP) Socket slave number Socket result of the exchange Socket time out
%SW123 %SW124 %SW125 %SW126 %SW127	SOCK1 PORT_SOCK1 NESC_SOCK1 RES_SOCK1 TIMEOUT_SOCK1	Socket communication protocol (see Modbus details below) Socket port number in slave mode (502 in Modbus, 10000 in VNC, 80 in HTTP, 161 in SNMP) Socket slave number Socket result of the exchange Socket time out

%SW128	SOCK2	Socket communication protocol (see Modbus details below)
%SW129	PORT_SO	Socket port number in slave mode (502 in Modbus, 10000 in VNC,
%SW130	CK2	80 in HTTP, 161 in SNMP)
%SW131	NESC_SO	Socket slave number
%SW132	CK2	Socket result of the exchange
	RES_SOC	Socket time out
	K2	
	TIMEOUT	
	_SOCK2	
%SW133	SOCK3	Socket communication protocol (see Modbus details below)
%SW134	PORT_SO	Socket port number in slave mode (502 in Modbus, 10000 in VNC,
%SW135	CK3	80 in HTTP, 161 in SNMP)
%SW136	NESC_SO	Socket slave number
%SW137	CK3	Socket result of the exchange
	RES_SOC	Socket time out
	K3	
	TIMEOUT	
	_SOCK3	
%SW138	W_MAC_	Wi-Fi connector MAC address.
%SW139	ADDR1	
%SW140	W_MAC_	
%SW141	ADDR2	
%SW142	W_MAC_	
%SW143	ADDR3	
	W_MAC_	
	ADDR4	
	W_MAC_	
	ADDR5	
	W_MAC_	
	ADDR6	
%SW144	W_IP_AD	Wi-Fi connector IP address.
%SW145	DR1	
%SW146	W_IP_AD	
%SW147	DR2	
	W_IP_AD	
	DR3	
	W_IP_AD	
	DR4	

%SW148 %SW149 %SW150 %SW151	W_SUBNET_MASK1 W_SUBNET_MASK2 W_SUBNET_MASK3 W_SUBNET_MASK4	Wi-Fi connector subnet mask.
%SW152 %SW153 %SW154 %SW155	W_GATEWAY1 W_GATEWAY2 W_GATEWAY3 W_GATEWAY4	Wi-Fi connector gateway.
%SW156 %SW157 %SW158 %SW159 %SW160	WSOCKET0 PORT_WSOCKET0 NESC_WSOCKET0 RES_WSOCKET0 TIMEOUT_WSOCKET0	Socket communication protocol (see Modbus details below) Socket port number in slave mode (502 in Modbus, 10000 in VNC, 80 in HTTP, 161 in SNMP) Socket slave number Socket result of the exchange Socket time out
%SW161 %SW162 %SW163 %SW164 %SW165	WSOCKET1 PORT_WSOCKET1 NESC_WSOCKET1 RES_WSOCKET1 TIMEOUT_WSOCKET1	Socket communication protocol (see Modbus details below) Socket port number in slave mode (502 in Modbus, 10000 in VNC, 80 in HTTP, 161 in SNMP) Socket slave number Socket result of the exchange Socket time out

%SW166	WSOCK2	Socket communication protocol (see Modbus details below)
%SW167	PORT_WS OCK2	Socket port number in slave mode (502 in Modbus, 10000 in VNC, 80 in HTTP, 161 in SNMP)
%SW168		Socket slave number
%SW169	NESC_WS	Socket result of the exchange
%SW170	OCK2 RES_WSO CK2 TIMEOUT _WSOCK2	Socket time out
%SW171	WSOCK3	Socket communication protocol (see Modbus details below)
%SW172	PORT_WS OCK3	Socket port number in slave mode (502 in Modbus, 10000 in VNC, 80 in HTTP, 161 in SNMP)
%SW173		Socket slave number
%SW174	NESC_WS	Socket result of the exchange
%SW175	OCK3 RES_WSO CK3 TIMEOUT _WSOCK3	Socket time out
%SW200	RF0	LoRa connector configuration (see Modbus details below).
%SW201	NESC_RF0	Slave number (this variable is saved in the SD card)
%SW202	RES_RF0	Exchange results (see Modbus details below)
%SW203	FREQ_RX_ RF0	Channel of reception frequency (see radio-frequency details below)
%SW204		Channel of sending frequency (see radio-frequency details below)
%SW205	FREQ_TX_ RF0 TIMEOUT _RF0	For the MASTER Modbus, maximum time value in ms between the sending of a frame and the reception of the answer. Default value 3000ms.
%SW206	RF1	LoRa connector configuration (see Modbus details below).
%SW207	NESC_RF1	Slave number (this variable is saved in the SD card)
%SW208	RES_RF1	Exchange results (see Modbus details below)
%SW209	FREQ_RX_ RF1	Channel of reception frequency (see radio-frequency details below)
%SW210		Channel of sending frequency (see radio-frequency details below)
%SW211	FREQ_TX_ RF1 TIMEOUT _RF1	For the MASTER Modbus, maximum time value in ms between the sending of a frame and the reception of the answer. Default value 3000ms.

3.2.13 Modbus Detail

SER0 to SER7 SOCK0 to SOCK3 WSOCK0 to WSOCK3 RF0 to RF1	Port configuration. These variables are not saved.	
Value name	Variable value	Description
COM_PROTOCOL_NONE COM_PROTOCOL_TCP COM_PROTOCOL_TCP_CLIENT	0	No protocol or Master Modbus
COM_PROTOCOL_IOPUS	1	IOBus protocol
COM_PROTOCOL_MODBUS	2	Slave Modbus
COM_PROTOCOL_MODBUS_TCP	3	Slave TCP Modbus
COM_PROTOCOL_GFX	4	Gfx protocol (remote screen control)
COM_PROTOCOL_HTTP	5	HTTP protocol
COM_PROTOCOL_UDP COM_PROTOCOL_UDP_CLIENT	6	UDP master protocol UDP client protocol
COM_PROTOCOL_DATA	7	TBC
COM_PROTOCOL_TCP_SERVER	8	TBC
COM_PROTOCOL_UDP_SERVER	9	TBC
COM_PROTOCOL_HTTP_QUERY	10	HTTP protocol for the queries (only since 23/01/17)

Note:

- Default value for serial ports is Modbus. MODBUS_TCP for socket 0 and HTTP for sockets 1 to 3, except for PLCs including graphic screen where socket 1 is GFX.
- The use of the variable's name instead of its value for initializing the system word is strongly recommended.

NESC_SER0 à NESC_SER7 NESC SOCK0 à NESC SOCK3 NESC_WSOCK0 à NESC_WSOCK3	Slave number for the slave Modbus mode. These variables do not necessarily need to be set to 0 for performing master Modbus or communication without protocol. These variables are saved on the SD card.	
Value name	Variable value	Description
	0	Only master Modbus
	1 à 255	Modbus slave number

Note : Default value is 1.

RES_SER0 to RES_SER7 RES SOCK0 to RES SOCK3 RES_WSOCK0 to RES_WSOCK3 RES_RF0 to RES_RF1	Result of the exchange. It runs either in transfer or reception.	
Value name	Variable value	Description

COM_STATE_READY	0	End of communication
COM_STATE_WAIT	1	Time delay before sending the frame
COM_STATE_SEND	2	Communication under way (transfer or reception)
COM_STATE_ERROR	240	Limit from which the default zone is entered
COM_STATE_ERROR + MODBUS_ERROR_BAD_RESPONSE	253	Bad answer
COM_STATE_ERROR + MODBUS_ERROR_INCOMPLETE_RESPONSE	254	Incomplete answer
COM_STATE_ERROR + MODBUS_ERROR_NO_RESPONSE	255	Time out : no answer

SPD_SER0 à SPD_SER7		Exchange speed. These variables are saved on the SD card.
Value name	Variable value	Description
COM_SPEED_NONE	0	
COM_SPEED_300	1	300 bauds
COM_SPEED_1200	2	1200 bauds
COM_SPEED_2400	3	2400 bauds
COM_SPEED_4800	4	4800 bauds
COM_SPEED_9600	5	9600 bauds
COM_SPEED_19200	6	19200 bauds
COM_SPEED_38400	7	38400 bauds
COM_SPEED_57600	8	57600 bauds
COM_SPEED_76800	9	76800 bauds
COM_SPEED_115200	10	115200 bauds

Note :

- Default value is 7.
- Speeds 76800 and 115200 do not work for all ports.

FOR_SER0 à FOR_SER7		Exchange format parameter. These variables are saved on the SD card.
Value Name	Variable value	Description
COM_FORMAT_NONE	0	
COM_FORMAT_8N2	1	8 bits, no parity, 2 stop bits
COM_FORMAT_8N1	2	8 bits, no parity, 1 stop bit
COM_FORMAT_8E1	3	8 bits, even parity, 1 stop bit
COM_FORMAT_8O1	4	8 bits, odd parity, 1 stop bit
COM_FORMAT_7E2	5	7 bits, even parity, 2 stop bits

COM_FORMAT_7O2	6	7 bits, odd parity, 2 stop bits
COM_FORMAT_7E1	7	7 bits, even parity, 1 stop bit
COM_FORMAT_7O1	8	7 bits, odd parity, 1 stop bit
COM_FORMAT_7R2	9	7 bits, parity forced to 0, 2 stop bits
COM_FORMAT_7R1	10	7 bits, parity forced to 0, 1 stop bit
COM_FORMAT_7S2	11	7 bits, parity forced to 1, 2 stop bits
COM_FORMAT_7S1	12	7 bits, parity forced to 1, 1 stop bit

Note : Default value is 2.

3.2.14 Radio Frequency details

FREQ_RX_RF0 to FREQ_RX_RF1 FREQ_TX_RF0 to FREQ_TX_RF1		Number of the frequency channel of the radio connector. These variables are saved on the SD card.
Value name	Variable value	Description
RF_FREQ_NONE RF_FREQ_0	0	868,125 MHz
RF_FREQ_1	1	868,475 MHz
RF_FREQ_2	2	868,950 MHz
RF_FREQ_3	3	869,525 MHz

3.2.15 Edge management

Every « edge » object is managed by an internal variable independent from the variable used inside:

- The edge is valid for a full cycle: if the variable moves after the edge, the edge is seen at the following tour.
- Possible bugs with indexed bits if the index varies from a cycle to another.
- Uses 1 byte per "edge" object.
- Can manage complex expressions and words bits.

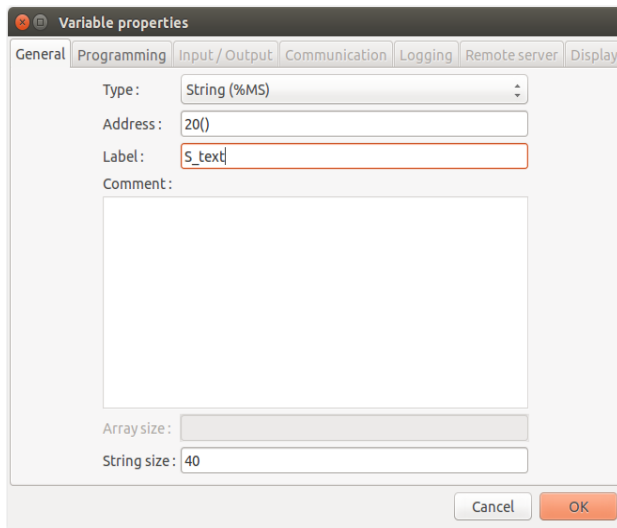
3.3 Importing the variables by overwriting the present variables.

3.3.1 General

Address: for a simple variable it is a digital value. For a table it is the address of the first element followed by "[]".

Number of elements: indicates the number of element in a table scenario.

Number of characters: indicates the number of characters in a character string scenario (%MS).



3.3.2 Programming

Init value: value charged in the variable when charging the application or when asking initialization. If the variable is saved, this value is only used at first start.

Saved: If the PLC has a saved memory, the value will be kept when the power is interrupted. For backups on EEPROM or FRAM, the instant of the backup is at the user initiative.

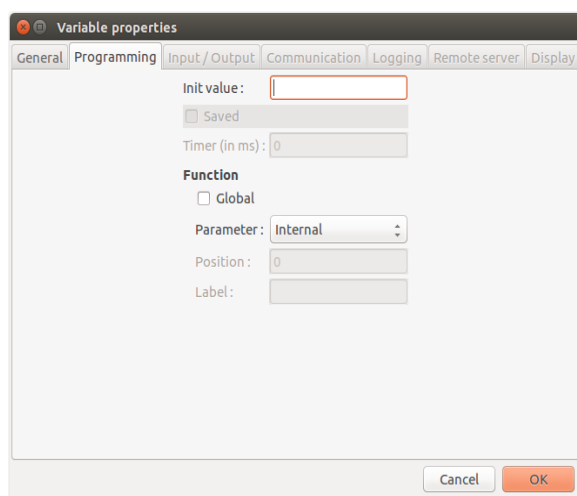
Timer (in ms): the period at which the variable will be decremented from 1 to the value 0.

Global: this property is to be used with functions. It allows the variable to keep its value between two calls.

Parameter: This is used for the variables of a function. See below the chapter dealing with the realization of functions.

Position: This is used for the variables of a function. This is the number of the variable for the graphical representation or the C-language call.

Label: This is used for the variables of a function. This is the text that appears on the graphic representation.

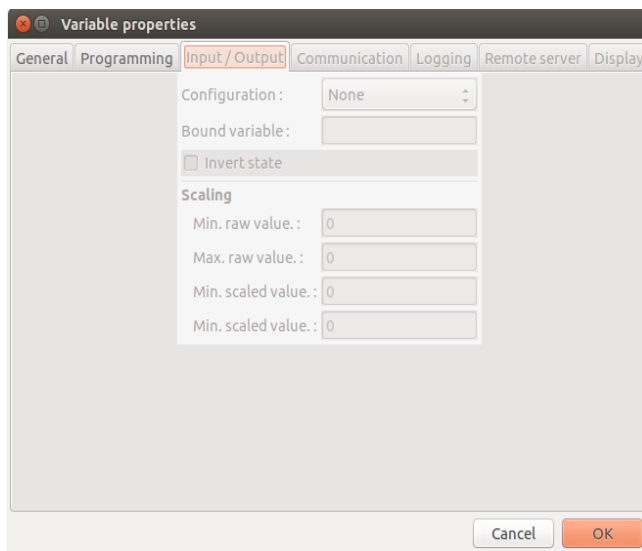


3.3.3 Input / Output

Configuration: setting for analog inputs (see chapter on analog inputs).

Bound variable: allows to link to another application variable to possibly scale.

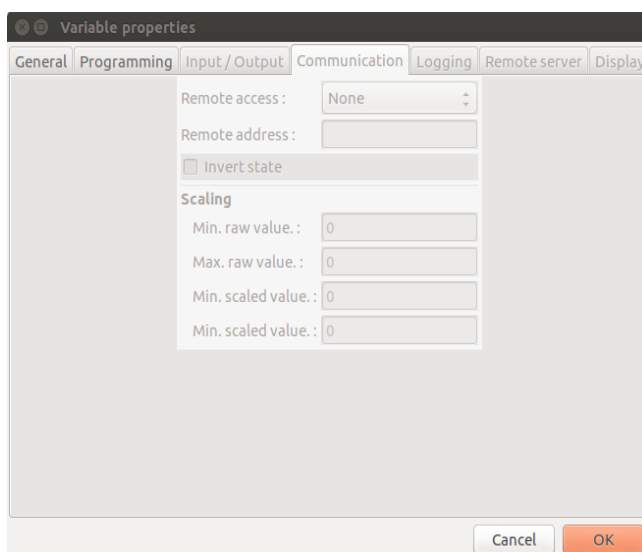
Invert state: only used for binary variables. Allows to reverse the state between the input/output variable and the associated variable.



3.3.4 Communication

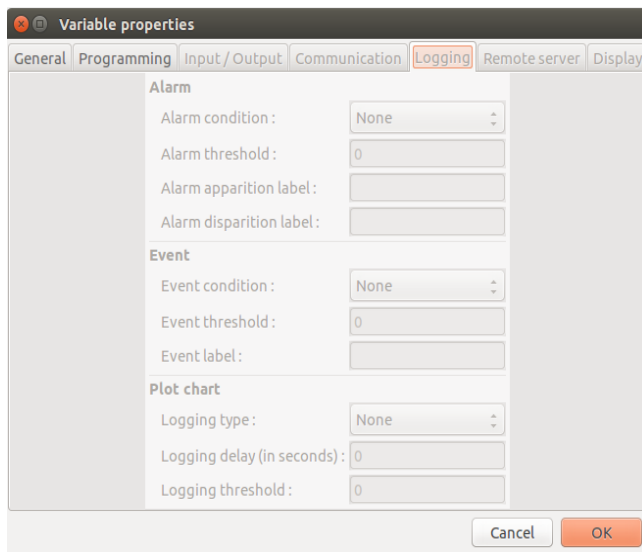
Remote address: This is the name of the variable of a distant device followed by a point and the number of the equipment. See the chapter IO Bus for more details.

Invert state: Used only for binary variables. Allows to reverse the state between the local variable and the remote variable.



3.3.5 Logging

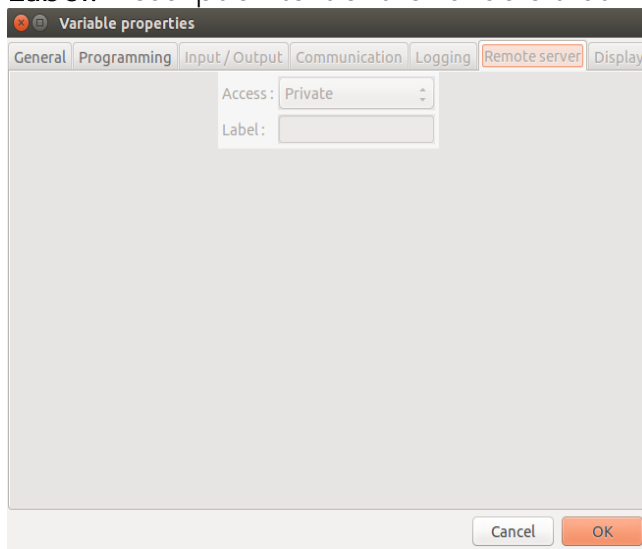
Allows to set the logging of a variable.



3.3.6 Remote server

Access: Indicates whether the variable is read-and write-accessible by MicroSERVER.

Label: Description text of the variable that will be displayed on MicroSERVER.



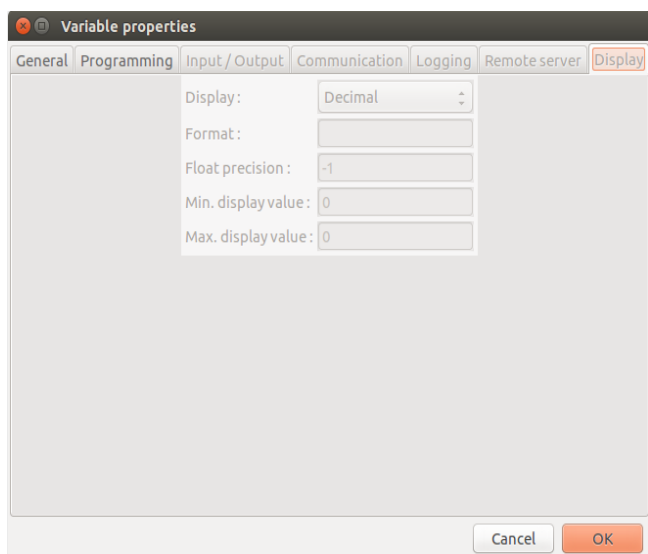
3.3.7 Display

Display: Allows to set the display format for dynamic visualization in MicroLADDER.

Format: Allows to set the display format in MicroHMI by allowing static text to be added before and after the numeric value.

Float precision : Sets the number of decimal places for floating-type values.

Min and Max display value: Used for graph bar-type graphical objects or MicroHMI curve objects. This allows to set the viewing range.



3.4 Application implementation

3.4.1 Fonction

A function is an application that is imported into another application. It is very useful when there are identical sub-parts in one or several programs. A function can make itself call to other functions.

3.4.2 Fonction implementation

Make an application and save it as a classic application without importing system code and without declaring a type of PLC.

For the declaration of variables, the field "parameter" must be filled with one of the following four possibilities:

Internal: The variable is only valid inside the function. It is initialized at each call of the function unless the "global" property is selected.

Input: When the function is called, it takes the value that the calling application gives it. It appears in the drawing of the function on the Ladder page.

Output: When the function is called, it takes the value that the calling application gives. At the end of the function's execution, the value is returned to the calling application. It appears in the drawing of the function on the Ladder page.

External: When the function is called, it takes the value that the calling application gives. At the end of the function's execution, the value is returned to the calling application. It does not appear in the drawing of the function on the Ladder page.

Notes:

- It is possible to use the Timer property in a function provided that the variable is declared with the global property. The system will take care of the decrement with the correct period, even if the block is not called regularly.
- A function cannot access the bits and system words (%S and %SW). You must pass

them as a parameter.

- Each time a function is used, it has its own memory space. This allows to use the same function several times in a program and to keep in memory from one cycle to another (if variable with the global property) data for each call.

3.4.3 Using a function

The function must be imported into the application by launching the command "Library / Import a function". Even if the function is used more than once, only one import is required.

3.4.4 Use in Ladder

Place in a program page an object "function".

For binary information it is possible to connect contacts and spools to the connections of the function. For numeric-type information, you need to double-click the function and fill in the exchanged variables. This method can also be used for binary-type information.

Use in C

Call the function by its name and pass as parameters the variables in the declaration order starting with the input variables, then the output variables and finally the external variables. This sounds like a function call in C, except that the output and external variables are refreshed, even if they are in the function call.

3.4.5 Using library

It is possible to import a library of functions. The file to import has the LIB extension. This is a specific format created by Sirea.

3.4.6 Calling a page

In a C-language page, it is possible to call another page either by its number or by its mnemonic.

Example :

```
page_2() ;  
init() ;
```

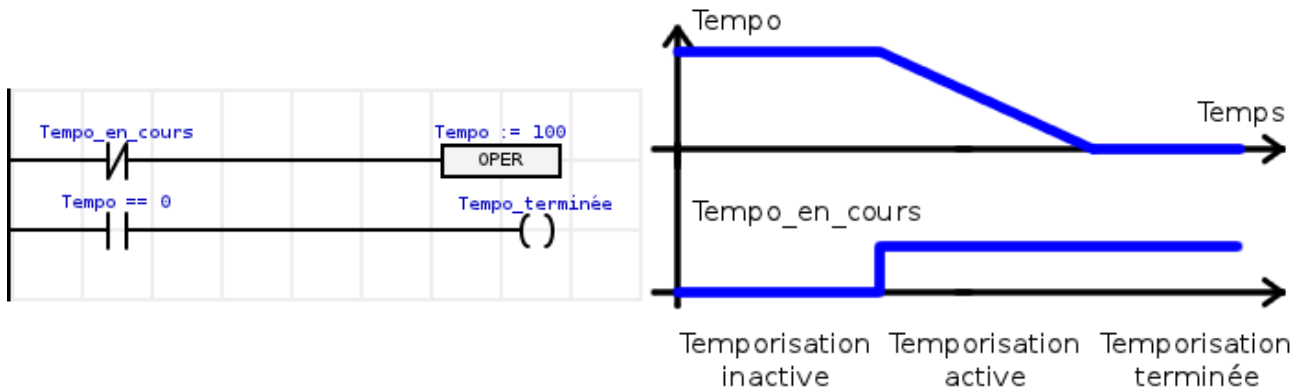
3.4.7 Temporization

To achieve temporization, the "Timer " property of a variable must be used. Just put in the property "Timer" the period (in ms) of decrement of the variable and the system manages this variable as a temporization. The duration of the temporization is equal to the value of the variable multiplied by the «Timer» property of that same variable.

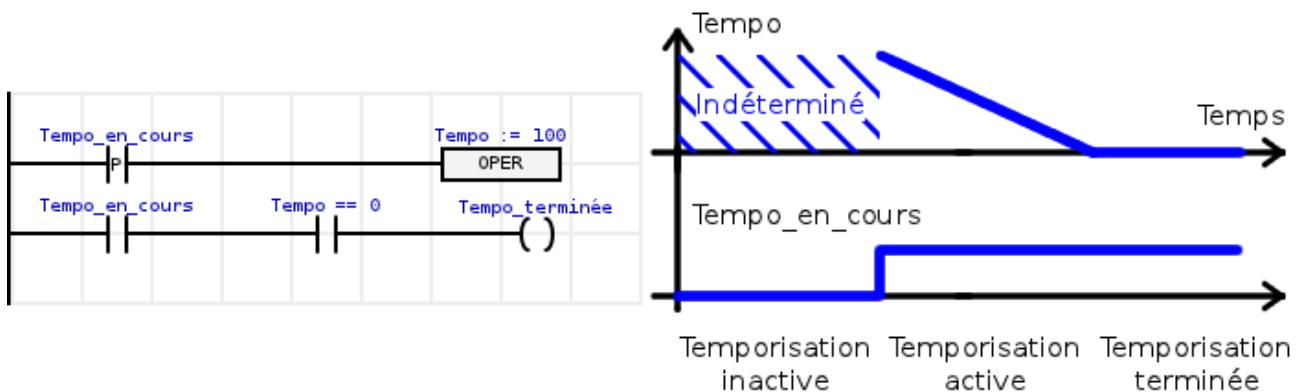
In the general case, a temporization has 3 different states: inactive temporization, active temporization and temporization completed. In MicroLADDER, there are only 2 possible states: active temporization and temporization completed.

The first solution is to force the temporization to its maximum value only and obligatory when the temporization is inactive. In the active temporization and completed temporization, do not write to the time-controlled variable. The temporization status will

be indicated when the time-controlled variable equals 0.



The second solution is to force the temporization to its maximum value when the temporization becomes active. In this case, the temporization should not be tested when it is inactive.



Example of C-language management switch (nGrafcet)

```
{
.....
    case 4 :
        //Init tempo
        Tempo:= 100;
        //Etape suivante
        nGrafcet:= 5;
    break;
    case 5 :
        if (Tempo == 0) {.....}
    break;
.....}
```

Note: It is possible to use a bit to make a timed-controlled variable. In this case, the time base is equal to the value of the temporization.

3.4.8 Global variables

It is possible to declare global variables and functions. To do this, put the code between

<global>and </global>tags. When parsing the file, this code will be moved to a globally declared area.

This brings a restriction because it is not possible to use <global>and</global> texts directly in a string character.

3.4.9 Available RAM size

The declared variable uses some RAM space, and if this variable is initialized, it uses space in flash memory.

To know the volume of RAM used, check the "size.txt" file created after each compilation. All values are expressed in bytes.

Text column: size of the program code. It should not be higher than the size of the flash memory – 64kB- used by the monitor.

Data column: size of initialized variables.

Bss column: size of non-initialized variables.

Dec column: memory size (program + variables) in decimal

Hex column: memory size (program + variables) in hexadecimal

The sum of the data column and the bss column should not exceed the RAM.

The sum of the text column and the data column should not exceed the Flash, knowing that 64 ko are already used by the monitor.

Example:

```
text  data  bss  dec  hex filename
60380   16  4644 65040 fe10 main.elf
```

Example 1. Available RAM size

text	data	bss	dec	hex	filename
60380	16	4644	65040	fe10	main.elf

Memory size:

PLC	RAM	Flash
mArm7-A1	32kB	512kB
mArm7-A2	512kB of saved RAM	512kB
mArm7-A3	32kB	512kB
mArm7-A4	32kB	512kB
mArm7-A5	32kB	512kB
mArm7-A6	64kB of saved RAM	512kB

3.5 Saved variables

3.5.1 Saved RAM

Some PLCs have a saved RAM. Just by selecting the option "Saved" inside the variable properties, the backup is autonomously managed by the system.

3.5.2 EEPROM or FRAM

If the PLC does not have a saved RAM but an EEPROM or a FRAM, it is possible to use the saved variables. When %S18 switches to 1, a backup is launched for all the variables that have the option "Saved" selected.

With the current MicroLADDER version, instructions for the manual backup should be no longer used. There is indeed a substantial risk to use memory areas used by the system.

Note: It is not recommended to do this in all running cycles, as the EEPROM is limited in number of writing and the time required to access it may be a little too long.

3.5.3 System words

%SW15 and %SW22 system words are available to the user for the backup of words on the SD card. Backups are only launched when a change in the value takes place. Backups can be launched at the end of the Modbus communication with the programming device (if the value has been forced by the communication) and at the end of the PLC cycle (if the value has been changed by the application). Each backup takes 180-200ms.

Word reading is performed transparently at the initialization.

A double backup is launched: on the SD card and on the EEPROM or FRAM.

3.6 Watchdog

3.6.1 Cycle Time Exceeded

You must set %SW1 in Ms. When %SW0 is greater than %SW1, the system sets %S15 to 1. There is no blockage of the cycle. %S15 must be delivered voluntarily to 0. If %SW1 is 0, there is no cycle time control.

3.6.2 Watchdog soft

You must set %SW2 in Ms. When %SW0 is greater than %SW2, the system performs a backup of the variables, forces the TOR and ANA outputs to 0, passes the PLC in STOP mode, sets %S21 to 1, writes 1 to the RESET_SW setting on the SD card. If %SW2 is 0, the system uses the default value (3000 ms).

3.7 System functions/Internal functions

3.7.1 Configuration of the system code

3.7.1.1 Optimize the code size

When this option is 1, the size of the code is reduced, but the execution time is increased. By default, the option is set to 0.

3.7.1.2 AUTO_STOP

When this option is set to 0, during a soft watchdog, the application restarts by itself. %S21

is set to 1 and the application must reset it to 0.

When this option is set to 1, during a soft watchdog, the application will not reboot, and the PLC will remain on the main loop. %S21 is set to 1 and is reset to 0 if the application is restarted. By default, this option is 1.

3.7.2 DHCP

When this option is set to 1, it allows the possibility to use the DHCP function (assigning IP address automatically by a DHCP server).

When this option is set to 0, it saves in code size. By default, the option is set to 1.

3.7.3 DNS

When this option is set to 1, it allows the possibility to use the DNS function (converting a literal web address to an IP address by a DNS server).

When this option is set to 0, it allows to save in code size. By default, the option is set to 1.

3.7.4 GFX

When this option is set to 1, it allows remote control of the HMI using the MPad software and the Ethernet link. An Ethernet connector socket is needed for this feature.

When this option is set to 0, it allows to save in code size. By default, the option is set to 0.

3.7.5 HTTP

When this option is set to 1, it allows access to the Web server. An Ethernet connector socket is needed for this feature. If the application has an HMI developed by MicroHMI, the MicroHMI application will be visualized (it needs 2 sockets); if the application does not have HMI, it will be an access to the WWW directory of the SD card. It is also possible to make Modbus communication through the HTTP protocol.

When this option is set to 0, it allows to save in code size. By default, the option is set to 0. See section 3.7.5 for HTTP server setting.

3.7.6 LCD

When this option is set to 1, it allows to manage the LCD screen of the PLC.

When this option is set to 0, it saves up to 50 KB of code. By default, the option is set to 1.

3.7.7 RF

It enables the management of the LoRa Radio frequency module on a pre-assigned serial port on each PLC. By default, the option is set to 0.

3.8 Communicating without protocol

To set a communication without protocol, the SERx and SOCKx system words must be set to COM_PROTOCOL_NONE (value 0).

3.8.1 Setting

3.8.1.1 ComSetCharTimeout (Parameter1, Parameter2)

This function allows you to set the inter-character timeout. This is useful with a radio

communication where the characters can arrive with a little more delay.

Parameter1: Communication port number. It is preferable to use the predefined variables COM_PORT_SER0, COM_PORT_SER1, COM_PORT_SER2, COM_PORT_SER3, COM_PORT_SER4, COM_PORT_SER5, COM_PORT_SER6, COM_PORT_SER7, COM_PORT SOCK0, COM_PORT SOCK1, COM_PORT SOCK2, COM_PORT SOCK3, COM _ PORT_WSOCK0, COM_PORT_WSOCK1, COM_PORT_WSOCK2, COM_PORT_WSOCK3.

Parameter2: The timeout value in Ms.

Example:

```
ComSetCharTimeout (COM_PORT_SER0, 100);
```

3.8.2 Transfer

3.8.2.1 ComPush (parameter1, parameter2, parameter3)

This function writes a character string into a temporary buffer. It is used before sending a character string to the PLC.

Parameter1: communication port number. It is better to use predefined variables: COM_PORT_SER0, COM_PORT_SER1, COM_PORT_SER2, COM_PORT_SER3, COM_PORT_SER4, COM_PORT_SER5, COM_PORT_SER6, COM_PORT_SER7, COM_PORT SOCK0, COM_PORT SOCK1, COM_PORT SOCK2, COM_PORT SOCK3, COM_PORT_WSOCK0, COM_PORT_WSOCK1, COM_PORT_WSOCK2, COM_PORT_WSOCK3.

Parameter2: character string of type unsigned char*.

Parameter3 : number of characters to be sent of unsigned short type

Example :

```
char s[] = "Test 123";  
ComPush (COM_PORT_SER0, s, 8)
```

3.8.2.2 ComPushByte (parameter1, parameter2)

This function writes a byte into a temporary buffer every time it is executed. This function is an alternative to ComPush, and is used before sending a character string to the PLC.

Parameter1: communication port number. It is better to use the predefined variables COM_PORT_SER0, COM_PORT_SER1, COM_PORT_SER2, COM_PORT_SER3, COM_PORT_SER4, COM_PORT_SER5, COM_PORT_SER6, COM_PORT_SER7, COM_PORT SOCK0, COM_PORT SOCK1, COM_PORT SOCK2, COM_PORT SOCK3, COM_PORT_WSOCK0, COM_PORT_WSOCK1, COM_PORT_WSOCK2, COM_PORT_WSOCK3.

Parameter2: characters of type unsigned char*.

Example

```
ComPushByte (COM_PORT_SER0, 'T');  
ComPushByte (COM_PORT_SER0, 'e');  
ComPushByte (COM_PORT_SER0, 's');  
ComPushByte (COM_PORT_SER0, 't');
```

```
ComPushByte (COM_PORT_SER0, ' ');  
ComPushByte (COM_PORT_SER0, '1');  
ComPushByte (COM_PORT_SER0, '2');  
ComPushByte (COM_PORT_SER0, '3');
```

3.8.2.3 ComSend (parameter1)

This function writes a character string previously written into a temporary buffer on the serial port, Ethernet connector or Wi-Fi.

The function ComPush or ComPushByte must be executed before; for the previous storage of the character string to be sent.

When transferring via Ethernet port or Wi-Fi, the function will not get back to the main program until end of transfer. When transferring via serial port, the function gets back to main program immediately and the transfer is performed in delayed mode during interruption. Therefore the RES_SERx system word should be checked or the ComFlushOutput function should be used to know the end of the transfer. The RES_SERx word is updated at the end of the PLC cycle.

Hence the program should not be blocked when waiting a change in status.

Parameter1: communication port number. The use of the predefined variables COM_PORT_SER0, COM_PORT_SER1, COM_PORT_SER2, COM_PORT_SER3, COM_PORT_SER4, COM_PORT_SER5, COM_PORT_SER6, COM_PORT_SER7, COM_PORT SOCK0, COM_PORT SOCK1, COM_PORT SOCK2, COM_PORT SOCK3, COM_PORT_WSOCK0, COM_PORT_WSOCK1, COM_PORT_WSOCK2, COM_PORT_WSOCK3 is preferable.

Example:

```
char s[] = "Test 123";  
ComPush (COM_PORT_SER0, s, 8);  
ComSend (COM_PORT_SER0);
```

Example: Use of ComPushByte and ComSend

```
ComPushByte (COM_PORT_SER0, 'T');  
ComPushByte (COM_PORT_SER0, 'e');  
ComPushByte (COM_PORT_SER0, 's');  
ComPushByte (COM_PORT_SER0, 't');  
ComPushByte (COM_PORT_SER0, ' ');  
ComPushByte (COM_PORT_SER0, '1');  
ComPushByte (COM_PORT_SER0, '2');  
ComPushByte (COM_PORT_SER0, '3');  
ComSend (COM_PORT_SER0);
```

3.8.2.4 ComFlushOutput (parameter1)

This function waits until the end of transfer. It is hence a blocking function.

Parameter1: communication port number. The use of the predefined variables COM_PORT_SER0, COM_PORT_SER1, COM_PORT_SER2, COM_PORT_SER3, COM_PORT_SER4, COM_PORT_SER5, COM_PORT_SER6, COM_PORT_SER7,

COM_PORT SOCK0, COM_PORT SOCK1, COM_PORT SOCK2, COM_PORT SOCK3,
COM_PORT WSOCK0, COM_PORT WSOCK1, COM_PORT WSOCK2,
COM_PORT WSOCK3 is preferable.

Example: Use of ComFlushOutput
ComFlushOutput (COM_PORT_SER1);

3.8.3 Reception

3.8.3.1 Return = ComGetFrameLength (parameter1)

This function indicates the number of characters that have been received. Reception is done in a 256 characters cyclic buffer. When the buffer is full, the reception resets to the beginning.

Return: return variable of type unsigned short. It indicates the number of characters received in the buffer. Its initial value is 0. It is increased until 255, it is then set to 256 and reset to 1. As a result, it directs to the next character position, except when indicating 256.

Parameter 1: communication port number. The use of predefined variables COM_PORT_SER0, COM_PORT_SER1, COM_PORT_SER2, COM_PORT_SER3, COM_PORT_SER4, COM_PORT_SER5, COM_PORT_SER6, COM_PORT_SER7, COM_PORT SOCK0, COM_PORT SOCK1, COM_PORT SOCK2, COM_PORT SOCK3, COM_PORT WSOCK0, COM_PORT WSOCK1, COM_PORT WSOCK2, COM_PORT WSOCK3 is preferable.

Example: Use of ComGetFrameLength
Long = ComGetFrameLength (COM_PORT_SER1);

3.8.3.2 Return = ComGetFrame (parameter1)

This function allows to retrieve any characters received. A pointer is sent to the beginning of the string of reception. After processing the characters received, the ComFlushInput must be used so that the next reception starts at the beginning of the buffer.

Return: return variable of type unsigned char*.

Parameter 1: communication port number. The use of predefined variables COM_PORT_SER0, COM_PORT_SER1, COM_PORT_SER2, COM_PORT_SER3, COM_PORT_SER4, COM_PORT_SER5, COM_PORT_SER6, COM_PORT_SER7, COM_PORT SOCK0, COM_PORT SOCK1, COM_PORT SOCK2, COM_PORT SOCK3, COM_PORT WSOCK0, COM_PORT WSOCK1, COM_PORT WSOCK2, COM_PORT WSOCK3 is preferable.

Example: Use of ComGetFrame
char *String;
String = ComGetFrame(COM_PORT_SER1);
%MW20:=String[0];
%MW21:=String[1];
%MW22:=String[2];

3.8.3.3 ComFlushInput (parameter1)

This function resets the number of characters received to 0. Additionally, for the serial ports, it deletes the string of reception.

Parameter 1: communication port number. The use of predefined variables COM_PORT_SER0, COM_PORT_SER1, COM_PORT_SER2, COM_PORT_SER3, COM_PORT_SER4, COM_PORT_SER5, COM_PORT_SER6, COM_PORT_SER7, COM_PORT SOCK0, COM_PORT SOCK1, COM_PORT SOCK2, COM_PORT SOCK3, COM_PORT_WSOCK0, COM_PORT_WSOCK1, COM_PORT_WSOCK2, COM_PORT_WSOCK3 is preferable.

Example: Use of ComFlushInput
ComFlushInput(COM_PORT_SER1);

3.8.4 Modbus

See %SW34 and %SW137 for further details. Both standard Modbus and extended Modbus (Sirea Modbus) are available to all ports. That means that an application can be loaded through all the ports. By default, all ports are set to Sirea Modbus slave number 1 for the serial ports and TCP. Modbus slave number 1 for the Ethernet ports and Wi-fi.

3.8.4.1 Slave Modbus and slave Modbus TCP

NESC_SERx, NESC SOCKx or NESC_WSOCKx should be defined just with its slave number and either SERx, SOCKx or WSOCKx with the protocol (COM_PROTOCOL_MODBUS variable in the case of standard Modbus and COM_PROTOCOL_MODBUS_TCP variable in the case of TCP Modbus). All received requests are answered by the system.

3.8.4.2 Master Modbus

SERx, SOCKx or WSOCKx should just be defined without protocol (COM_PROTOCOL_NONE variable).

NESC_SERx does not necessarily be set to 0.

Read and write frame management is subjected to the application initiative.

Note that it's easier to use lobus (automatic Modbus) instead of Modbus functions.

3.8.4.3 Return = ModbusRead (parameter1 to parameter7)

This function allows to read words or bits of a Modbus slave. Speed and communication format are set inside the system words. The function must be launched with an edge. It takes a certain time slot for rolling out. The evolution status can be known by the RES_SERx variable or the RES SOCKx inside the system words.

Return: return variable of type unsigned char. It is 1 if the function is launched. It is 0 if there is a parameter error that prevents the function from launching.

Parameter1: port number. The use of predefined variables (COM_PORT_SER0, COM_PORT_SER1, COM_PORT_SER2, COM_PORT_SER3, COM_PORT_SER4, COM_PORT_SER5, COM_PORT_SER6, COM_PORT_SER7, COM_PORT SOCK0, COM_PORT SOCK1, COM_PORT SOCK2, COM_PORT SOCK3, COM_PORT_WSOCK0, COM_PORT_WSOCK1, COM_PORT_WSOCK2, COM_PORT_WSOCK3) is preferable.

Parameter2: Modbus type (0: classical Modbus, 1: TCP Modbus)

Parameter3: slave number.

Parameter4: type of variable to be read (bit or word, memory or input). The use of predefined

variables (MODBUS_TYPE_MW, MODBUS_TYPE_M, MODBUS_TYPE_IW, MODBUS_TYPE_I) is preferable.

Parameter5: address of the first piece of data in the memory of the master for the storage.

Parameter6: number of data to be read.

Parameter7: address of the first data in the slave.

3.8.4.4 Return = ModbusWrite (parameter1 to parameter8)

This function allows to write words or bits into a Modbus slave. Speed and communication format is set inside the system words. The function must be launched with an edge. It takes a certain time slot for rolling out. The evolution status can be known by the RES_SERx, RES SOCKx or RES_WSOCKx inside the system words.

Return: return variable of type unsigned char. It is 1 if the function is launched. It is 0 if there is a parameter error that prevents the function from launching.

Parameter1: port number. The use of predefined variables (COM_PORT_SER0, COM_PORT_SER1, COM_PORT_SER2, COM_PORT_SER3, COM_PORT_SER4, COM_PORT_SER5, COM_PORT_SER6, COM_PORT_SER7, COM_PORT SOCK0, COM_PORT SOCK1, COM_PORT SOCK2, COM_PORT SOCK3, COM_PORT_WSOCK0, COM_PORT_WSOCK1, COM_PORT_WSOCK2, COM_PORT_WSOCK3) is preferable.

Parameter2: Modbus type (0: classical Modbus, 1: TCP Modbus)

Parameter3: slave number.

Parameter4: type of variable to be written (bit or word). The use of predefined variables (MODBUS_TYPE_MW, MODBUS_TYPE_M) is preferable.

Parameter5: address of the first piece of data in the memory of the master for the transfer.

Parameter6: number of data to be written.

Parameter7: address of the first data in the slave.

Parameter8: it must be set to 1 for forcing the use of the write function either of several words (function code 10h instead of 06h) or several bits (function code 0Fh instead of 05h). Otherwise set to 0 and the code function will be selected by the system depending on the number of elements to be written.

3.8.4.5 Master TCP Modbus

An opening and closing of the socket must be managed. The frame management is identical to Modbus via serial port. The definition of the two first parameters ModbusRead and ModbusWrite must be done with great care. Before each read and write process, the status of the socket must be checked as it may have lost connection.

3.8.4.6 ComConnect (parameter1, parameter2, parameter3)

This function opens the socket. The socket status must be then checked before generating frames.

Parameter1: socket number. Set COM_PORT SOCK0, COM_PORT SOCK1, COM_PORT SOCK2, COM_PORT SOCK3 for the Ethernet socket and set COM_PORT_WSOCK0, COM_PORT_WSOCK1, COM_PORT_WSOCK2, COM_PORT_WSOCK3 for the Wi-Fi socket. Cette valeur doit être en accord avec le

premier paramètre des fonctions ModbusRead et ModbusWrite.

Parameter2: A string containing the IP address or address if the DNS is enabled.

Parameter3: The port number of the client or server to connect to. For example 502 for Modbus.

3.8.4.7 ComClose (parameter1)

This function closes the socket.

Parameter1: socket number. Put COM_PORT_SOCKET0, COM_PORT_SOCKET1, COM_PORT_SOCKET2, COM_PORT_SOCKET3 for Ethernet sockets and put COM_PORT_WSOCKET0, COM_PORT_WSOCKET1, COM_PORT_WSOCKET2, COM_PORT_WSOCKET3 for Wi-Fi sockets. This value must be in accordance with the first parameter of the ModbusRead and ModbusWrite functions.

3.8.4.8 Return = ComGetSockState(parameter1)

This function reads the socket status.

Return: return variable of type unsigned char. It is SOCK_STATE_CLOSED (value 0) if the socket is closed. It is SOCK_STATE_READY (value 1) if the socket is connecting (for a function master or slave). It is SOCK_STATE_CONNECT (value 2) if the socket is open and ready to operate. It is SOCK_STATE_CLOSE (value 3) if the socket is being closed.

Parameter1: socket number. Put COM_PORT_SOCKET0, COM_PORT_SOCKET1, COM_PORT_SOCKET2, COM_PORT_SOCKET3 for Ethernet sockets and put COM_PORT_WSOCKET0, COM_PORT_WSOCKET1, COM_PORT_WSOCKET2, COM_PORT_WSOCKET3 for Wi-Fi sockets. This value must be in accordance with the first parameter of the ModbusRead and ModbusWrite functions.

Note: The attempt to connexion of a socket may fail. It is thus necessary that time-out and non-connection is managed by the application.

3.9 Files management

The SD card or USB stick must be formatted in FAT32.

Operations on the files are blocking. They are time consuming, mainly when the FAT must be browsed for finding free sectors. They cannot be processed during interrupt. Most of the management functions of the files create "ResetWatchdogs" aiming at avoiding a watchdog overrun.

The medium is divided into clusters. Each cluster is divided into 4 sectors of 512 bytes each. To write data to a sector, the complete sector must be read, bytes are then modified and the 512 bytes are written again.

The hot insertion of the SD card executes a cFatInit that lasts 1255ms. Other operations take 30 to 100ms.

3.9.1 Structure of file names

The type of storage medium (SD for a SD card or USB for a memory stick), followed by ":" the file name (8 characters maximum) and the extension (3 characters maximum) must be indicated. The system does not make the difference between uppercase and lowercase.

Example: Structure of file names

```
char *achTmpFileName = "USB:LOG.TMP";
```

```
char *achTmpFileName = "SD:LOG.TMP";
```

3.10 Structure "FSFile "

"FSFile " is a complex variable type (structure) that contains information while working on a file (from its opening to closing). There is no particular action to be made to this variable, these are the functions that will be called to work on the file that will impact this variable.

If there are several cycle laps between the opening and closing of the file, this variable must be of the global type

3.10.1 Return = FSOpen (parameter1, parameter2, parameter3)

This function allows to open the file before reading or writing.

Return: variable of type unsigned char. It is 1 when the action is correct. It is 0 if an error occurs.

Parameter1: address of a variable of type FSFile.

Parameter2: string containing the file name

Parameter3: it indicates the opening of a file for reading (FA_READ), for data appending (FA_APPEND) or for writing at the beginning of the file (FA_WRITE).

Example : Use of FSOpen

```
FSFile fd;
```

```
unsigned char result;
```

```
result = FSOpen (&fd, "SD:Sirea.TXT", FA_WRITE);
```

3.10.2 FSClose (parameter1)

This function allows to close the file once the writing process has been completed. There is no need of calling this function if the file has been opened only for reading.

Parameter1: address of a variable of type FSFile.

Example : Use of FSClose

```
FSFile fd;
```

```
FSClose (&fd);
```

3.10.3 Return = FSeek (parameter1, parameter2)

This function allows the positioning inside a file. FSOpen must be executed before.

Return: variable of type unsigned char. It is 1 when the action is correct. It is 0 when the position requested is outside the file.

Parameter1: address of a variable of type FSFile.

Parameter2: byte to be reached.

Example : Use of FSeek

```
FSFile fd;
```

```
unsigned char result;
```

```
result = FSSeek (&fd, 10);
```

3.10.4 Return = FSDelete (parameter1)

This function allows to delete a file.

Return: variable of type unsigned char. It is 1 when the action is correct. It is 0 if an error occurs.

Parameter1: string containing the file name.

Example : Use of FSDelete

```
unsigned char result; char string [20];  
unsigned char result;  
result = FSRead (&fd, string, 10);
```

3.10.5 Return = FSWrite (parameter1, parameter2, parameter3)

This function allows to write a limited number of characters. FSOpen must be executed before and FSClose, afterwards.

Return: variable of type unsigned char. It is 1 when the action is correct. It is 0 if an error occurs.

Parameter1: address of a variable of type FSFile.

Parameter2: string containing the characters to be written.

Parameter3: number of characters to be written.

Example : Use of FSWrite

```
FSFile fd;  
char *string = "Good morning";  
unsigned char result;  
chResult = FSWrite (&fd, strChaine, 7);
```

3.10.6 Return = FSRead (parameter1, parameter2, parameter3)

This function allows to read a row of data containing several columns. FSOpen must be executed before.

Return: variable of type unsigned char. It contains the number of characters read.

Parameter1: address of a variable of type FSFile.

Parameter2: string that contains the number of characters read.

Parameter3: string containing the characters to be read.

Example :

```
FSFile fd;  
char string[20];  
unsigned char result;  
unsigned short nbCol;  
result = FSReadCSVRow (&fd, &nbCol, string, 20);
```

3.10.7 Return = FSReadLine (parameter1, parameter2, parameter3, parameter4)

This function allows to read a row. FSOpen must be executed before.

Return: variable of type unsigned char. It is 1 when the action is correct. It is 0 if an error occurs.

Parameter1: address of a variable of type FSFile.

Parameter2: string containing the characters to be read.

Parameter3: number of maximum characters to be read.

Parameter4: address of a variable of type unsigned char. The function will send 0 when the returned row is complete and 1 when the returned row is incomplete.

Example : Use of FSReadLine

```
FSFile fd;  
char strChaine[20];  
unsigned char chRetour;  
unsigned char chResult;  
chResult = FSReadLine (&fd, strChaine, 20, &chRetour);
```

3.10.8 Return = FSWriteCSVRow (parameter1, parameter2, parameter3)

This function allows to write a row of data containing several columns. FSOpen must be executed before and FSClose afterwards.

Return: variable of type unsigned char. It is 1 when the action is correct. It is 0 if an error occurs.

Parameter1: address of a variable of type FSFile.

Parameter2: number of columns to be written.

Parameter3: string containing the characters to be written. Different columns are separated with 0.

Example : Use of FSWriteCSVRow

```
FSFile fd;  
Char *strChaine = "Col1-Col2";  
chaine[4] = 0;  
unsigned char chResult;  
chResult = FSWriteCSVRow (&fd, 2, strChaine);
```

3.10.9 Return = FSReadCSVRow (parameter1 to parameter4)

This function allows to read a row of data containing several columns. FSOpen must be executed before.

Return: variable of type unsigned char. It is 1 when the action is correct. It is 0 if an error occurs.

Parameter1: address of a variable of type FSFile.

Parameter2: address of a variable of type unsigned short. The function will send to this variable the number of columns found.

Parameter3: string containing the characters to be read. Different columns are separated with 0.

Parameter4: number of maximum characters to be read.

Example : Use of FSReadCSVRow

```
FSFile fd;  
char strChaine[20];  
unsigned short nbCol;  
unsigned char chResult;  
chResult = FSReadCSVRow (&fd, &nbCol, strChaine, 20);
```

3.10.10 Return = FSMove (parameter1, parameter2)

This function allows to rename a file.

Return: variable of type unsigned char. It is 1 if the action is correct. It is 0 in case of an error.

Parameter1: string containing the name of the source file.

Parameter2: string containing the name of the destination file.

Example:

```
unsigned char chResult;  
ChResult = FSMove ( "SD: LOG. TMP ", " SD: LOG2. TMP ");
```

3.10.11 Return : FSCreateFolder (parameter1)

This function is used to create a directory.

Return: variable of type unsigned char. It is 1 if the action is correct. It is 0 in case of error (Directory already existing, problem of writing on the support,...).

Parameter1: string containing the name of the directory and its path.

Example:

```
%M0: = FSCreateFolder ( "test1 ");  
%M1: = FSCreateFolder ( "test1/test2 ");  
%M2: = FSCreateFolder ( "test1/test2/test3 ");
```

3.10.12 Return: FSCopy (parameter1, parameter2)

This function is used to copy a file.

Return: variable of type unsigned char. It is 1 if the action is correct. It is 0 in case of an error.

Parameter1: string containing the name of the source file.

Parameter2: character string containing the name of the destination file.

Example:

```
unsigned char chResult;  
ChResult = FSCopy ( "SD: LOG. TMP ", " SD: LOG2. TMP ");
```

3.11 Log Management

3.11.1 Time stamp format

It is a variable of type "long" containing the number of seconds elapsed since January 1st, 1970. Functions dateToTime and timeToDate are available for doing these calculations.

Time stamp is in UTC time. The date is in local time. It is therefore influenced by the summer/winter shift and time zone.

3.11.2 "Date" structure

The following elements are contained in this structure :

Table 11. "Date" structure elements.

int sec	Seconds after minute [0, 59]
int min	Minutes after hour [0, 59]
int hour	Hour starting in midnight [0, 23]
int mday	Month day [1, 31]
int mon	Month, starting in January [1, 12]
int year	Year
int wday	Day starting in Sunday [0, 6]
int yday	Day starting on January 1st [0, 365]
int isdst	Winter or summer time

3.11.3 Return = dateToTime (parameter1)

This function converts a "Date" structure to time stamp. It works from 01/01/1970 00:00:00 (return value = 0) to 01/19/2038 03:14:07 (return value = 2147483647). Otherwise, the return is -1.

Return: variable of type long

Parameter 1: variable of structure "Date"

Example : Use of dateToTime

```
long ITSDate;  
Date d;  
d.mday = 25;  
d.mon = 12;  
d.year = 2010;  
ITSDate = dateToTime(d);
```

3.11.4 Return = timeToDate (parameter1)

This function converts a time stamp to "Date" structure.

- Value 2147483647 sends 01/19/2038 03:14:07.
- Value 0 returns 01/01/1970 00:00:00.
- Value - 2147483648 sends 01/19/2038 03:14:08.
- Value -1 sends 02/07/2106 06:28:15.

Return: variable of structure "Date"

Parameter1: time stamp

Example : Use of timeToDate

```
long ITSDate;  
Date d;  
d = timeToDate(ITSDate);
```

Heure = d.hour;

3.11.5 LogValue Format

This structure allows the retrieval of logged data. The following elements are included:

Table 12. LogValue format elements.

unsigned long time;	Time stamp of the data logging.
LogVariable *var;	Pointer over the historical data
char *string;	string of the event or alarm, This string is NULL if it is inexistent.
unsigned char type;	When retrieving events, it is LOG_ENTRY_TYPE_EVENT (value 1) in the case of a message, LOG_ENTRY_TYPE_ALARM_ON (value 2) in the case of an error appearance and LOG_ENTRY_ALARM_OFF (value 3) in the case of error disappearance.
double value;	Value of the logged variable. For the retrieval of an event, the value of this field is the value of the variable.

3.11.6 Return = LogAlQuery (parameter1, parameter2, parameter3)

This function opens the database for retrieving the logged errors. LogFetch must be then executed within the same cycle for the data retrieval.

Return: variable of type unsigned short. Number of alarms retrieved for the requested period.

Parameter1: search starting date at time stamp format. Set to 0 for unlimited.

Parameter2: search ending date at time stamp format. Set to 0 for unlimited.

Parameter3: sort order. Use LOG_QUERY_ORDER_ASC for ascending data order (oldest register in first place) or LOG_QUERY_ORDER_DESC for descending data order (oldest register in last place).

Example : Use of LogAlQuery

```
long tsStart;
long tsEnd;
unsigned short nbrErr;
nbrErr = LogAlQuery (tsStart, tsEnd, LOG_QUERY_ORDER_DESC);
```

3.11.7 Return = LogEvQuery (parameter1, parameter2, parameter3)

This function opens the database for retrieving the logged errors that appeared, that disappeared and all the messages that appeared. LogFetch must be then executed within the same cycle for the data retrieval. It is in the field «type» of the variable of type LogValue that can make the difference.

Return: variable of type unsigned short. Number of events retrieved for the requested period.

Parameter1: search starting date at time stamp format. Set to 0 for unlimited.

Parameter2: search ending date at time stamp format. Set to 0 for unlimited.

Parameter3: sort order. Use LOG_QUERY_ORDER_ASC for ascending data order (oldest register in first place) or LOG_QUERY_ORDER_DESC for descending data order (oldest register in last place).

Example : Use of LogEvQuery

```
long ITSDebut;  
long ITSFin;  
unsigned short nNbrDef;  
nNbrEv = LogEvQuery (ITSDebut, ITSFin, LOG_QUERY_ORDER_DESC);
```

3.11.8 Return = LogTrQuery (parameter1 to parameter4)

This function opens the database for retrieving a value. LogFetch must be then executed within the same cycle for the data retrieval.

Return: variable of type unsigned short. Number of values retrieved for the requested period.

Parameter 1: variable of type unsigned short. Index of the variable to be retrieved.

Parameter 2: search starting date at time stamp format. Set to 0 for unlimited.

Parameter 3: search ending date at time stamp format. Set to 0 for unlimited.

Parameter 4: sort order. Use LOG_QUERY_ORDER_ASC for data order (oldest register in first place) or LOG_QUERY_ORDER_DESC for descending data order (oldest register in last place).

Example : Use of LogTrQuery

```
long ITSDebut;  
long ITSFin;  
unsigned short nNbrDef;  
LogVariable *var = LogFindVariable("%MW20");  
nNbrTr = LogEvQuery (var, ITSDebut, ITSFin, LOG_QUERY_ORDER_DESC);
```

3.11.9 Return = LogFetch (parameter1)

This function retrieves a piece of data. This function must be called several times for retrieving several data. LogAlQuery, LogEvQuery or LogTrQuery must be used just before retrieving a data package.

Every data package must be retrieved in the same cycle.

Return: variable of type unsigned char. It is 0 when a problem occurs.

Parameter 1: structure of type LogValue. It will contain the data.

Example : Use of LogFetch

```
LogValue stDef;  
LogFetch (&stDef);
```

3.11.10 Return = LogAlSave (parameter1, parameter2, parameter3)

This function allows to save alarms to an SD card or a memory stick.

Return: variable of type unsigned char. It is 0 when a problem occurs.

Parameter1: string containing the filename.

Parameter2: search starting date at time stamp format. Set to 0 for unlimited.

Parameter 3: search ending date at time stamp format. Set to 0 for unlimited.

Example: Use of LogAlSave

```
long ITSDebut;  
long ITSFin;  
unsigned short ret;  
ret = LogAlSave ("SD:Alarme.csv", ITSDebut, ITSFin);
```

3.11.11 Return = LogEvSave (parameter1, parameter2, parameter3)

This function allows to save events to an SD card or a memory stick.

Return: variable of type unsigned char. It is 0 when a problem occurs.

Parameter1: string containing the filename.

Parameter2: search starting date at time stamp format. Set to 0 for unlimited.

Parameter 3: search ending date at time stamp format. Set to 0 for unlimited.

Example : Use of LogEvSave

```
long ITSDebut;  
long ITSFin;  
unsigned short ret;  
ret = LogEvSave ("SD:Evenem.csv", ITSDebut, ITSFin);
```

3.11.12 Return = LogTrSave (parameter1 to parameter4)

This function allows to backup values to an SD card or a memory stick. "LogFindVariable" should be used before calling this function.

Return: variable of type unsigned char. It is 0 when a problem occurs.

Parameter1: string containing the filename.

Parameter2: number of the logged variable.

Parameter3: search starting date at time stamp format. Set to 0 for unlimited.

Parameter4: search ending date at time stamp format. Set to 0 for unlimited.

Example: Use of LogTrSave

```
long ITSDebut;  
long ITSFin;  
unsigned short ret;  
LogVariable *var = LogFindVariable("%MW20");  
if (var) {ret = LogTrSave ("SD:Valeur.csv", var, ITSDebut, ITSFin);}
```

3.11.13 Return = LogSave (parameter1)

This function is used to export the result of a history query to an already opened file.

Return: variable of type unsigned char. It's worth 0 if there's a problem.

Parameter1: The address of a variable of type FSFile.

Example:

```
FSFile fd;
```



```
char chRet
long ITSDebut;
long ITSFin;
unsigned short nNbrDef;
LogVariable *var = LogFindVariable("%MW20");
nNbrTr = LogTrQuery (var, ITSDebut, ITSFin, LOG_QUERY_ORDER_DESC);
if (FSOpen (&fd, "SD:Extract.csv", FA_WRITE))
{
    chRet = LogSave (&fd);
    FSClose (&fd);
}
```

3.11.14 LogPurge ()

This function deletes all the events and curves of all variables.

Example : Use of LogPurge

```
LogPurge ();
```

3.11.15 LogEvPurge ()

This function deletes all the events.

Example : Use of LogEvPurge

```
LogEvPurge ();
```

3.11.16 Return = LogTrPurge (parameter1)

This function deletes the complete log of a variable. "LogFindVariable" should be used before calling this function.

Return: variable of type unsigned char. It is 1 when the variable exists, 0 otherwise.

Parameter1: number of the logged variable.

Example : Use of LogTrPurge

```
LogVariable *var = LogFindVariable("%MW20");
ret = LogTrPurge (var);
```

4 IO Bus

The IO Bus relies on the Modbus protocol to communicate between a master and slaves. The principle is to make the connection between the variables of the master and those of the slave and the system automatically manages the communication frames to update these variables. Everything takes place on the master side. The slave simply responds to the frames as in the Modbus standard.

4.1 Declaration of slave Equipment

The screenshot shows the 'Device properties' window with the following values:

- Index: 1
- Label: Translator
- Local port: Serial
- Serial settings: 19200, 8N1
- Protocol: Modbus
- Slave number: 1
- Delay (in ms): 50
- Timeout (in ms): 500
- Max frame length: (empty)
- ☐ Group variables

In «program/equipment's/Add remote equipment », fill in the information concerning communication with the slave.

Index: Each equipment is marked with a unique number.

Label: Text that serves only as a helper and does not affect the operation.

Port: Choice between serial, Ethernet, Wi-Fi or Radio. For a serial port, you the port number, the speed and the communication format must be specified. For an Ethernet or Wi-Fi port, the IP address of the slave must specified. For a radio port, the frequency channel must be specified.

Protocol: Choose between Modbus (this is the standard Modbus), Sirea (It is the extended Modbus which can only be used with the Sirea device) or Modbus (inverted words) (words that make up long integers and floats are inverted in relation to the Modbus Standard).

Slave number: This is the Modbus slave number.

Delay: This is the interframe delay of the communication in MS.

Timeout: This is the time between the start of waiting for a response and the passing in error in order to continue the communication.

Maximum length of a frame: the maximum number of words in a frame.

Group variables: Allows when reading data to make longer frames but less numerous by making the reading of uninteresting data.

4.2 Declaration of variables

In the Declaration of variables, in the "Communication " tab, the field "remote address " must be filled with the address of the slave followed by a point and the equipment number.

Example: %MW16.1 (the master variable will be linked to the %MW16 variable of equipment 1).

Also, in the "Communication " tab, it is possible to scale the value.

4.3 State of communication with equipment

There is information on the state of communication with equipment. They are contained in a "LogStats " structure. To be able to read this structure, you first have to get a pointer over the equipment and then get a pointer over the statistical structure of the equipment.

Sirea has made available a whole library of function block allowing to get simply the state of communication.

The communication can be done by serial port if there is a GPRS modem, but this option may disappear in the next version of the software. More generally, use ethernet port.

Example to obtain the statistical structure of equipment 1:

```
LogDevice *dev = LogGetDevice (1);
```

```
LogStats stats = LogGetStats (dev);
```

Structure "LogStats" content

unsigned long lastTimeOK	Time stamp of the last frame OK
unsigned long lastTimeERR	Error on the Time stamp of the last frame
unsigned char lastState	State of the last frame (0 : no communication, 1 : communication OK, more than 250 : Error)
unsigned long nbFramesOK	Number of frame OK
unsigned long nbFramesERR	Error on the number of frame

Example to mount a defect after 5 seconds without proper frame:

TDefComPV is a timed variable.

```
if (stats.lastState == 1 || %S16) {TDefComPV := 5;}
```

```
if (TDefComPV == 0) {BDefComPV := 1;} else {BDefComPV := 0;}
```

5 HTTP Protocol

For a PLC with an Ethernet port, it is possible to develop a Web server with MicroHMI.

Two sockets must be used. The first is set in COM_PROTOCOL_HTTP mode with its port at 80 and the next is set to COM_PROTOCOL_HTTP_QUERY mode with its port at 81. Using 2 sockets allows you to have a more fluid display and to properly load the images to be displayed.

If the PLC has an Ethernet port, the system code HTTP option is enabled, and the application does not have an HMI, the Web server will be able to retrieve the files in the WWW directory of the SD card. Just indicate in a browser the IP address of the PLC and the name of the file.

6 Wi-Fi

6.1 Mode

There is the Ad Hoc mode to make a Wi-Fi access point from a PLC and client mode.

This is the %SW95 (W_MODE) variable that is used to manage it.

Constant	Value	Description
WSOCK_MODE_OFF	0	No connection Wi-Fi

WSOCK_MODE_STA	1	Client mode connection
WSOCK_MODE_AP	2	Ad Hoc mode connection

6.2 WSocketSSID (parameter 1)

This function allows you to initialize the name of the Wi-Fi station, both in client mode and Ad Hoc mode.

This property can also be set with the "W_SSID " parameter of the file "main. CFG ". The standard allows up to 32 characters.

Parameter1: A string containing the network SSID.

Example:

```
WSocketSSID (TP-LINK _ 9B4144);
```

6.3 WSocketSecKey (parameter 1)

This function allows you to set the security key, both in client mode and Ad Hoc mode.

This property can also be set with the "W_SKEY " parameter of the file "main. CFG ".

Parameter1 : Character string containing the security key

Example :

```
WSocketSecKey ("123456") ;
```

6.4 WSocketSecType (parameter 1)

This function is used to adjust the type of encryption. It is especially useful in Ad Hoc mode because in client mode, it is enough not to call it to use automatic detection.

This property can also be set with the "W_STYPE " parameter of the file "main. CFG ".

Parameter1: A string containing the type of encryption. Possible values are "" (Empty character string uses automatic detection), OPEN (no encryption), WEP, WPA, WPAAES, WPA2AES, WPA2TKIP, WPA2.

Example:

```
WSocketSecType ("");
```

6.5 WSocketKey (parameter 1, parameter 2)

This function allows to set the encryption type and the security key. This is the equivalent of WSocketSecKey and WSocketSecType.

Parameter1: A string containing the type of encryption. Possible values are "" (Empty character string uses automatic detection), OPEN (no encryption), WEP, WPA, WPAAES, WPA2AES, WPA2TKIP, WPA2.

Parameter2: Character string containing the security key

Example:

```
WSocketKey ( "", "123456 ");
```

7 Connection to a server

It is very easy to set up a connection to a MicroSERVER in order to trace data. The data to be remounted must be set up in the tabs "logging " and "remote server ". This communication can be done through a serial port if there is a GPRS modem behind.

7.1 Setting

The Setup window is accessible in MicroLADDER in "program/equipment/Basic equipment".

Device properties

Type : Manager

Label : Castres

Memory size (in bytes) : 1024

Remote index : 850

Local port : Ethernet 3

µServer's address : 194.117.213.206:13214

µServer communication timeout (in seconds) : 60

µServer's password : mdp

Cancel OK

The parameters are identical to the use of the function "LogSetRemoteDevice " described below.

Memory size (in bytes): for PLC that do not have saved RAM, you must specify the size of RAM allocated to memorizing the histories. For PLC with a saved RAM, the entire RAM is used.

It is not necessary to specify the port of the server if it is the default port (13214) that is used.

7.2 Setting by programming

If there are connection parameters that can evolve, it is possible to call the LogSetRemoteDevice function (Parameter1, Parameter2, Parameter3, Parameter4, Parameter5, Parameter6, Parameter7, Parameter8, Parameter9, Parameter10, Parameter11). This function is active on the front. It is called only once or if the communication parameters changes.

Parameter1: A long integer indicating the index of the device that wants to connect to the server.

Parameter2: A string containing the device type. This information is only used to make the display on the server.

Parameter 3: A character string containing the label or location. This information is only

used to make the display on the server.

Parameter 4: Identification of the communication port used for communication.

Parameter 5: Indication of the speed of communication. For a communication in Ethernet or Wi-Fi, it is necessary to put COM_SPEED_NONE.

Parameter 6: Communication format. For a communication in Ethernet or Wi-Fi, it is necessary to put COM_FORMAT_NONE.

Parameter 7: A string containing the server address.

Parameter 8: remote server port. By default, it's 13214.

Parameter 9: Identification of the radio frequency in case of using lora.

Parameter 10: An unsigned long integer containing the time out in seconds.

Parameter 11: A character string containing the password.

```
Example: LogSetRemoteDevice (850, "Gestionnaire", "Castres", COM_PORT_SOCK3,
COM_SPEED_NONE, COM_FORMAT_NONE, "194.117.213.206", 13214,
RF_FREQ_NONE, 60, "mdp") ;
LogSetRemoteDevice (850, "Gestionnaire", "Castres", COM_PORT_SER2,
COM_SPEED_9600, COM_FORMAT_8N1, "194.117.213.206", 13214, RF_FREQ_NONE, 60,
"mdp") ;
```

It is not necessary to set up the protocol of the Ethernet socket or Wi-Fi, the function does. To know the status of the connection, you must test the variable "LogDataPortStep ". It is greater than or equal to "LOG_DATA_PORT_STEP_WAIT_CMD " (value 6) and lower than LOG_DATA_PORT_STEP_RESET (value 11) when everything is fine.

The connection settings are saved. It is not necessary to restart the function "LogSetRemoteDevice " every time the application is restarted, but only after a load.

8 History Management

On all PLC it is possible to manage histories. If there is no saved RAM, the data will be lost in case of power outage.

It is possible to have events (memorization of all the dates of appearance and disappearance of faults, memorization of all the dates of appearance of the messages), alarms (recovery of the faults present and the date appearing) and the traces (with dates and values of a data).

This information is stored in memory. It is then possible to read these data and save it on the SD card or the USB stick. It is also possible to retrieve this information on an external system using a specific protocol (see the paragraph concerning the connection to a server).

Setting the history

The settings are in the "Logging " tab of the properties of a variable.

8.1 Alarm

Condition of the alarm:

None: No memorization

<, ≤, >, ≥, !=, ==: type of comparison to cause the alarm to be memorized

Alarm threshold: value to compare to cause the alarm to be memorised

Alarm appearance wording: Text used when alarm appears

Alarm disappearing wording: text used when the alarm disappears

8.2 Event

Condition of the event:

None: No memorization

<, ≤, >, ≥, !=, ==: type of comparison to cause the event to be memorized

Event threshold: The value to compare to cause the event to be memorized

Event Labelling: Text used when the event appears

8.3 Curve

Type of logging:

None: No logging

Standard: The value is used as it is

Averaged: The value is averaged over the duration of logging

Period of logging (in second): See explanation below

Threshold of logging: see explanation below

For a curve data to be memorized, the elapsed time must be greater than or equal to the set time and the delta of value is greater than or equal to the parameter threshold.

In the case of an unaveraged value, at the end of the period, if the delta of variation exceeds the hysteresis, this value is recorded. In the case of an averaged value, a mean is made during the period, at the end of the period, if the delta of variation exceeds the hysteresis, there is a recording of this value.

8.4 Return = LogFindVariable (parameter1)

This function is used to point the LogVariable structure of a historical variable. It must be used before doing an operation on a historical variable.

Return: pointer-type variable on a LogVariable structure.

Parameter1: A string containing the variable.

Example:

```
LogVariable * var = LogFindVariable ("%MW20 ");
```

8.5 Return = LogGetVariable (parameter 1)

This function does the same thing as the LogFindVariable function but from the index of the historical variable. The indexes to use this function must be known, but the indexes are assigned to compile and therefore unknown when writing the program.

Return: pointer-type variable on a LogVariable structure.

Parameter1: The index of the historical variable.

Example :

```
LogVariable * var = LogGetVariable (4);
```


9 Functions for character strings

9.1 Return = StrToNum (parameter1)

This function converts a string of characters to a numeric value with comma. It only deals with the first numeric characters. Writing « strToNum » also works,

Return: dual-type variable.

Parameter1: The character string to convert.

Example:

```
int nEntier = StrToNum ( "-200 ");  
float fFlottant = StrToNum ( "3.14 ");
```

9.2 Sprintf (parameter1, parameter2, parameter 3)

This function is similar to the standard C-language function "sprintf ". It is used to populate strings with numeric values and/or string characters.

This function can be replaced by StrSet, which in addition makes character string length controls.

Parameter1: A pointer to the string to fill.

Parameter2: The structure of the character string to create with the constant elements and the type of the variable elements. Example of variable element types: % c (character), % l or % d (integer), % s (string), % f (floating).

Parameter 3 and following: A list of the variable elements to be put in the character string.

Example:

```
char * strUnit = "Bar";  
Char strValeur [20];  
int nMesure;  
sprintf (strValeur, "measure =% i% s ", NMesure, StrUnit);
```

9.3 Upper (parameter1)

This function replaces the lowercase characters in a character string with uppercase characters.

This function is replaced by StrToUpper, but the old function can also be used.

Parameter1: pointer to character string

Example:

```
char * strTexte = "Hello ";  
Upper (StrTexte);
```

9.4 Lower (parameter1)

This function replaces the uppercase characters in a character string with lowercase characters.

This function is replaced by StrToLower, but the old function can also be used.

Parameter1: pointer to character string

Example:

```
char * strTexte = "Hello ";  
Upper (StrTexte);
```

9.5 StrSet (parameter1, parameter2, parameter3)

This function is identical to sprintf but it does character string size checks. So it's better to use this function.

9.6 StrToLower (parameter1)

This function is identical to lower.

9.7 StrToUpper (parameter1)

This function is identical to upper.

9.8 Return = StrGetChar (parameter1, parameter2)

This function is used to retrieve a character within a string.

Return: char variable containing the recovered character or 0 if the requested position is outside the character string.

Parameter1: A pointer to the character string to examine.

Parameter2: A variable of type unsigned long indicating the place of the searched character in the character string.

Example:

```
Char ChCar;  
chCar = StrGetChar(%MS20, 10) ;
```

9.9 Return = StrSetChar (parameter1, parameter2, parameter3)

This function allows you to change a character within a string. It is secure in relation to the length of the character string.

Return: variable of type unsigned char. It's worth 1 if the action is correct. It's worth 0 if the action couldn't be done.

Parameter1: A pointer to the character string to examine.

Parameter2: A variable of type unsigned long indicating the place of the character to be changed in the string.

Parameter3: Character to modify

Example:

```
char ChCar = ' C ' ;  
unsigned char chRep = StrSetChar (%MS20, 10, ChCar);
```