µLadder

User Manual

©SIREA - 69 RUE DE L'INDUSTRIE - ZI DE MELOU - 81100 CASTRES-FRANCE Tél. +33 (0)5 63 72 93 92 Fax: + 33 (0)5 63 72 93 19

TABLE OF CONTENTS

INDEX OF EXAMPLES

INDEX OF FIGURES

INDEX OF TABLES

I. INTRODUCTION

This document provides a guide the use of µLadder V8 software and associated software for the programming of PLCs (Programmable Logic Controllers) from mArm7 family.

Special features of each PLC are described within a separated document.

Document version	04/22/14

©SIREA - 69 Rue de l'Industrie - ZI DE MELOU - 81100 CASTRES-FRANCE

Tél. +33 (0)5 63 72 93 92 Fax: + 33 (0)5 63 72 93 19

II. SOFTWARE ENVIRONMENT

1. Installation

1.1. System requirements

µLadder can be installed either on Windows or Linux operating systems.

On Windows 7 and 8, it is necessary to install it with administrator privileges.

1.2. Compiler

Both Windows and Linux use the GCC compiler, version 4.6.2. For Windows, execute the installation file "gcc-arm-none-eabi-4_6-2012q4-20121016.exe". Select "Add path to environment variable" at the end of the installation and close the command window that remains open. For Linux, execute the installation file "gcc-arm-none-eabi-4.6.2.deb". On Linux, the old version of GCC compiler can be used.

Choose the installation required according to your operating system.

Note: on Windows 7 and Windows 8, the installation must be performed from the Administrator account and access has to be provided to all users.

1.3. µLadder

This software can be used for creating the application, compiling it, doing dynamic visualization and forcing variables. For the installation of µLadder on Windows, execute the file "setup-mladder-**.exe ", where ** is the version number. On Linux, execute the file "mladder-**.deb", where ** is the version number. µLadder software versions under development do not include the version number.

1.4. µControl

This software can be used for loading the application, doing dynamic visualization and forcing variables. Version 6 or more recent needs to be installed.

For the installation of µControl on Windows, execute the file "setup-mcontrol-**.exe ", where ** is the version number. On Linux, execute the file "mcontrol-**.deb", where ** is the version number. µControl software versions under development do not include the version number.

1.5. µDriver

This software is used by µControl and µLadder for the communication with the PLC. Version 5.3 or more recent needs to be installed.

For the installation of µDriver 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. µDriver software versions under development do not include the version number.

1.6. µIHM

This software is used for creating IHM for certain PLCs that include a touchscreen (specifically, µArmA2, µArmA8 and µArmA9, PLCs). This software is complementary to µLadder, and has been conceived for the development of Human–Computer Interaction interfaces, HCI, (Interface Homme Machine, IHM, in French).

For the installation of µIHM on Windows, execute the file "setup-mihm-**.exe", where ** is the version number. On Linux, execute the file "mihm-**.deb", where ** is the version number. µIHM software versions under development do not include the version number.

2. System architecture

2.1. Monitor software

The monitor (or monitor software) is the base software layer, which is loaded by Sirea. The user will not need to modify it unless a new software upload may require to reload 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. From this moment, it is possible to operate without the SD card.

2.2. "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 rewritten in full. This file should not be modified. Some characteristics from this file can be changed through variables if it's needed (see Table 1).

Table 1.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 have to 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.

Variables for ports configuration are described on the section on *System words.*

Saved variables are described on the section on Saved variables.

MAIN.CFG file is shown next with default values, when being these values inexistent. This shows that, when a MAIN.CFG file is not found on the SD card, the PLC launches the main loop of the application available on the memory. If there is no application available, the system will lock out.

```
FORCE_MON=0
```

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

2.3. Firmware

From version 6 of µLadder on, there is no need to install the firmware. The firmware can be imported to the application. This will avoid problems when choosing the suitable version for the compilation.

The file to be imported is the "mArm.sys". Version 8 of µLadder needs at least version 4.0 of firmware.

2.4. Loading applications

It is not necessary to have a SD card. When having a SD card, it must be formatted into FAT32.

2.4.1. µControl

µControl software enables to load ".hex" files, that means, already compiled files. 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 µControl.

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 inexistent.

2.4.2. SD card

If the PLC includes a SD card you can just copy the file compiled by µLadder ("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 for loading of 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 and main.cfg on the PC and then he only transfers the .hex file, but not the main.cfg generated by µLadder.
- 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. For starting the application, the loading button must be then pressed during 3 seconds. The operating status LED indicates the starting of the application with a blinking, mainly with the status LED light on.

2.5. PLC operating status LED

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

2.5.1. Monitor mode

The LED blinks slowly : 500ms on, 500ms off.

The PLC is waiting in Modbus mode. It can be controlled with µControl.

2.5.2. Transfer of the program from SD card to PLC memory

The LED blinks fast: 100ms on, 100ms off.

The transfer lasts some seconds. It takes place after loading with µControl, either on loading request through the button or on loading request through the SD card parameters. For further information regarding loading details, see section on *Loading applications*.

2.5.3. Application in STOP mode

The LED blinks slowly, mainly in off mode: 100ms on, 900ms off.

The PLC is no longer on the monitor, the main loop has been launched, the application is not executed.

2.5.4. Application in RUN mode

The LED blinks slowly, mainly in on mode: 900ms on, 100ms off.

The application has been executed.

The fixed red LED is on.

2.5.5. Incompatibility of versions between monitor and application

This LED mode could only take place when booting the application. This could cause problems on the saved variables.

3. Type of data

It is important to remark that µLadder 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.

3.1. 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.

3.2. ANALOG inputs

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

Prefix for ANALOG inputs: %IW (IW: Input words). Example: %IW100.

Certain PLCs allow the configuration of the analog inputs through the use of jumpers. The property “configuration” of analog variables must be then adjusted with µLadder to meet the proper value range and calibration needs.

Table 2. 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.*

3.3. DIGITAL outputs

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

Prefix for DIGITAL outputs: %Q (Q: Output bool). Example %Q0.

3.4. ANALOG outputs

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

Prefix for ANALOG outputs: %QW (QW: Output words). Example %QW100.

3.5. 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 to 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 1kΩ must be connected to the output terminals.
- The output PWM signal varies between 0V and Vcc (supply voltage).

3.6. 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 that accessing a byte or a word.

3.7. Integer

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

3.8. Long

Value range from – 2 147 483 648 to 2 147 483 647.

Prefix for long: %MD (MD: Main Double). Example : %MD0.

3.9. Float

Value range from $3,4*10^{-38}$ to $3,4*10^{38}$, both positive and negative values.

Prefix for float: %MF (MF: Main Float). Example : %MF0.

3.10. 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.

3.11. System bits

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 reestablished to 0 automatically after executing the first cycle.

Bit	Mnemonic	Description
		This bit is slightly different from bit %S16 (DEC_M).
%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. Reestablished to 0 automatically at the end of the initialization.
%S3		Spare
%S4		Spare
%S5 %S6 %S7 %S8	MSEC_10 MSEC_100 SEC_1 MIN_1	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.
%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

Bit	Mnemonic	Description
%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 current status Bit to 0: outputs are updated by the user program
%S13		Spare
%S14		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 reestablished 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 reestablished 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.

Bit	Mnemonic	Description
		It causes the re-reading of “var.csv” file and the reset of the alarms and events history report.
%S20		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 reestablished to 1 by the system when the program restarts. See Mx_CYC : %SW2 and section on *Watchdog* for further information.
%S22		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 to %S49		Spare

3.12. System words

Word	Mnemonic	Description
%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.

Word	Mnemonic	Description
%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		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.
%SW11	MODE_DST	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		Spare
%SW13	VERSION_SYS	System code version. Only read access.

Word	Mnemonic	Description
%SW14	VERSION_APP	Application version. This word is available to the user for the record of the application version number.
%SW15 %SW16 %SW17 %SW18 %SW19 %SW20 %SW21 %SW22	SAV1 SAV2 SAV3 SAV4 SAV5 SAV6 SAV7 SAV8	These 8 words are available to the user for word backup. See section on *Saved variables*for further details.
%SW23		Spare
%SW24		Spare
%SW25	FREQ_TIMER	Calling period of interrupting functions in µs. Functions to be called need to have the "Call on interrupt" property confirmed. Value 0: no calling Value different to 0: calling period in µs. 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.

Word	Mnemonic	Description
		Reading mask. Allows to unable the inputs refreshing. Each input is linked to a bit. IHM inputs are not affected.
%SW27	FET1	1st and 2nd input bytes
%SW28	FET2	3ird and 4th input bytes
%SW29	FET3	5th and 6th input bytes
%SW30	FET4	7th and 8th input bytes
%SW31	FET5	9th and 10th input bytes
%SW32	FET6	11th and 12th input bytes Note : a 16 bits input card straddles 2 system words.
%SW33		Spare
%SW34	SER0	Port configuration (see Modbus details below)
%SW35	NESC_SER0	Slave number (variable saved on the SD card)
%SW36	RES_SER0	result of the exchange (see Modbus details below)
%SW37	SPD_SER0	Speed code (see Modbus details below)
%SW38	FOR_SER0	Communication format (see Modbus details below)
%SW39	TIMEOUT_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.

Word	Mnemonic	Description
%SW40	SER1	Port configuration (see Modbus details below)
%SW41	NESC_SER1	Slave number (variable saved on the SD card)
%SW42	RES_SER1	result of the exchange (see Modbus details below)
%SW43	SPD_SER1	Speed code (see Modbus details below)
%SW44	FOR_SER1	Communication format (see Modbus details below)
%SW45	TIMEOUT_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.
%SW46	SER2	Port configuration (see Modbus details below)
%SW47	NESC_SER2	Slave number (variable saved on the SD card)
%SW48	RES_SER2	result of the exchange (see Modbus details below)
%SW49	SPD_SER2	Speed code (see Modbus details below)
%SW50	FOR_SER2	Communication format (see Modbus details below)
%SW51	TIMEOUT_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.
%SW52	SER3	Port configuration (see Modbus details below)
%SW53	NESC_SER3	Slave number (variable saved on the SD card)
%SW54	RES_SER3	result of the exchange (see Modbus details below)
%SW55	SPD_SER3	Speed code (see Modbus details below)
%SW56	FOR_SER3	Communication format (see Modbus details below)
%SW57	TIMEOUT_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.

Word	Mnemonic	Description
%SW58	SER4	Port configuration (see Modbus details below)
%SW59	NESC_SER4	Slave number (variable saved on the SD card)
%SW60	RES_SER4	result of the exchange (see Modbus details below)
%SW61	SPD_SER4	Speed code (see Modbus details below)
%SW62	FOR_SER4	Communication format (see Modbus details below)
%SW63	TIMEOUT_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.
%SW64	SER5	Port configuration (see Modbus details below)
%SW65	NESC_SER5	Slave number (variable saved on the SD card)
%SW66	RES_SER5	result of the exchange (see Modbus details below)
%SW67	SPD_SER5	Speed code (see Modbus details below)
%SW68	FOR_SER5	Communication format (see Modbus details below)
%SW69	TIMEOUT_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.
%SW70	SER6	Port configuration (see Modbus details below)
%SW71	NESC_SER6	Slave number (variable saved on the SD card)
%SW72	RES_SER6	result of the exchange (see Modbus details below)
%SW73	SPD_SER6	Speed code (see Modbus details below)
%SW74	FOR_SER6	Communication format (see Modbus details below)
%SW75	TIMEOUT_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.

Word	Mnemonic	Description
%SW76	SER7	Port configuration (see Modbus details below)
%SW77	NESC_SER7	Slave number (variable saved on the SD card)
%SW78	RES_SER7	result of the exchange (see Modbus details below)
%SW79	SPD_SER7	Speed code (see Modbus details below)
%SW80	FOR_SER7	Communication format (see Modbus details below)
%SW81	TIMEOUT_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.
%SW100	MAC_ADDR1	Ethernet connector MAC address.
%SW101	MAC_ADDR2	
%SW102	MAC_ADDR3	
%SW103	MAC_ADDR4	
%SW104	MAC_ADDR5	
%SW105	MAC_ADDR6	
%SW106	IP_ADDR1	Ethernet connector IP address.
%SW107	IP_ADDR2	
%SW108	IP_ADDR3	
%SW109	IP_ADDR4	
%SW110	SUBNET_MASK1	Ethernet connector subnet mask.
%SW111	SUBNET_MASK2	
%SW112	SUBNET_MASK3	
%SW113	SUBNET_MASK4	

Word	Mnemonic	Description
%SW114	GATEWAY1	Ethernet connector gateway.
%SW115	GATEWAY2	
%SW116	GATEWAY3	
%SW117	GATEWAY4	
%SW118	SOCK0	Socket communication protocol (see Modbus details below)
%SW119	PORT_SOCK0	Socket port number in slave mode (502 in Modbus, 10000 in VNC, 80 in HTTP, 161 in SNMP)
%SW120	NESC_SOCK0	Socket slave number
%SW121	RES_SOCK0	Socket result of the exchange
%SW122	TIMEOUT_SOCK0	Socket time out
%SW123	SOCK1	Socket communication protocol (see Modbus details below)
%SW124	PORT_SOCK1	Socket port number in slave mode (502 in Modbus, 10000 in VNC, 80 in HTTP, 161 in SNMP)
%SW125	NESC_SOCK1	Socket slave number
%SW126	RES_SOCK1	Socket result of the exchange
%SW127	TIMEOUT_SOCK1	Socket time out
%SW128	SOCK2	Socket communication protocol (see Modbus details below)
%SW129	PORT_SOCK2	Socket port number in slave mode (502 in Modbus, 10000 in VNC, 80 in HTTP, 161 in SNMP)
%SW130	NESC_SOCK2	Socket slave number
%SW131	RES_SOCK2	Socket result of the exchange
%SW132	TIMEOUT_SOCK2	Socket time out

Word	Mnemonic	Description
%SW133	SOCK3	Socket communication protocol (see Modbus details below)
%SW134	PORT_SOCK3	Socket port number in slave mode (502 in Modbus, 10000 in VNC, 80 in HTTP, 161 in SNMP)
%SW135	NESC_SOCK3	Socket slave number
%SW136	RES_SOCK3	Socket result of the exchange
%SW137	TIMEOUT_SOCK3	Socket time out

Modbus Detail

SER0 to SER7 SOCK0 to SOCK3	Port configuration. These variables are not saved.	
Value name	Variable value	Description
COM_PROTOCOL_NONE COM_PROTOCOL_TCP	0	No protocol or Master Modbus
COM_PROTOCOL_IOBUS	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	6	UDP master protocol
COM_PROTOCOL_SNMP	7	SNMP protocol

Note:

- Default value for serial ports is Modbus. MODBUS_TCP for socket 0 and HTTP for sockets 1 to 3, with the exception of 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 to NESC_SER7 NESC_SOCK0 to NESC_SOCK3	Slave number for the slave Modbus mode*	
Value name	Variable value	Description
	0	Only master Modbus
	1 to 255	Modbus slave number

**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.*

Note:

- Default value is 1.

RES_SER0 to RES_SER7 RES_SOCK0 to RES_SOCK3	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

RES_SER0 to RES_SER7 RES_SOCK0 to RES_SOCK3	Result of the exchange. It runs either in transfer or reception.	
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 to SPD_SER7	Exchange speed*	
Value name	Variable value	Description
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

**These variables are saved on the SD card.*

SPD_SER0 to SPD_SER7	Exchange speed	
COM_SPEED_115200	10	115200 bauds

Note:

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

FOR_SER0 to FOR_SER7	Exchange format parameter*	
Value name	Variable value	Description
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

** These variables are saved on the SD card.*

Note:

- Default value is 2.

3.13. Available RAM size

The firmware takes some RAM space, similarly as each declared variable in µLadder does. It is not the case, though, of C variables declared in C pages as they are local variables declared in the battery memory that disappear at the end of the page execution. The system additionally uses RAM for its variables.

To know the RAM used, the “size.txt” file created after each compilation should be checked. 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 analyzed 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.

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

4. Saved variables

4.1. 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.

4.2. EEPROM or FRAM

It is possible to use the saved variables when the PLC has an EEPROM or FRAM instead of a saved RAM. When %S18 switches to 1, a backup is launched for all the variables that have the option “Saved” selected.

With the current µLadder version, instructions for the manual backup should be no longer used. There is indeed a high risk when using 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.

4.3. System words

%SW15 and%SW22 system words are available to the user for the backup of the words to 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: to the SD card and to the EEPROM or FRAM.

5. Watchdog

5.1. Maximum runtime

%SW1 must be set in ms. When %SW0 is higher than %SW1, the system sets %S15 to 1. There is no system blocking. %S15 must be deliberately reestablished to 0. There is no time control in the cycle when %SW1 is 0.

5.2. Soft Watchdog

%SW2 must be set in ms. When %SW0 is higher than %SW2, variables are saved, BIN and ANA outputs are forced to 0, the PLC is switched into STOP mode, %S21 is set to 1 and the value 1 is written to the RESET_SW parameter of the SD card.

If %SW2 is 0, the default value (3000 ms) is used by the system.

6. System functions/Internal functions

6.1. Communication

6.1.1. 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.

6.1.1.1. Slave Modbus

NESC_SERx should be defined just with its slave number and either SERx or SOCKx 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.

6.1.1.2. Master Modbus

SERx or SOCKx 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.

6.1.1.2.1. Return = ModbusRead (parameter1, parameter2, parameter3, parameter4, parameter5, parameter6, 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_ETH0, COM_PORT_ETH1, COM_PORT_ETH2, COM_PORT_ETH3) 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.

6.1.1.2.2. Return = ModbusWrite (parameter1, parameter2, parameter3, parameter4, parameter5, parameter6, parameter7, parameter8)

This function allows to write words or bits into 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_ETH0, COM_PORT_ETH1, COM_PORT_ETH2, COM_PORT_ETH3) 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.

6.1.1.3. 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.

6.1.1.3.1. SockConnect (parameter1, parameter2, parameter3, parameter4, parameter5, parameter6)

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

Parameter1: socket number. Set to 0, 1, 2 or 3. It must be in accordance with the first parameter of ModbusRead and ModbusWrite functions.

Parameter2: Parameter 1 of IP address.

Parameter3: Parameter 2 of IP address.

Parameter4: Parameter 3 of IP address .

Parameter5: Parameter 4 of IP address.

Parameter6 : Client or server port number to which the connection must be established. For example 502 for Modbus.

6.1.1.3.2. SockClose (parameter1)

This function closes the socket.

Parameter1 : socket number. Set to 0, 1, 2 or 3. It must be in accordance with the first parameter of ModbusRead and ModbusWrite functions.

6.1.1.3.3. Return = SockReadStatus (parameter1)

This function reads the socket status.

Return: return variable of type unsigned char. It is TCP_STATUS_CLOSED (value 0) if the socket is closed. It is TCP_STATUS_CONNECT (value 1) if the socket is connecting. It is TCP_STATUS_READY (value 2) if the socket is open and ready to operate.

Parameter1 : socket number. Set to 0, 1, 2 or 3. It must be in accordance with the first parameter of ModbusRead and ModbusWrite functions.

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

6.1.2. Communication without protocol

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

6.1.2.1. Transfer

6.1.2.1.1. ComPush (parameter1, parameter2, parameter3)

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

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_ETH0, COM_PORT_ETH1, COM_PORT_ETH2, COM_PORT_ETH3 is preferable.

Parameter2: string of type unsigned char*.

Parameter3 : unsigned short; number of characters to be sent.

Example 2. Use of ComPush

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

6.1.2.1.2. *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 string to the PLC.

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_ETH0, COM_PORT_ETH1, COM_PORT_ETH2, COM_PORT_ETH3 is preferable.

Parameter2: unsigned char*.

Example 3. Use of ComPushByte

```
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');
```

6.1.2.1.3. ComSend (parameter1)

This function sends a string previously written into a temporary buffer via serial port or Ethernet socket.

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

When transferring via Ethernet port, 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 interrupt. Therefore the RES_SERx system word should be checked or the ComFlushOutput function should be used for knowing 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_ETH0, COM_PORT_ETH1, COM_PORT_ETH2, COM_PORT_ETH3 is preferable.

Example 4. Use of ComPush and ComSend

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

Example 5. 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);
```

6.1.2.1.4. 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_ETH0, CO_PORT_ETH1, COM_PORT_ETH2, COM_PORT_ETH3 is preferable.

Example 6. Use of ComFlushOutput

```
ComFlushOutput (COM_PORT_SER1);
```

6.1.2.2. Reception

6.1.2.2.1. 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_ETH0, COM_PORT_ETH1, COM_PORT_ETH2, COM_PORT_ETH3 is preferable.

Example 7. Use of ComGetFrameLength

```
Long = ComGetFrameLength (COM_PORT_SER1);
```

6.1.2.2.2. 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_ETH0, COM_PORT_ETH1, COM_PORT_ETH2, COM_PORT_ETH3 is preferable.

Example 8. Use of ComGetFrame

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

6.1.2.2.3. 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_ETH0, COM_PORT_ETH1, COM_PORT_ETH2, COM_PORT_ETH3 is preferable.

Example 9. Use of ComFlushInput

```
ComFlushInput(COM_PORT_SER1);
```

6.2. SD card management

The SD card must be formatted in FAT32.

Operations on the SD card 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 SD card create ResetWatchdogs aiming at avoiding a watchdog overrun.

The SD card 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.

6.2.1. Structure of file names

The type of storage medium (SD for a SD card or USB for a memory stick), the file name (8 characters maximum) and the extension (3 characters maximum) must be indicated.

Example 10. Structure of file names

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

6.2.2. 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 11. Use of FSOpen

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

6.2.3. 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 12. Use of FSClose

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

6.2.4. Return = FSSeek (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 13. Use of FSSeek

```
FSFile fd;
unsigned char result;
result = FSSeek (&fd, 10);
```

6.2.5. 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 14. Use of FSDelete

```
unsigned char result;
```

```
result = FSDelete ("SD:LOG.TMP");
```

6.2.6. Return = FSCopy (parameter1, parameter2)

This function allows to copy 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 source filename.

Parameter2: string containing the destination filename.

Example 15. Use of FSCopy

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

6.2.7. Return = FSMove (parameter1, parameter2)

This function allows to rename 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 source filename.

Parameter2: string containing the destination filename.

Example 16. Use of FSMove

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

6.2.8. Return = FSRead (parameter1, parameter2, parameter3)

This function allows to read a limited number of characters. 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 characters to be read.

Example 17. Use of FSRead

```
FSFile fd;
```

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

6.2.9. 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 18. Use of FSWrite

```
FSFile fd;
char *string = "Good morning";
unsigned char result;
result = FSWrite (&fd, string, 12);
```

6.2.10. 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 19. Use of FSReadLine

```
FSFile fd;
char string[20];
unsigned char return;
unsigned char result;
result = FSReadLine (&fd, string, 20, &return);
```

6.2.11. Return = FSReadCSVRow (parameter1, parameter2, parameter3, 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 20. Use of FSReadCSVRow

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

6.2.12. 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 21. Use of FSWriteCSVRow

```
FSFile fd;
char *string = "Col1-Col2";
unsigned char result;
string[4] = 0;
result = FSWriteCSVRow (&fd, 2, string);
```

6.3. LCD screen

Some PLCs, µArmA1, A5 and A7, have a 2-row LCD screen (16 characters per row) with back-lighting.

Characters to be displayed are stored in 2 memory areas. Standard characters are stored in the DDRAM. The CGRAM can store 8 characters defined by the user. The 'µ' character is defined as first character of this memory area.

Functions presented below are immediately executed, that means with no delay.

6.3.1. WriteText(parameter1, parameter2, parameter3, parameter4)

This function is included in the standard library of µLadder. It enables to display a text on the LCD screen. It is possible to configure the text on the first line and on the second line and to align both texts on the screen independently.

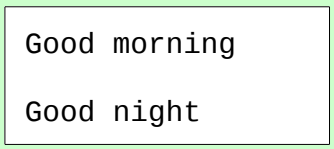
Parameter1: text to be displayed in the first line of type char*.

Parameter2: text to be displayed in the second line of type char*.

Parameter3: alignment of the first line on the screen (0: left, 1:centered, 2: right)

Parameter4: alignment of the second line on the screen (0: left, 1:centered, 2: right)

Example 22. Use of WriteText




```
/*A string is displayed at each line, aligned
to the left*/
WriteText("Good morning","Good night",0,0);
```

6.3.2. AffClear()

This function deletes the screen content. The execution of this function takes around 10ms. It is faster to display two blank rows instead.

6.3.3. AffSend(parameter1, parameter2)

This function allows to send a command or data to the screen. It is used for the configuration and definition of specific characters. This is a very low-level function, only used in specific cases.

Parameter1: Type of information. KB_AFF_TYPE_DATA (value 1) should be used for data sending and KB_AFF_TYPE_INSTR (value 0) should be used for command sending.

Parameter2: Data or instruction.

Example 23. Use of AffSend

```
c
```

```
/*Show character 'c'*/
AffSend(KB_AFF_TYPE_DATA, 'c');
```

6.4. Log management

It is possible to manage the log on those PLCs that include a saved RAM connected to the address bus of the processor (µArmA2 and µArmA8). It is possible to retrieve events (record of all the appearance and disappearance dates of errors, record of all the appearance dates of messages), alarms (retrieval of active errors and appearance date) and tracks (with dates and values of a piece of data).

These data are stored to the memory. It is possible then to read them and save them to a SD card or memory stick. These data can be also retrieved on an external system by using a specific Modbus protocol.

6.4.1. Parameter setting file

This file must be named “var.csv” and located on the SD card.

This file contains the variable list and logging parameters. It is read at the switching on, after the application loading or when forcing %S19 to 1. At this moment, the database is reset to zero.

Error texts and messages can only be changed during the reading of this file.

- Column 1: index, it is the number of the logged variable, from 1 to 65535.
- Column 2: complete variable address in the equipment, used for variable logging in other PLCs (For IOBus coomunication), such as %MW10.1 (this allows to read %MW10 in the equipment and automatically re-copy the value in this variable, address defined by column 3). Similar for writing.
- Column 3: address of variable such as %MW50 or %MW51: x0.
- Column 4: minimum value of the variable before adjusting to scale.
- Column 5: maximum value of the variable before adjusting to scale
- Column 6: minimum value of the variable after adjusting to scale

- Column 7: maximum value of the variable after adjusting to scale
- Column 8: 1 when variable is logged as error, otherwise 0.
- Column 9: string of operation for inducing logging as error.
- Column 10: comparison value for inducing logging as error.
- Column 11: string for the appearance of an error
- Column 12: string for the disappearance of error
- Column 13: 1 when variable is logged as message, otherwise 0.
- Column 14: string of operation for inducing logging as message.
- Column 15: comparison value for inducing logging as message.
- Column 16: string for the appearance of a message.
- Column 17: 1 if the variable is logged as a curve, 0 otherwise
- Column 18: 1 if the value must be averaged, 0 otherwise.
- Column 19: cycle period or averaging period before possible recording.
- Column 20: value difference (hysteresis) causing the storage of a new piece of data.

Note:

- A curve piece of data will only be logged if both hysteresis and time exceed. In the case of a non-averaged value, at the end of the period, if the variation delta exceeds the hysteresis, this value will be stored. In the case of an averaged value, an average is performed during the period and, at the end of the period, if the delta variation exceeds the hysteresis, this value is stored.

```
113;;%MW211:x7;;;;;1;>;0;"Lower threshold error";"End of the lower
threshold error";;;;;
114;;%MW211:x8;;;;;1;>;0;"Higher threshold error";";"End of higher
threshold error";;;;;
151;;%MW212:X1;;;;;;;;;;1;>;0;"Push alarm test button";;;;
417;;%MF14;;;;;0;;;;;0;;;;1;;3;0.2
418;;%MF16;;;;;0;;;;;0;;;;1;;3;0.2
```

6.4.2. 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.

6.4.2.1. "Date" structure

The following elements are contained in this structure :

Table 3. "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

6.4.2.2. 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

Parameter1: variable of structure “Date”

Example 24. Use of dateToTime

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

6.4.2.3. 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”

Parameter 1: time stamp, variable of type long

Example 25. Use of timeToDate

```
long tsDate;
Date d;
d = timeToDate(tsDate);
hour = d.hour;
```

6.4.3. LogValue Format

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

Table 4. LogValue format elements.

unsigned long time;	Time stamp of the data logging.
unsigned short varIndex;	Number of the logged variable (index of "var.csv" file)
char *string;	string of the event or alarm, This string is NULL if it is inexistent.
double value;	Value of the logged variable. For the retrieval of an event, the value of this field is the value of the variable.
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.

Note:

- The value of the variable when retrieving a digital value is registered in the “value” field. When retrieving alarms, the value is always 0. 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.

- If there is no accurate information in the “var.csv” file regarding the threshold that causes a message to be logged (column 14 and 15), logging is performed at every change in the value.

6.4.4. 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.

Parameter 2: 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 26. Use of LogAlQuery

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

6.4.5. Return = LogEvQuery (parameter1, parameter2, parameter3)

This function opens the database for retrieving the log of all errors appeared and disappeared and all messages appeared. LogFetch must be then executed within the same cycle for the data retrieval. The difference can be made in the “type” field of the variable of type LogValue.

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.

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

Parameter3: 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 27. Use of LogEvQuery

```
long tsStart;
long tsEnd;
unsigned short nbrEv;
nbrEv = LogEvQuery (tsStart, tsEnd, LOG_QUERY_ORDER_DESC);
```

6.4.6. Return = LogTrQuery (parameter1, parameter2, parameter3, 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 28. Use of LogTrQuery

```
long tsStart;
long tsEnd;
unsigned short nbrTr;
nbrTr = LogTrQuery (400, tsStart, tsEnd, LOG_QUERY_ORDER_DESC);
```

6.4.7. Return = LogFetch (parameter)

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 29. Use of LogFetch

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

6.4.8. 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 30. Use of LogAlSave

```
long tsStart;
long tsEnd;
unsigned char ret;
ret = LogAlSave ("SD:Alarm.csv", tsStart, tsEnd);
```

6.4.9. 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 31. Use of LogEvSave

```
long tsStart;
long tsEnd;
unsigned char ret;
ret = LogEvSave ("SD:Event.csv", tsStart, tsEnd);
```

6.4.10. Return = LogTrSave (parameter1, parameter2, parameter3, parameter4)

This function allows to backup values 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: 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 32. Use of LogTrSave

```
long tsStart;
long tsEnd;
unsigned char ret;
ret = LogTrSave ("SD:Value.csv", 400, tsStart, tsEnd);
```

6.4.11. LogPurge ()

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

Example 33. Use of LogPurge

```
LogPurge ();
```

6.4.12. LogEvPurge ()

This function deletes all the events .

Example 34. Use of LogEvPurge

```
LogEvPurge ();
```

6.4.13. Return = LogTrPurge (parameter1)

This function deletes the complete log of a variable.

Return: variable of type unsigned char. It is1 when the variable exists, 0 otherwise.

Parameter1: number of the logged variable.

Example 35. Use of LogTrPurge

```
unsigned char ret;
ret = LogTrPurge (400);
```

III. PROGRAMMING IN µLADDER

1. Getting started

1.1. Start and quit µLadder

After installation of the µLadder software, the access to the program can be done from the Start Menu on Windows operating systems of through the Dash on some Linux operating systems, just searching for µLadder in the Search Box. No shortcuts are created on the Desktop by default. Once the program is open, the µLadder main window (see section 1.2) pops up.

For leaving the program just go to the menu bar (see section 1.2.2) and do File>Quit. You will be asked to save changes or to discard them before closing µLadder. Temporary files are not saved in µLadder so make sure of saving them manually. You can also leave µLadder by clicking on the Close Button located in the upper righthand part of the main window on Windows operating systems or upper lefthand part of the main window on Linux operating systems.

1.2. Using the GUI

When opening the µLadder, 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 µLadder installed. In the example of Figure 1, you can see " µLadder v8.1". That means that version 8.1 of µLadder software has been installed.
- **Tool bar:** the toolbar includes the following items:
  - ➔ Page selection.
  - ➔ Add button.
  - ➔ Remove 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. For further information see section 1.2.3.

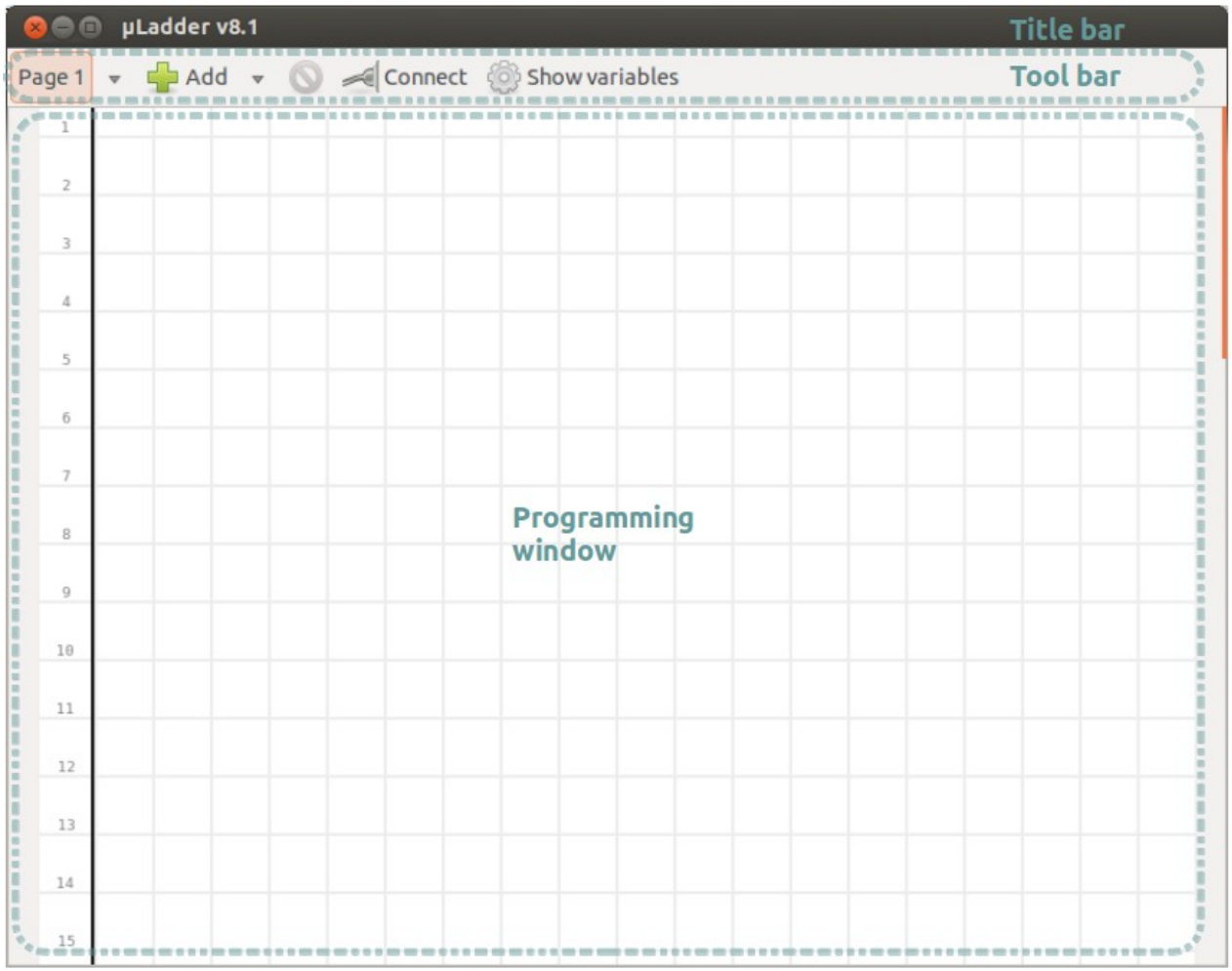


Figure 1. µLadder main window.

1.2.1. Tool bar

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

Figure 2. Toolbar of µLadder

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 a item selection in a ladder page. 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 from the Ladder menu located on the menu bar.
- **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 Slave number, Channel and µDriver's address in order to connect to your PLC. For disconnecting, just click on this option while the connection with the PLC is active. For further information regarding communication with the PLC, see section 3.2. You can also access the *Connection to µDriver* Window from the Communication menu located on the menu bar.
- **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.

1.2.2. Menu bar

The µLadder 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 3. µLadder menu bar.

The following drop menus are included in the menu bar:

- **File (Alt+F).** Common functions are included in the file menu: New, Open, Save, Save As, Quit.
  - ➔ **New (Ctrl+N)**. Create a new project. If you are 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 µLadder. 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.

  ➔ **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.

  ➔ **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.

  ➔ **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.

  ➔ **Firmware settings.** Click on this option to change the firmware settings into AUTO-STOP (by default), HTTP, LOG or SNMP.

  ➔ **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 for developping 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.

  ➔ **Import GUI.** Click on this option if you have created a GUI with µIHM that you want to include in your current project.

  ➔ **GUI settings.** Click on this option to change GUI settings. For further information see section 10.

  ➔ **Export GUI.** Click on this option to export a GUI previously imported.

  ➔ **Remove GUI.** Click on this option to remove a GUI previously imported.

- **Library.**

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

- **Pages.**
  - ➔ **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 accesible from this menu.
- **Ladder**

  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 dissapear 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 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.

➔ **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.

- **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 Slave number, Channel and µDriver's address in order to connect to your PLC. For further information regarding communication with the PLC, see section 3.2. You can also access the *Connection to µDriver* 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.

1.2.3. 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)

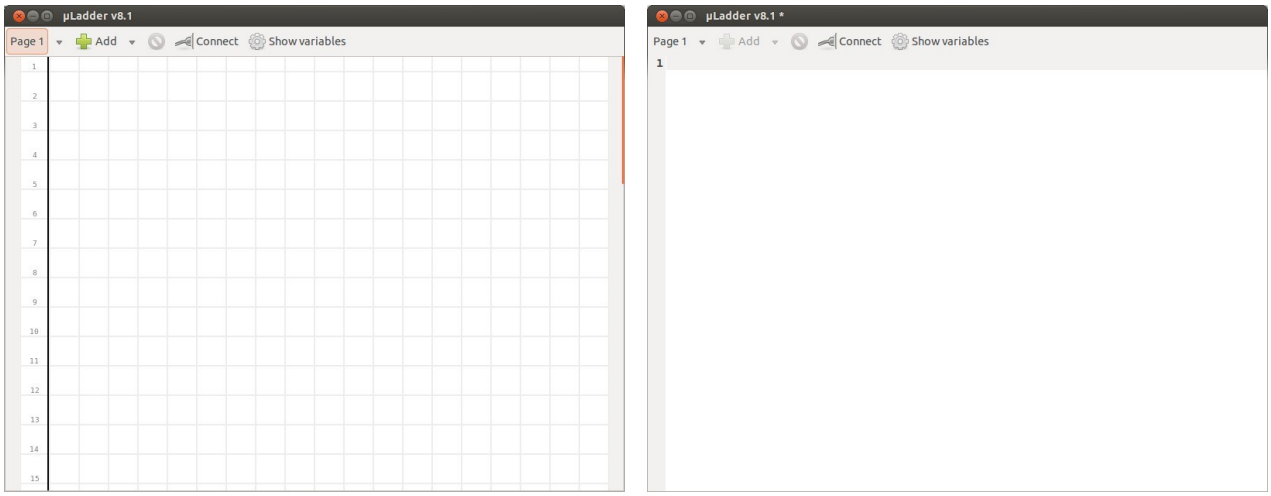


Figure 4. µLadder programming windows: ladder (left) and C (right).

For further information regarding ladder and C programming languages, see Section 1.3.

1.3. The programming languages

1.3.1. 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 hardwired 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.

Fur further information regarding items available in ladder logic for PLC programming, see

section 6.

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

2. Importing firmware

Every time you create a new project, the firmware has to be imported to the application. The file to be imported is the "mArm.sys". Be aware that version 8 of µLadder needs at least version 4.0 of firmware. After having imported the firmware you will be able to set the program type but not before (only bloc function is available).

In order to import firmware follow the instructions described in section 1.2.2

3. Connecting with the PLC

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

3.1. µdriver

µdriver 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 µdriver is not installed, it would be impossible to establish connection with the PLC.

3.2. Establishing connection

To establish connection from µLadder with the PLC, you have to click on the Connect option (see

section 1.2.) and establish connection through mDriver, as said before. Once selected this option, a connection window pops up (see Figure 5).

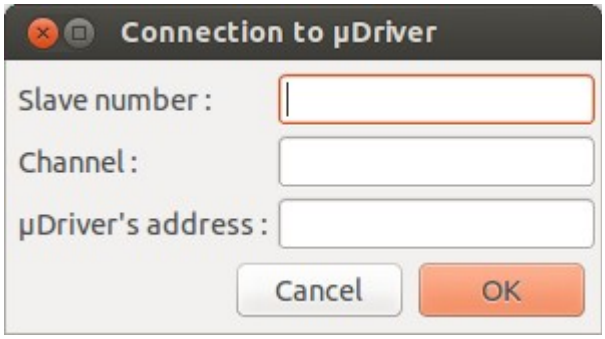


Figure 5. Connection to mDriver window.

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

- **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.
- **Channel:** address of the slave PLC. It depends on the communication protocol: serial (port number) or Ethernet (IP address).
- **mDriver's address:** mDriver 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 mDriver may differ depending of the way you are connecting with the PLC from your computer. Most typical configuration parameters can be seen in Table 5.

Table 5. Typical configuration parameters for connecting to µDriver.

Type of connection	Configuration	
PC – Ethernet - PLC	Slave number	
	Channel	IP address specified in the main.cfg. Typically: 192.168.0.123
	µdriver's address	Nothing
PC – Serial - PLC	Slave number	When having more than one PLC connected in serial to the same RS2485 port, the slave number must be specified"
	Channel	Serial port number (1,2,3,4,...)
	µdriver's address	Nothing
PC – PicoFox – Ethernet/Serial-PLC	Slave number	See above
	Channel	See above
	µdriver's address	Address of the system communicating directly with the PLC. In the case of using a PicoFox, the typical IP address is: 192.168.0.101. Make sure of checking this IP address.

Once correctly established the connection parameters, 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)

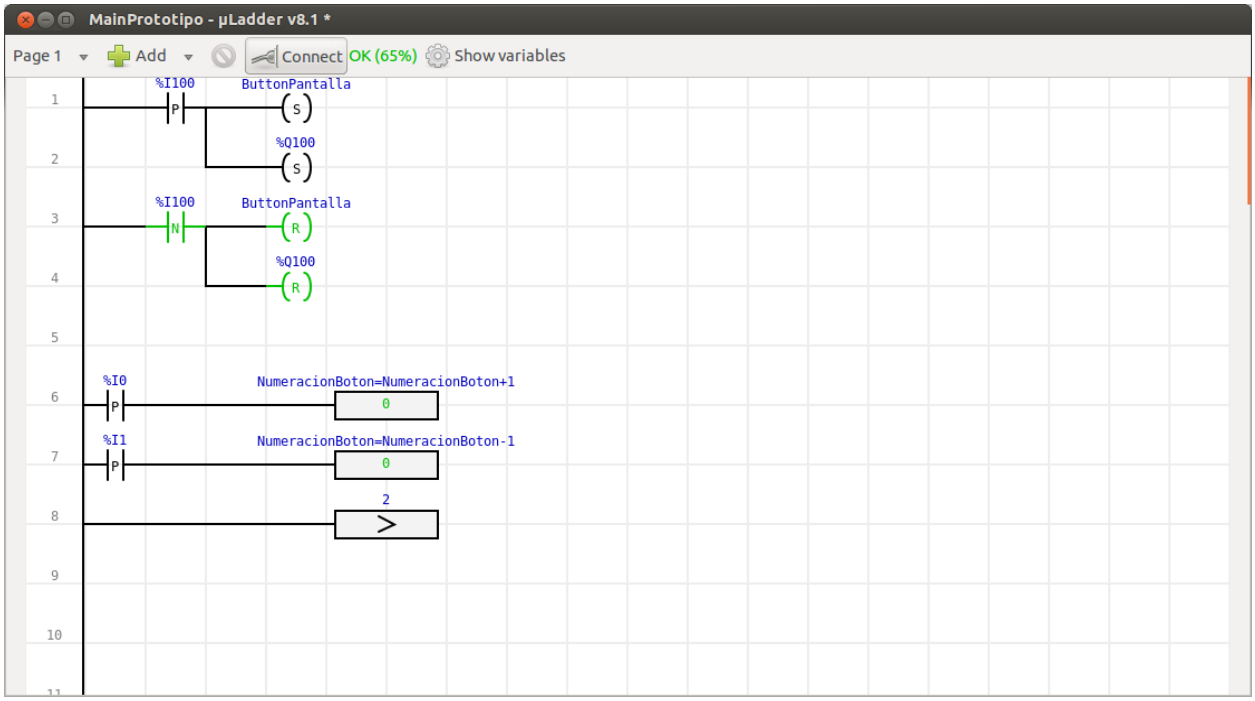


Figure 6. Screen capture of the µLadder main window with the PLC connected and communicating.

- The Connect option of the Communication drop-down menu of the menu bar is unabled and the Disconnect option enabled.
- A green message “OK” has appeared on the tool bar. Active instructions of the program appear painted in green as well (See Figure 6)

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.

4. Using pages

The number of pages able to be used in any µLadder program is unlimited, being each ladder page limited to 50 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 the main page of the program. It is called at each PLC cycle. Pages not called from Page 1 will not be interpreted or compiled. Than means that, if a user creates a program using pages from Page 2 on, but leaving Page 1 empty, the compilation will result into an empty program.

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 (see section 4.2.) 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.

For further information related to the creation and use of pages, see next sections.

4.1. Creating a page

A new µLadder 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. (See section 1.2)

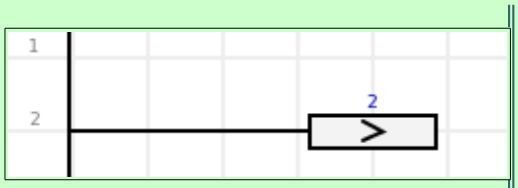
The number of pages to be used in a µLadder project is unlimited, and pages can interact between them by using the *call* command.(See section 4.2) This command is a object in ladder code and a function in C code.

4.2. Call of a page

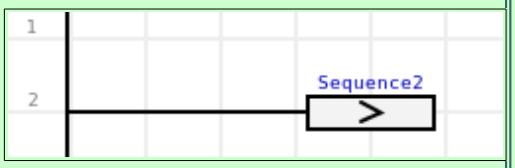
A page can be called from another page inside a program, no matter the type of code used in each of the pages. That means a ladder page can be called from a C page and vice-versa.

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.

Example 36. 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.

Example 37. Inserting a Call command with C.

```
page_2() ;
init() ;    For calling a page named “init”.
```

5. Variable editor

Variables in µLadder, 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 µLadder program can be shown and edited from the variable editor (see Figure 7). You can access the variable editor by clicking on Show variables (see section 1.2).

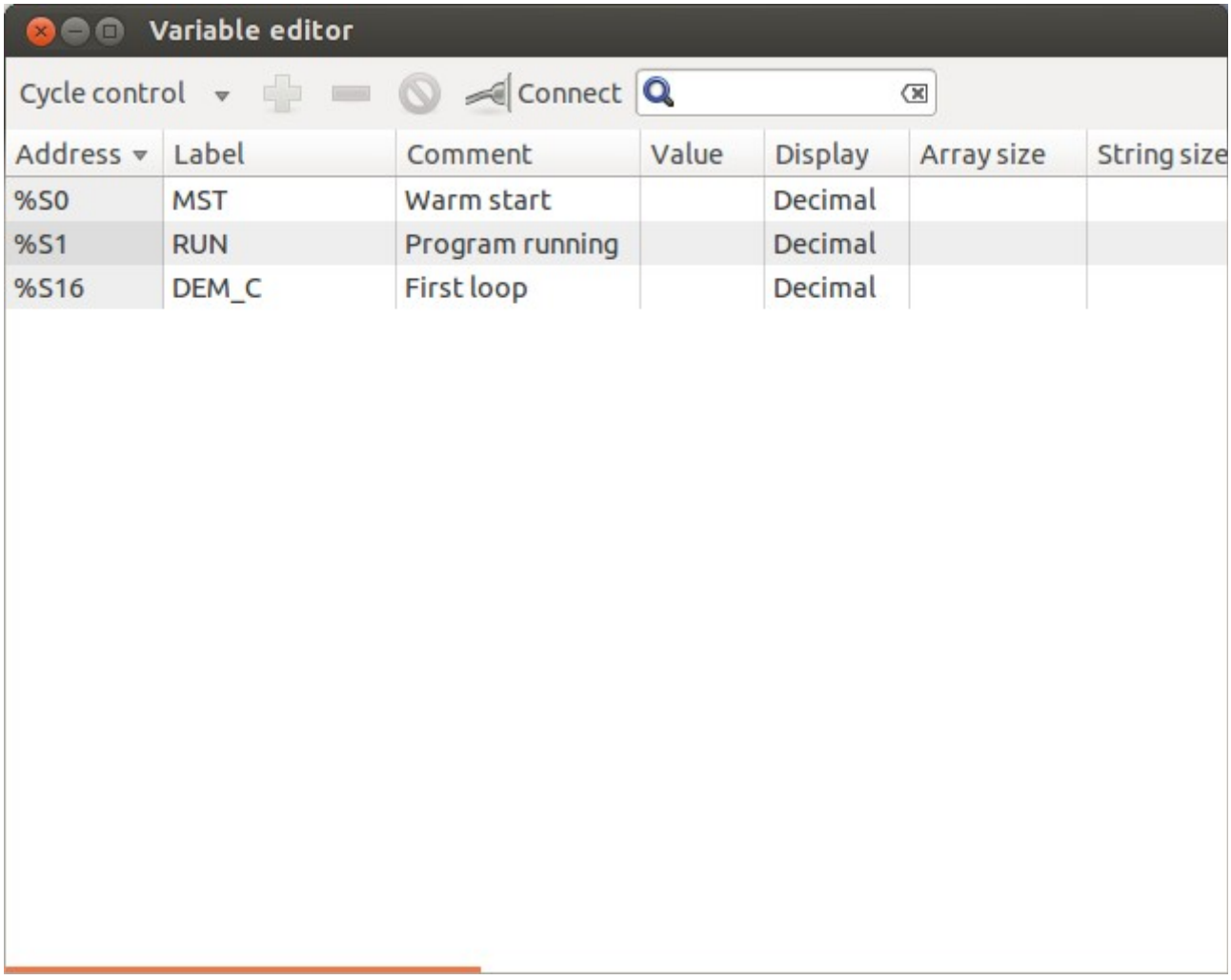


Figure 7. Variable editor showing the cycle control system variables.

5.1. 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 8.

Figure 8. Variable editor tool bar.

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

- **User variables/System variables.** Select the type of variables you want to be shown from the drop-down list: system variables or user variables.
- **Add.** Click on this option to include a new variable to the current µLadder project. Give this new variable a name (either an address or a mnemonic) and define its properties in the new properties window that pops-up. See section 5.3 for further information on variable properties and section 5.4.3 for creating variables.
- **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 Slave number, Channel and µDriver's address in order to connect to your PLC. For disconnecting, just click on this option while the connection with the PLC is active. For further information regarding communication with the PLC, see section 3.2. You can also access the *Connection to µDriver* 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 mnemonic. 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).

5.2. Variable editor tool bar

The Variable Editor has its own menu bar with a series of drop-down menus included. (See Figure 9)

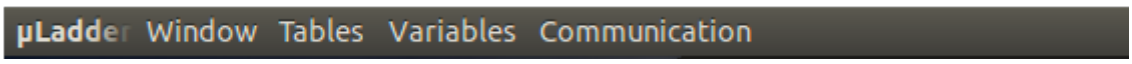


Figure 9. Variable editor menu.

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 µLadder, a category or group of variables is called table. By default µLadder 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 µLadder project.

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

  ➔ **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.

- **Variables**

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

  ➔ **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.

  ➔ **Find.** This option allows to track the use of the variable inside the program. Pages, lines and instructions where the selected variable is used is shown.

  ➔ **Properties.** Click on this option to edit properties of the selected variable.

➔ **Remove.** Click on this option to 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 delete them. Variables could not be deleted when being used inside the program. You can also delete them from the tool bar.

- **Communication (Alt+C).** This drop-down menu is similar to the Communication drop-down menu of the µLadder 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 µDriver's address in order to connect to your PLC. For further information regarding communication with the PLC, see section 3.2. You can also access the *Connection to µDriver* 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.

5.3. Variable properties

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

- **Address:** is the memory address of the variable in the PLC. For better understanding prefixes used for the address of variables, see section 3 of SOFTWARE ENVIRONMENT chapter.
- **Label:** is the mnemonic given to the variable.
- **Comment:** short explanation of the variable.
- **Value:** current value of the variable.
- **Display:** type of numeral system used for the display of the variable value. It can either be decimal (the default option), binary or hexadecimal.
- **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.
- **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.
- **Global:** this property can be used with functions. It allows the variable keeping its value between two calls.
- **Timer (in ms):** time period for the variable to be decreased from 1 to 0
- **Configuration:** parameter setting for the analog inputs (see section on *ANALOG inputs*)
- **Parameter:** used for the declaration of function variables. See section 8 on *Creation of a function* for further information. “Internal” is used for a main application variable.
- **Min. display value:** minimum value to be displayed of a variable for synoptics, gauges, etc. in µIHM.
- **Max. display value:** maximum value to be displayed of a variable for synoptics, gauges, etc. in µIHM.
- **Float precision:** when having float variables, the precision for the value display for µIHM can be determined.

5.4. Type of variables

Variables considered in a µLadder program can be divided into two main tables: system variables and user variables. They are described next.

5.4.1. System variables

System variables 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:

- **Cycle control:** %S0, %S1, %S16
- **Date&Time:** %S5-%S8, %S24, %SW5-%SW11
- **Ethernet:** %SW100-%SW137
- **I/O settings:** %S11, %S12, %SW26-%SW32
- **Interrupt:** %SW25
- **Serial ports:** %SW34-%SW81

- **System infos:** %SW13, %SW14
- **Variable setting:** %S2, %S18, %S19, %SW15-%SW22
- **Watchdogs:** %S15, %S20-%S22, %SW0-%SW2, %SW4

For further information about system variables, see section see section 3 of SOFTWARE ENVIRONMENT chapter.

5.4.2. User variables

The user may need additional variables for a program created in µLadder. 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 µLadder software and thus they will be shown in the Variable Editor. However, a manual entry is needed for the declaration of arrays.

Please see next section on creation of variables to see how to manually create variables from the Variable Editor.

5.4.3. Creating variables

Variables already entered in the program are automatically created by the µLadder 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 5.3) on the new window that pops-up (see Figure 10).

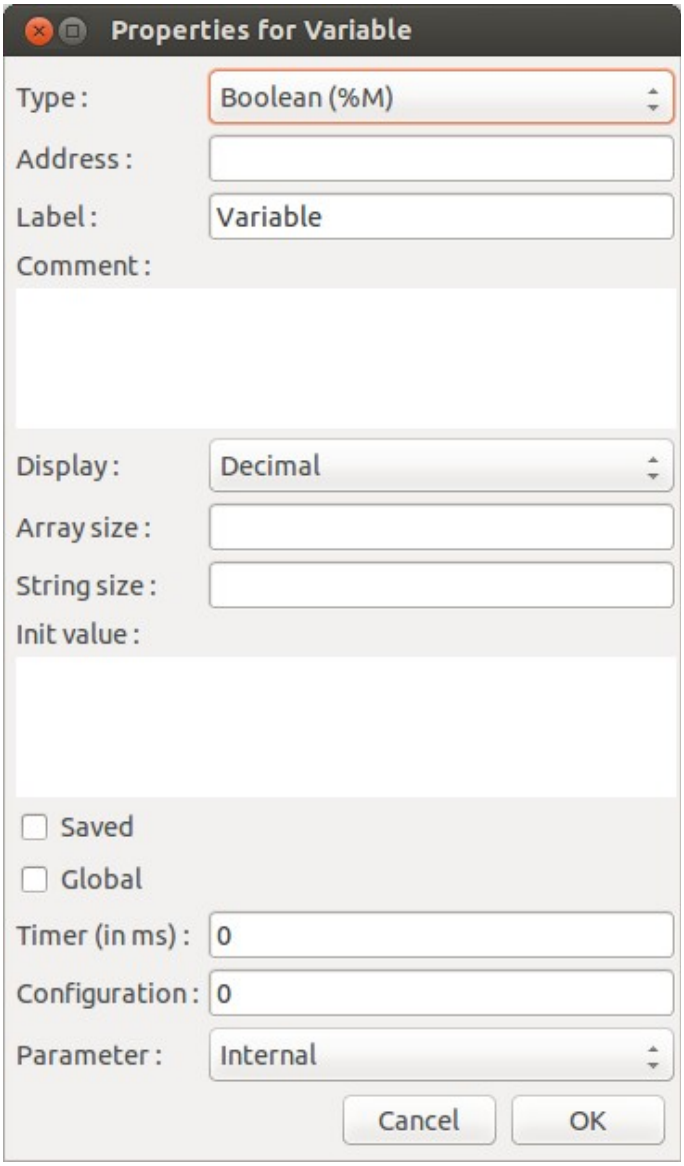


Figure 10. Variable properties window.

Properties “Init value”, “Saved”, “Global”,“Configuration” and “String size” are considered also for arrays. But if the variable is declared alone, variable properties take priority over array properties. The use of %MW10[1] is preferable over %MW11, as a variable is declared and its properties will be prioritized over array's properties.

The definition of an address is not required. The compilation will locate the variable at a free space within the variable area of the same type.

5.4.4. Using variables in a program

Variables can be referred to in µLadder program both by calling them through their address or mnemonic. However, if they have a mnemonic, 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.

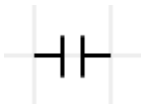
To export variables, go to Tables>Export table variables. A dialogue box allows to specify the structure of the CSV file during the execution of these processes. It is preferable not to change the default structure for avoid possible import problems. If you change it when exporting, make sure that you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

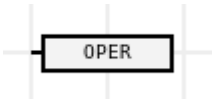
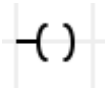
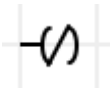
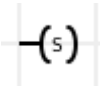
Variable import is performed by overwriting current variables, so make sure you do not have in your current project a variable with the same address or mnemonic, as it will be directly replaced.

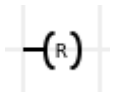
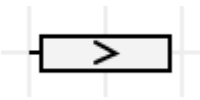
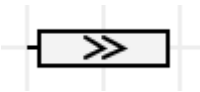
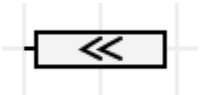
6. Objects available in Ladder

Basic elements or items available in µLadder 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	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.
 Closed contact	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.

Item: symbol and name	Description
 Rising edge	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.
 Falling edge	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.
FUNCTION	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.
 Operation	Operations between variables can be defined with the use of the Operation instruction.
 On	It is activated when the combination on the left results into 1. Its activation means that it has a value logic 1.
 Off	It is activated when the combination on the left results into 0. Its activation means that it has a value logic 0.
 Set	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.

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.

For connecting items with each other, just double click in the background line of the ladder grid there 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.

6.1. Open contact

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.

It can be used either with a binary variable, or as a comparison of the type %MW0>0 with integer, long or float variables. That means every value of the variable different to 0 will activate the contact.

6.2. Closed contact

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.

Same remarks as those already done for open contact can be done.

6.3. Rising edge

A rising edge is activated when there is a change in the value of the variable represented (input, internal variable or system bit) from 0 to 1.

Each edge object is managed by an internal variable independent from the variable internally used:

- 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.
- Possible bug with indexed bits, if the index has varied from one cycle to the next cycle.
- 1 byte is used by the edge object
- Complex expressions and word bits can be managed.

6.4. Falling edge

A rising edge is activated when there is a change in the value of the variable represented (input, internal variable or system bit) from 1 to 0. Same remarks as those already done for rising edge can be done.

7. Creation of a program

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

7.1. First program in ladder

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

- PLC mArm 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 IHM input.

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 mArm 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 mArm 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. (See section I.Error: Reference source not found) Figure 11 shows this program already implemented in µLadder.

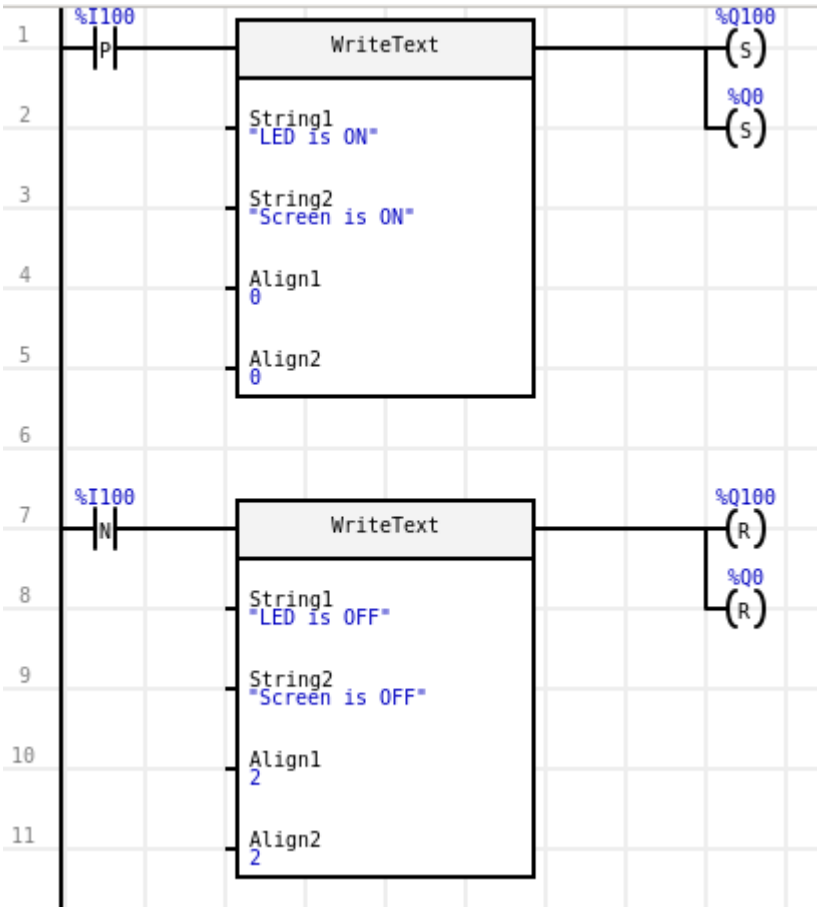


Figure 11. First program created with ladder.

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

7.2. First program in C

The similar exercise as the one explained in section 7.1 is being performed in the current section, so the same context has to be analyzed.

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);
    }
```

7.3. 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 in sections 7.1 and 7.2 is being divided into two pages as follows:

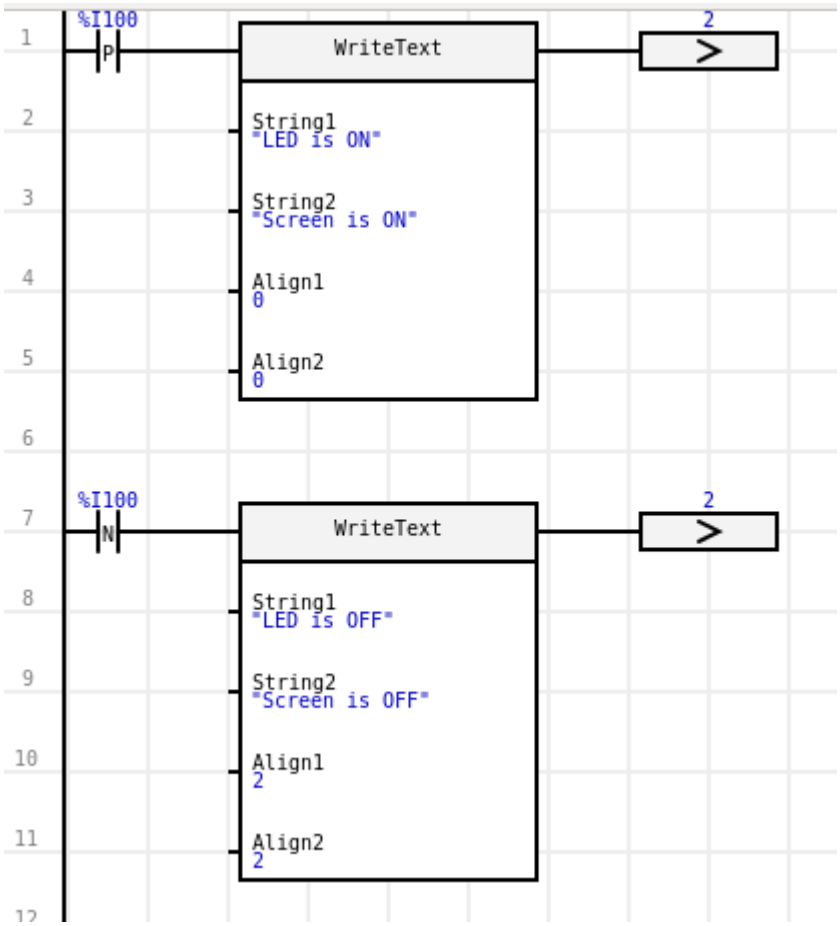
- A first page includes the program that shows the text on the screen when pushing the push button.
- A second page includes the program that activates the LED and the back-light of the LED screen.

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 µLadder combining ladder and C code.

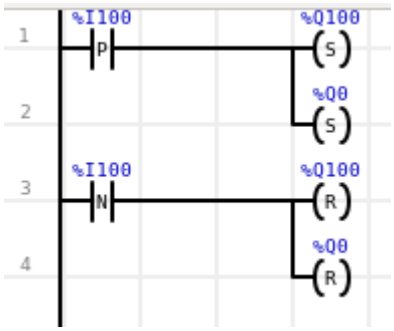
Page 1	

Page 2	`if (%I100==1)`
	`{`
	`%Q100=1;`
	`%Q0=1;`
	`}`
	`else`
	`{`
	`%Q100=0;`
	`%Q0=0;`
	`}`

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 µLadder combining C and ladder code.

Page 1	`if (%I100 == 1)`
	`{`
	`WriteText("LED is ON","Screen is ON", 0,0);`
	`page_2();`
	`}`
	`else`
	`{`
	`WriteText("LED is OFF","Screen is OFF", 0,0);`
	`page_2();`
	`}`

Page 2	

7.4. 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. (See section 2.4 on the SOFTWARE ENVIRONMENT chapter)
- 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 mControl software. (See section 2.4 on the SOFTWARE ENVIRONMENT chapter)

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 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 2.2, 2.4, and 3.13.

8. Creation of a function

A function is an application which can be imported inside another application. This can be useful when managing identical subparts 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.

8.1. Defining variables

As said in section 5.4, 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. For further information regarding variable properties see section 5.3.

8.2. Use of a function

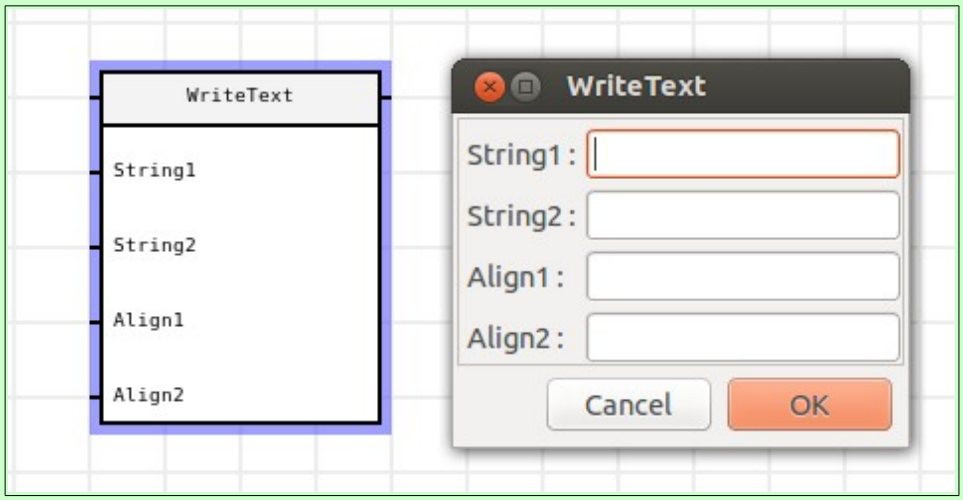
The function must be imported inside the application by launching the “Library/Import function“ command. A single import is enough, even if the function is used many times.

8.2.1. 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.

Example 38. WriteText function.



8.2.2. 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.

Example 39. WriteText function.

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

9. Using timer

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

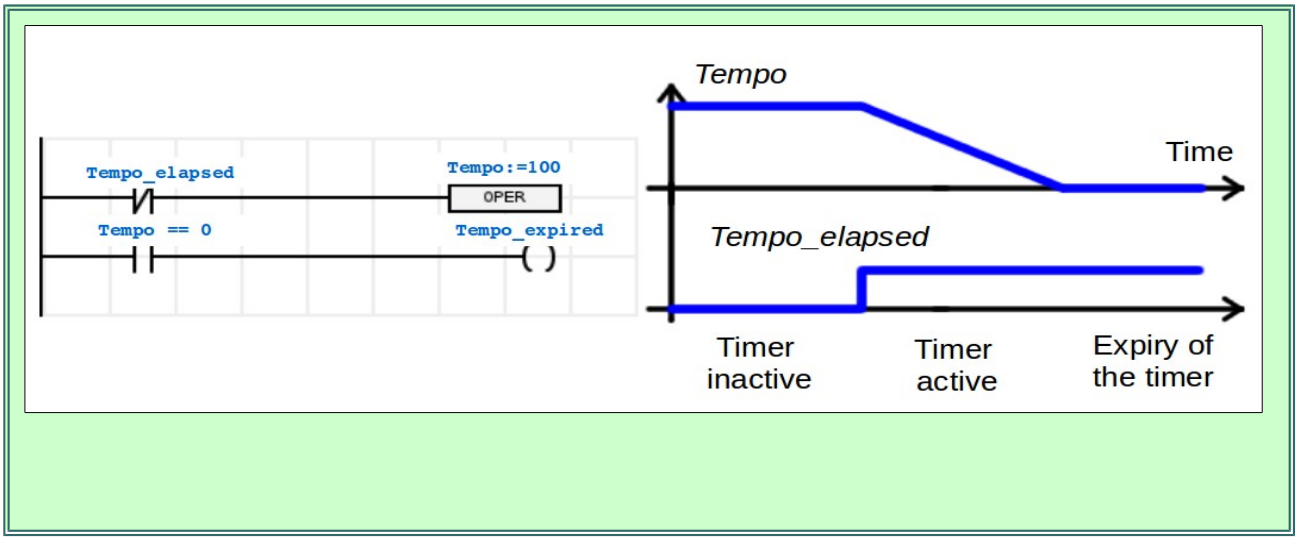
- The timer is inactive when its value is zero or lower.
- The timer is active while its value is higher than zero and lower than its countdown period value. In this case it decreases with the countdown period indicated in the variable.

Example 40.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.

9.1. 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.

9.2. Examples of the use of timer

For better understand how to use a timer in a program, the following example is being done. Considering again the PLC µArm A1, the program that is to be created 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 12.

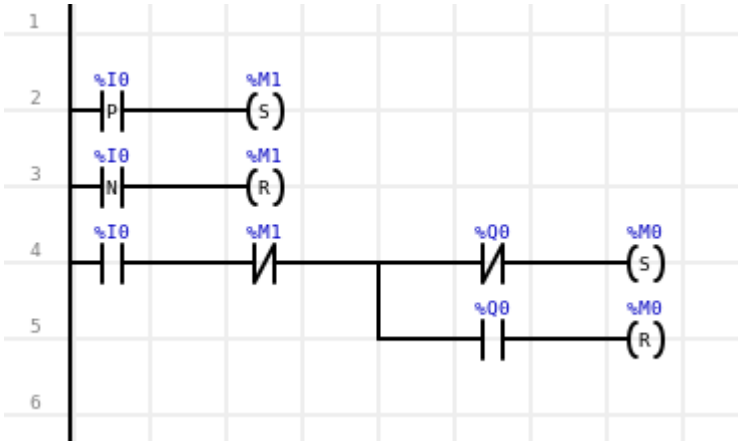


Figure 12. Example with the use of a timer in µLadder.

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 mili-seconds, so 3000ms should be written).

10. Using a GUI

µArm A2 PLCs include a touchscreen, whose graphic user interface can be easily designed with the µIHM software. This software is complementary to µLadder, and has been conceived for the development of Human–Computer Interaction interfaces, HCI, (Interface Homme Machine, IHM, in French).

Adding a GUI to a µLadder program provides an extra degree of flexibility in the PLC programming with µLadder, 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.

10.1. µIHM

µIHM is the specific software created for the easy design of graphical HCI. 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 µIHM, please read the µIHM user guide.

10.2. How to insert a GUI

As previously introduced, the management of GUIs can be only considered when programming a

µArm A2. When programming this type of PLCs, it is possible to additionally include a GUI into a µLadder application when programming. You can do it by selecting Program>Import GUI from the menu bar, and selecting a .vu file already created with µIHM.

10.3. How to configure a GUI

Once a GUI has been imported into a µLadder program, both files, the µIHM .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 µLadder and assigned in both files.

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

10.4. Example

For better understand how to use a GUI in a program, the following example is being done. Considering again the PLC µArm A2, the program that is to be created has the following configuration:

- First of all, a simple GUI is being created with µIHM 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 180º, 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 µLadder 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 µIHM 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 *<=%SW5>*.
- **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 <=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 13.

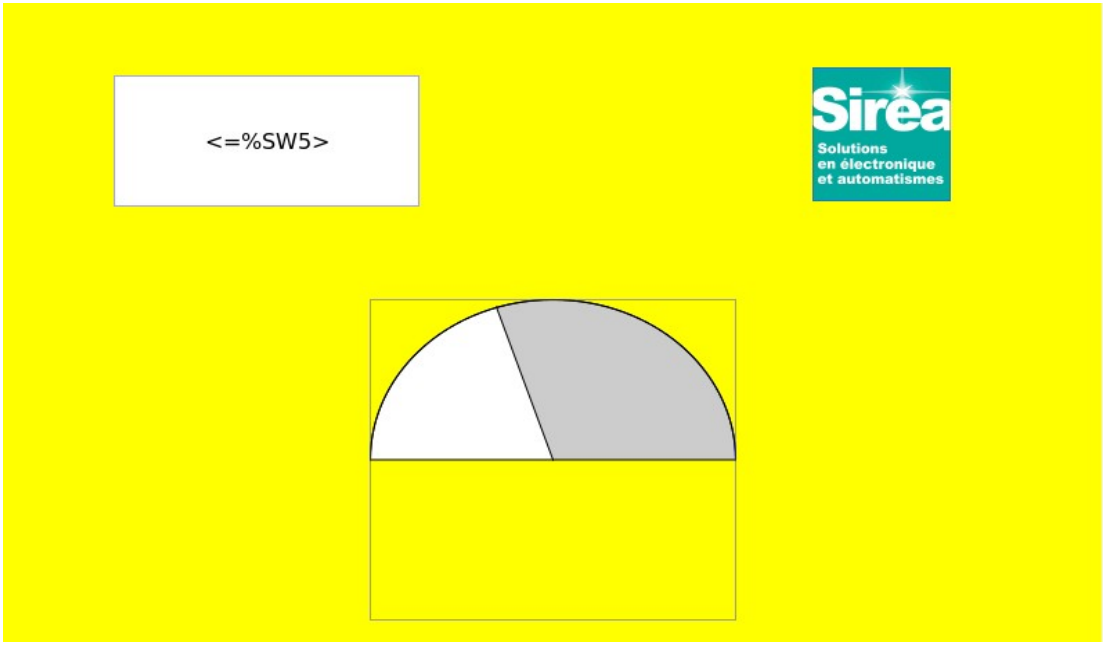


Figure 13. Example of the appearance of the interface created.

The next step is to create the program in µLadder and import the GUI just created. For this purpose, create a new project in µLadder, do Program>Import GUI 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 µLadder. As %SW5 is a system word, there is no need to define it.

IV. ANNEXES

1. Annex A. Revision history

Date	Modification
01/07/14	First version of the µLadder User Guide.
04/22/14	Second version of the µLadder User Guide. New structure of the document into four chapters and introduction of guidelines for programming in µLadder.

2. Annex B. Software version history

Date	Modification
02/23/12	Original version
04/26/12	Update of errors, messages and log of values
06/08/12	LogPurge, LogEvPurge, LogTrPurge functions included
06/29/12	Modification of %S13
06/04/12	Watchdog error correction (%S21 instead of %S20)
07/16/12	%SW11 included
07/27/12	Red fix mode on status LED included PWM output management included
08/03/12	Note for the compiler installation included
09/20/12	Parameter 8 for ModbusWrite function included
10/26/12	Modification of “LogValue” structure for events
12/06/12	Additional information for the use of functions included
03/01/13	New compiler installation update New section for the management of edges included Error correction in the call of a page in C
03/12/13	Possibility to use the functions’ variable property timer (from 03/02/13) included
05/13/13	Additional section for the operation of timer included
05/24/13	Accuracy on time stamp management functions introduced
06/10/13	Accuracy on ComGetFrameLength, ComSend and ComGetFrame functions introduced
10/16/13	Modification of values for SERx and SOCKx system words SockConnect, SockClose and SockReadStatus functions included, replacing

	TCPConnect, TCPClose and TCPReadStatus functions
18/03/14	Integration of version 8 of µLadder and version 4 of firmware. Different improvements done after the creation of the first µLadder User Manual (english version). Addition of the value names for SPD_SER* et FOR_SER*. Addition of ComPush and ComPushByte functions, modification of ComSend.