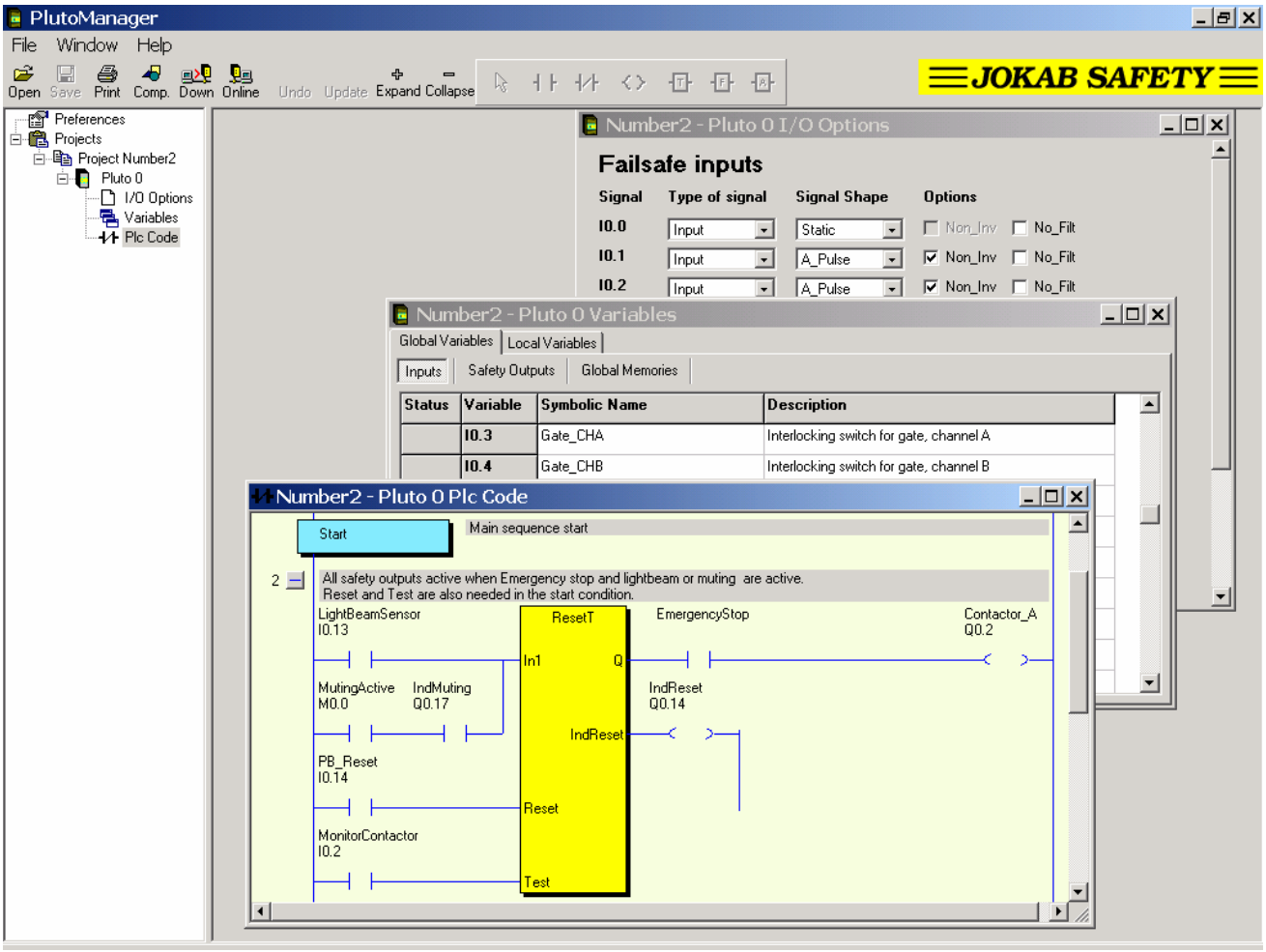


PLUTO

Safety-PLC

Programming manual

About this manual
This manual is divided in two parts; part 1 describing how to use the programming tool Pluto Manager and part 2 describing the language rules.

Part 1 begins with the chapter 'Making your first program' which leads you through the creation of a simple example. For first time users this can be a good way to get started.

The programming language is related to the programming standard IEC 61131-3. The programming can also be done in text form with a standard text editor. Before downloading to the system the code must be compiled to hex-format. Download of the hex-file to a PLUTO-unit and monitoring is possible by either Pluto Manager or a standard terminal program as Hyper Terminal.

Table of contents

Part 1

Part 2

Part 1
Pluto Manager

1 Safety note
Note that logic faults, like for example, an emergency stop that controls the wrong output cannot be detected by this software tool. Programs must therefore be reviewed and the safety applications carefully tested before being used in applications.

2 Installation
Installation of Pluto Manager is performed by executing the self extracting EXE-file (InstallPlutoManager… .exe) without any parameters. This leads the user through the installation allowing the user to select the appropriate location.

To run the program a registration code is required. However it is possible to use it without code in DEMO mode where compilation and on-line functions are disabled.

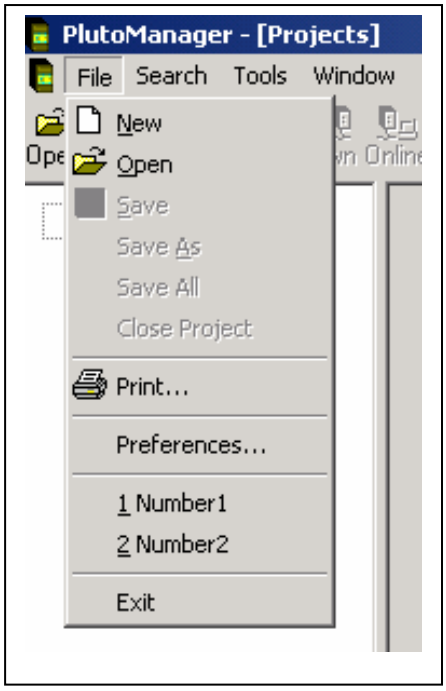
To run in full version, the program requires a registration code. The code can be entered by start of the program.

3 Making your first program

The quickest way to introduce yourself to the Pluto Manager is to write an application. This tutorial guides you through the creation of a Pluto program.

Creating a new project

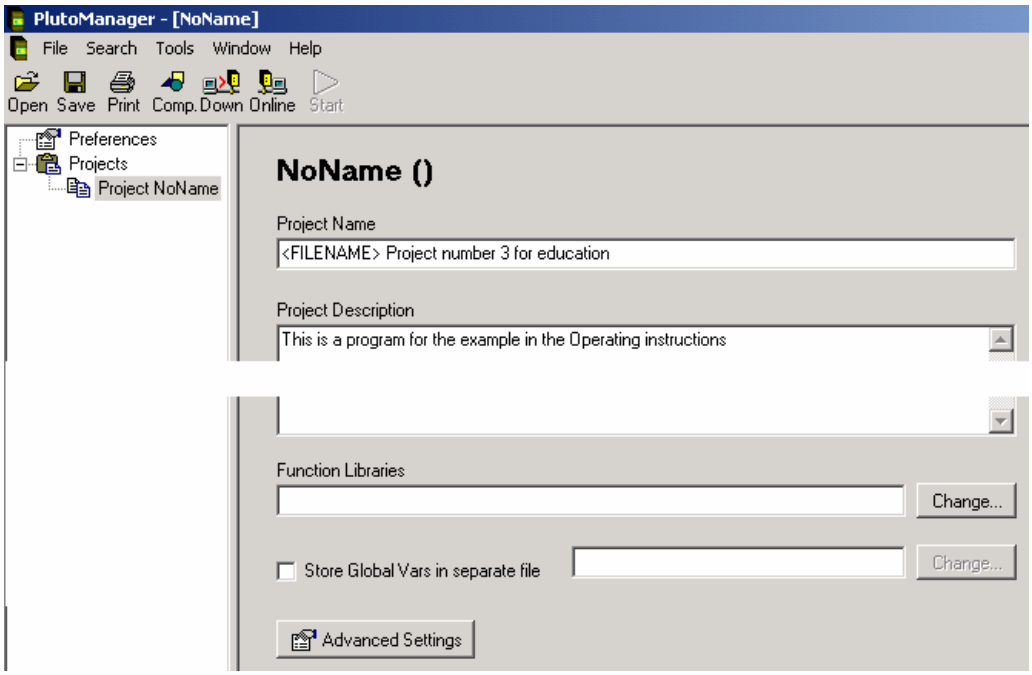
After opening Pluto Manager a new project can be created by choosing ‘New’ under the ‘File’ menu. If an existing program is to be loaded, select ‘Open’



Name and description

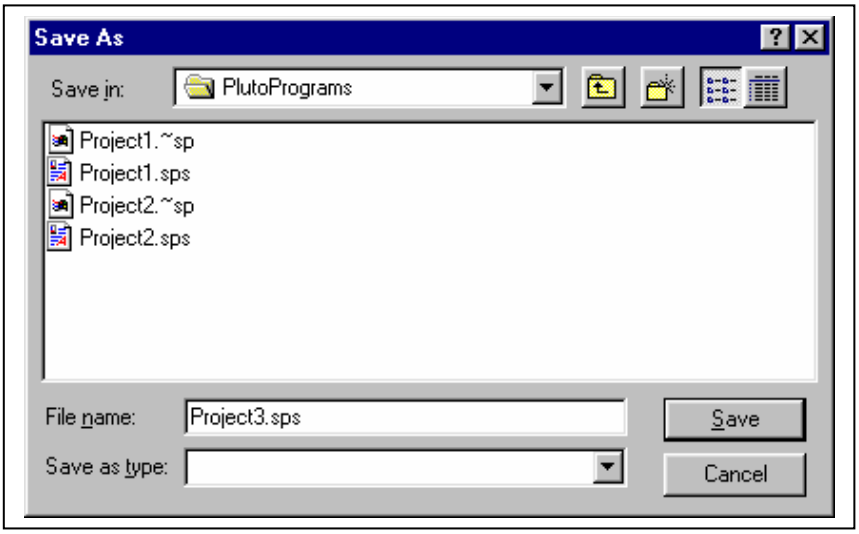
An initial page with fields for ‘Project Name’ and ‘Project Description’ is shown.

‘Project Name’ is later downloaded to the Pluto units and when going on-line it is checked. <FILENAME> is default and will be substituted with the program file name.
‘Project Description’ is just for making your own notes.



Saving

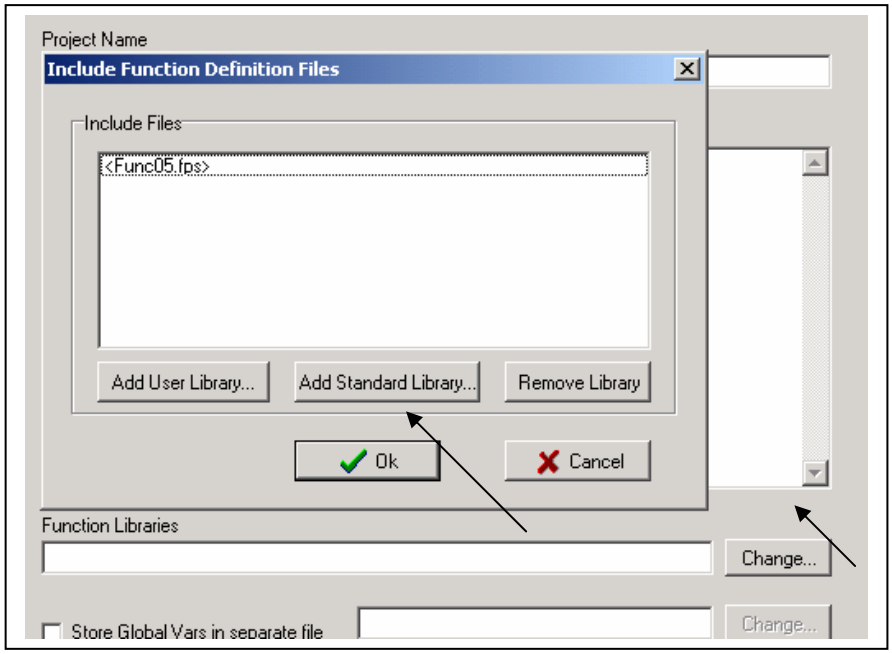
At this stage it can be a good idea to save for the first time. The toolbar provides quick mouse access to save. When the project is not saved before, Pluto Manager displays the Save As dialog box.
'Save' and 'Save As' can also be found under the File menu.
The source file is automatically saved with file extension .sps if nothing else is specified.



Selection of function block library

The Pluto system offers the possibility for using pre-programmed function blocks/macros for different safety functions and safety devices. These function blocks are stored in separate library files. Standard libraries are provided by Jokab but it is also possible to make user specific libraries.

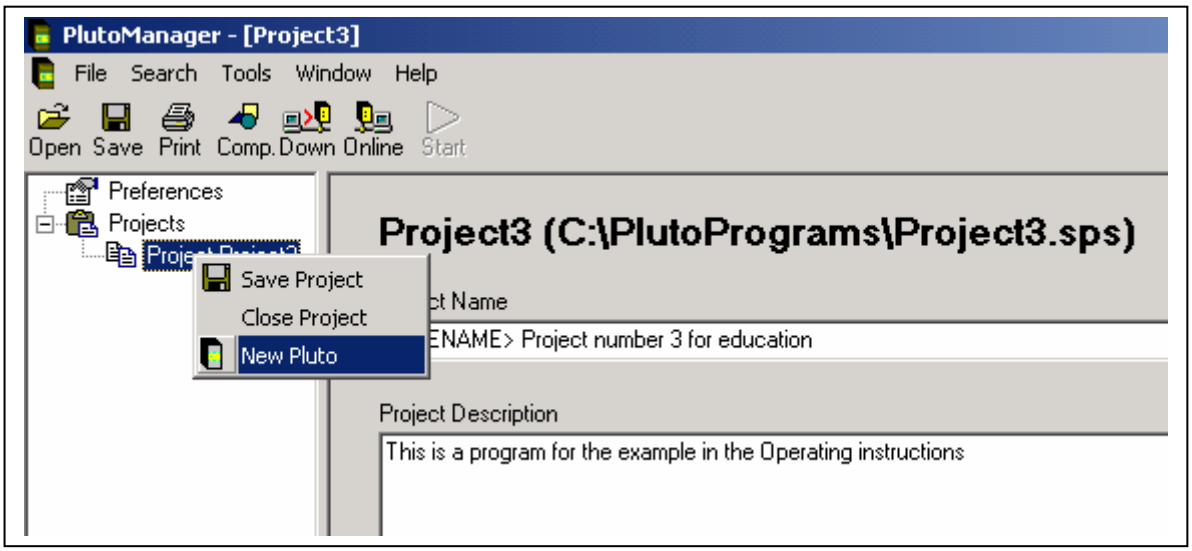
Select 'Function library', 'Change' and then 'Add standard Library' a list with available libraries is shown.



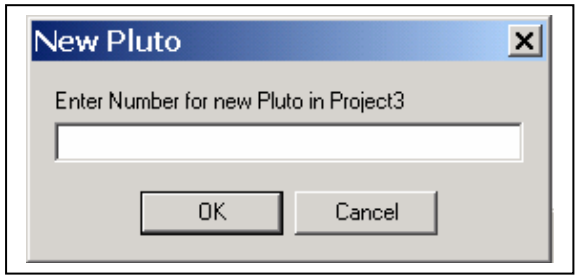
By 'Add standard Library' Pluto Manager looks for the files at '..\PlutoManager\Library' where they normally are stored by the installation program.
If 'Add User Library' is selected, Pluto Manager looks for the files in the directory where the project files are stored, in this case C:\PlutoPrograms.

Hardware setup

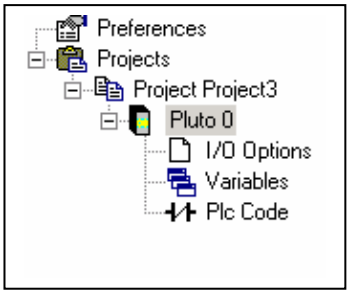
Next step is setting up the project according to the installed hardware.
Go to the tree menu to the left and make a right mouse click on the project name. Select ‘New Pluto’ when the new dialog is opened.



A dialog box for entering Pluto station number appears. The station number can be anything between 0…31.



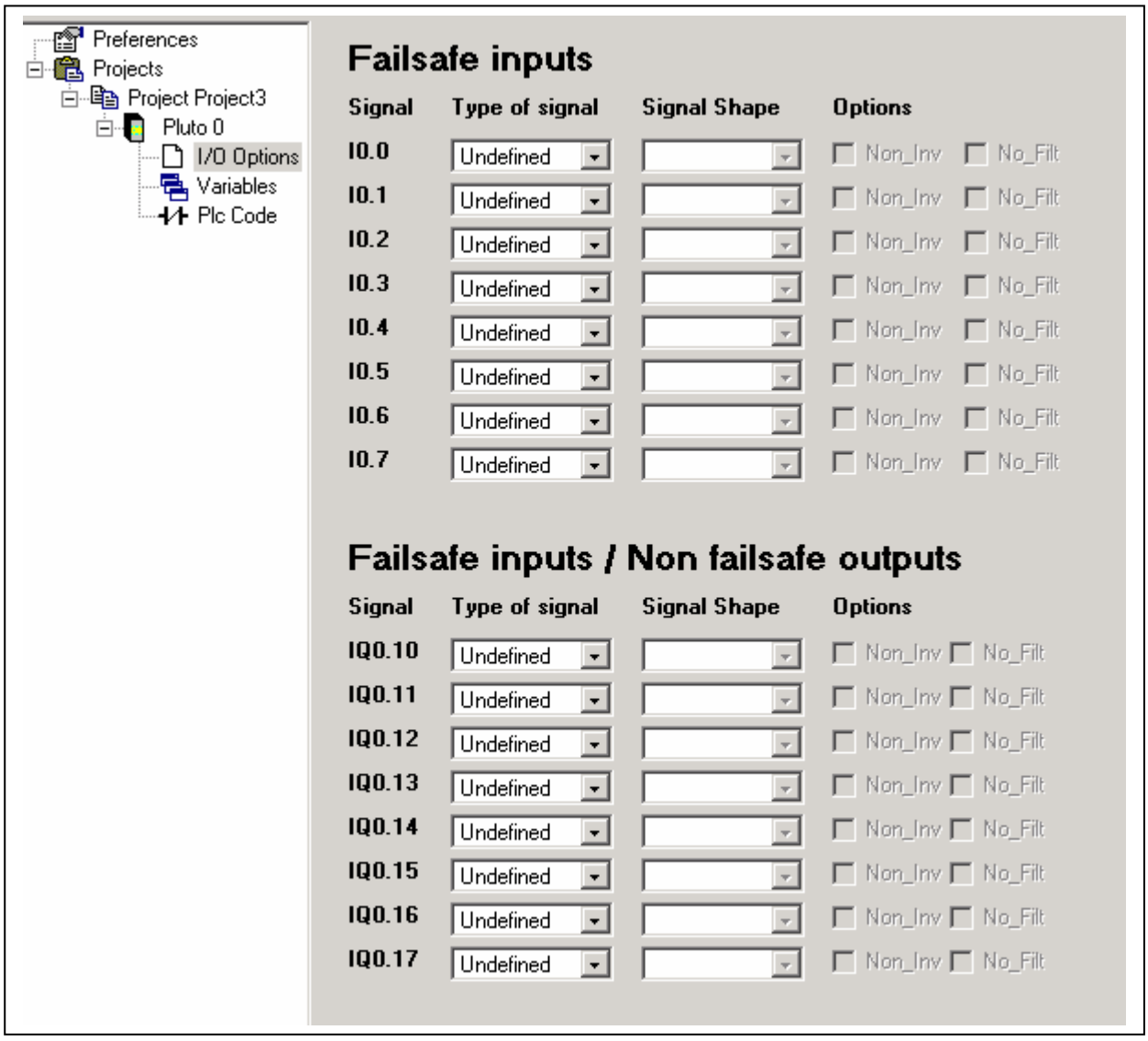
When the station number is selected the tree is expanded with a Pluto unit symbol and on a level below ‘I/O options’, ‘Variables’ and ‘PLC Code’ each representing a window.



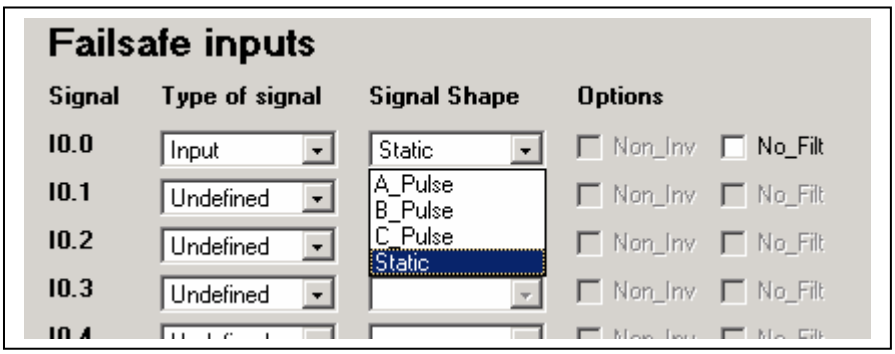
Configuration of I/O

Since the I/Os can be used in different ways, a configuration must be performed. This configuration must reflect the hardware design.

The 'I/O Option' window, lists the terminals I0…I7 and IQ10…IQ17. The safety outputs Q0…Q3 are not listed since they can only be used in one way.

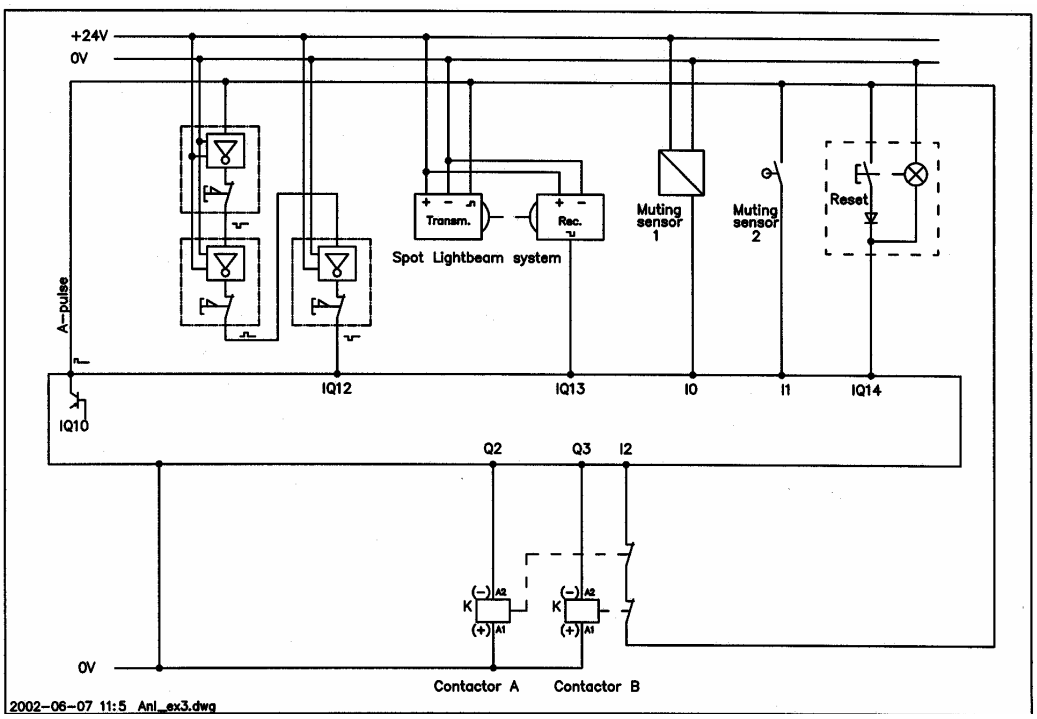


The preferred setting is selected via drop down lists.

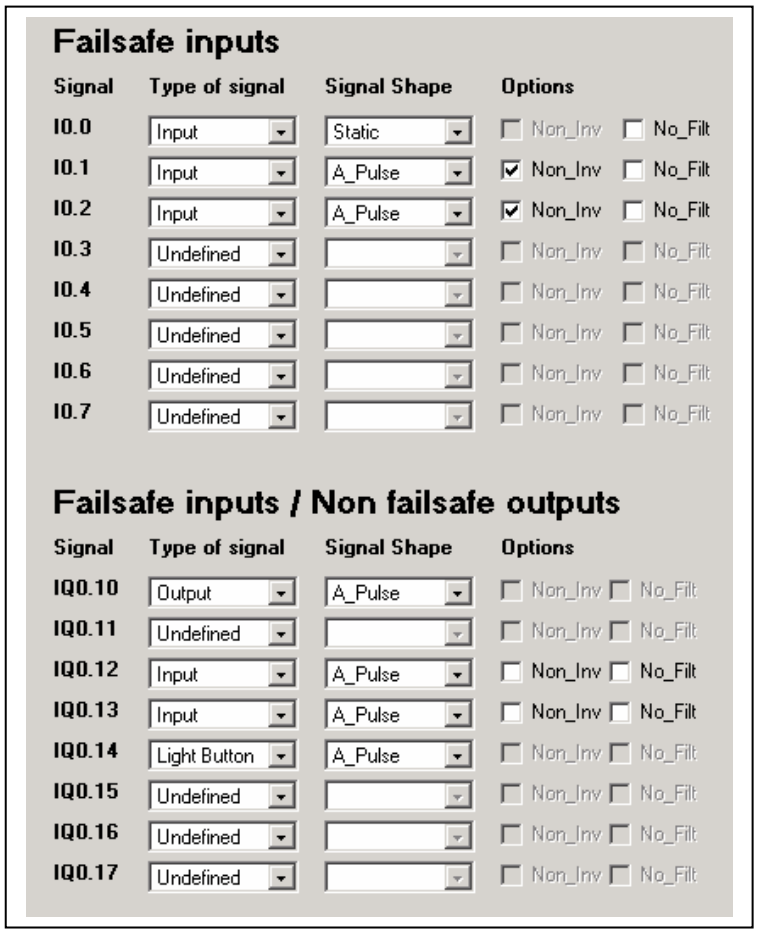


Example of setup of I/O-options

The pictures below show first, an example of wiring and then the corresponding configuration in the 'I/O Option' window.



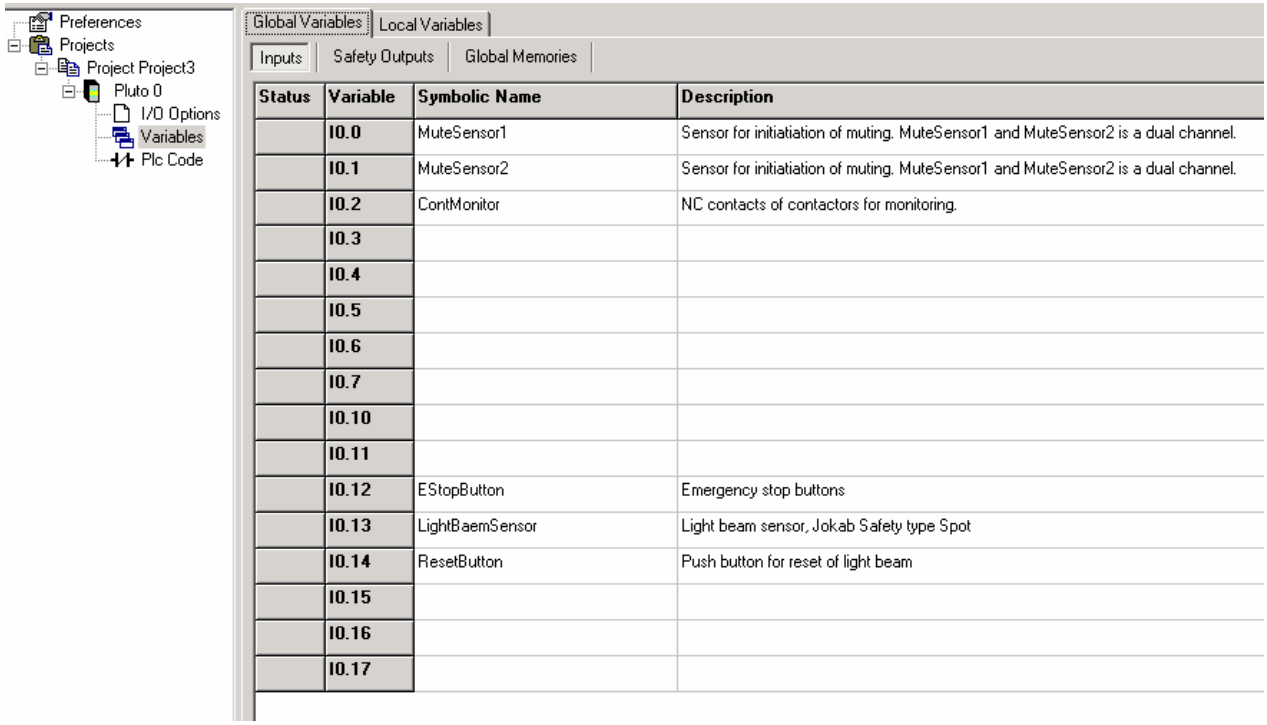
Note: The configuration of I/O is dependant on the hardware design. The correct use of inputs, outputs, dynamic signals etc. which is safety related, is normally the hardware designer's responsibility.



Naming of variables

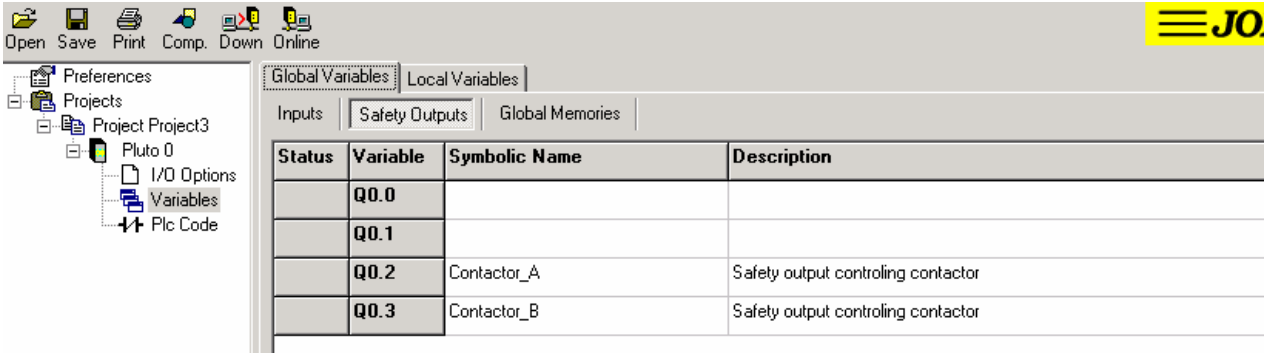
Open the window ´Variables' by a left mouse click on the corresponding symbol in the tree in the left field. All variables, inputs, outputs, memories, registers etc., can be given a name which further on, when programming the ladder logic can be used instead of the real I/O name. The naming can be left out or can be done afterwards.

In the field 'Description' an explanation of the variable can be made.



Status	Variable	Symbolic Name	Description
	I0.0	MuteSensor1	Sensor for initiatiation of muting. MuteSensor1 and MuteSensor2 is a dual channel.
	I0.1	MuteSensor2	Sensor for initiatiation of muting. MuteSensor1 and MuteSensor2 is a dual channel.
	I0.2	ContMonitor	NC contacts of contactors for monitoring.
	I0.3		
	I0.4		
	I0.5		
	I0.6		
	I0.7		
	I0.10		
	I0.11		
	I0.12	EStopButton	Emergency stop buttons
	I0.13	LightBaemSensor	Light beam sensor, Jokab Safety type Spot
	I0.14	ResetButton	Push button for reset of light beam
	I0.15		
	I0.16		
	I0.17		

Names and descriptions for inputs in Pluto 0.

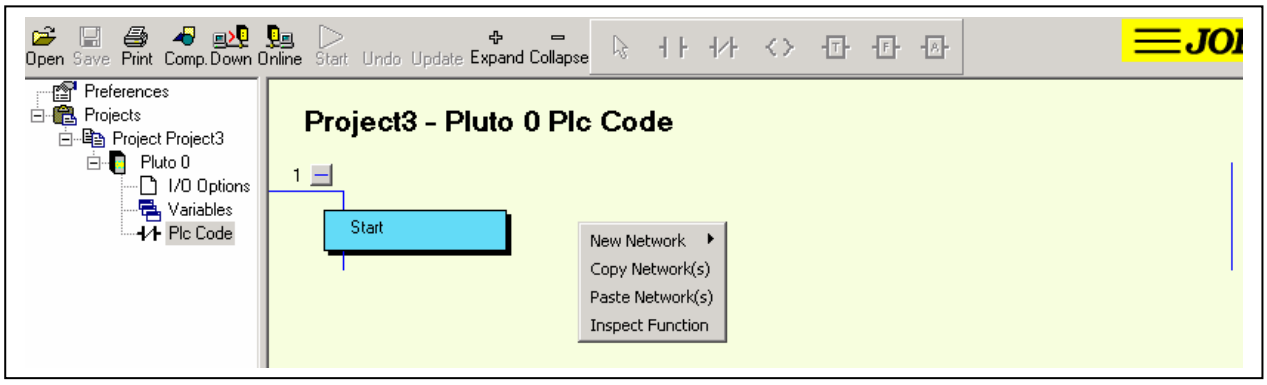


Status	Variable	Symbolic Name	Description
	Q0.0		
	Q0.1		
	Q0.2	Contactor_A	Safety output controling contactor
	Q0.3	Contactor_B	Safety output controling contactor

Names and descriptions for outputs in Pluto 0.

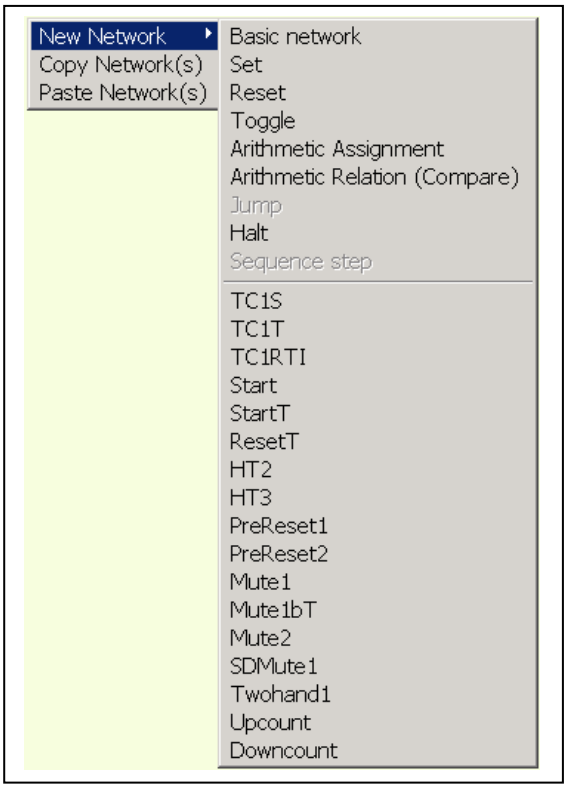
Programming the ladder logic

Open the window ‘PLC Code’ by a left mouse click on the corresponding symbol in the tree in the left field.
With a right mouse click a new network (rung) can be opened. A new network is always inserted after the network which the cursor is pointing at.
A dialog box with three options is shown, of which one is ‘New Network’.



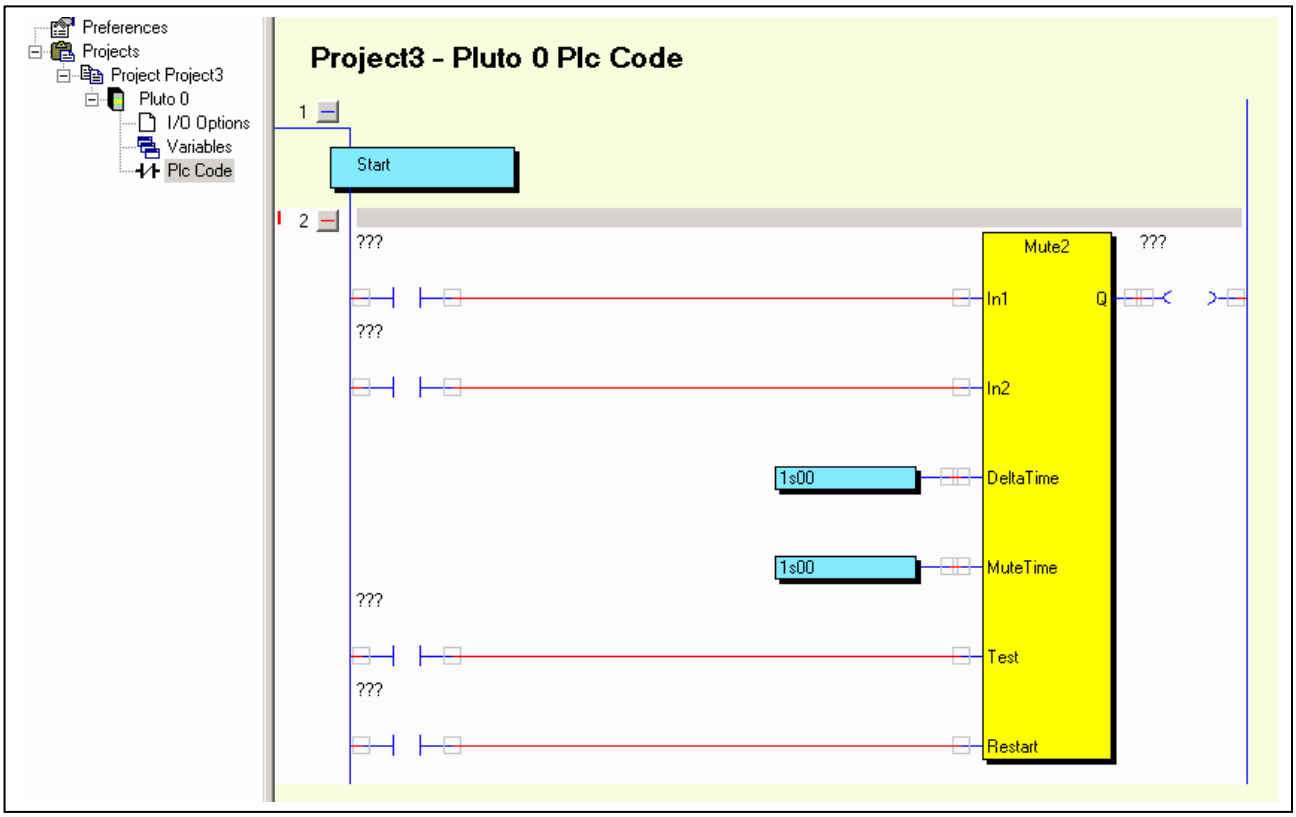
By pointing on ‘New Network’ a new menu is expanded. The menu has two parts divided by a delimiter. Before the delimiter basic ladder functions are listed, and after the delimiter available function blocks are listed if a function block library is selected. See previous ‘Selection of function block library’. The block functions are described in a separate document.

The menu is divided into two parts by a horizontal line. Above this line basic ladder functions are listed, below are the available function blocks (if a function block library has been selected)

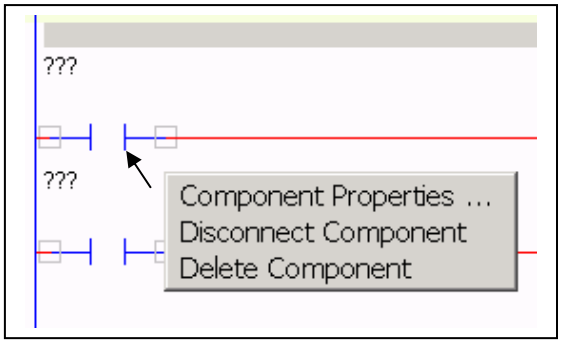


In this example we need a muting function and have found that the block ‘Mute2’ is suitable. A left mouse click on ‘Mute2’in the menu generates a ladder network showing the ‘Mute2’ block.

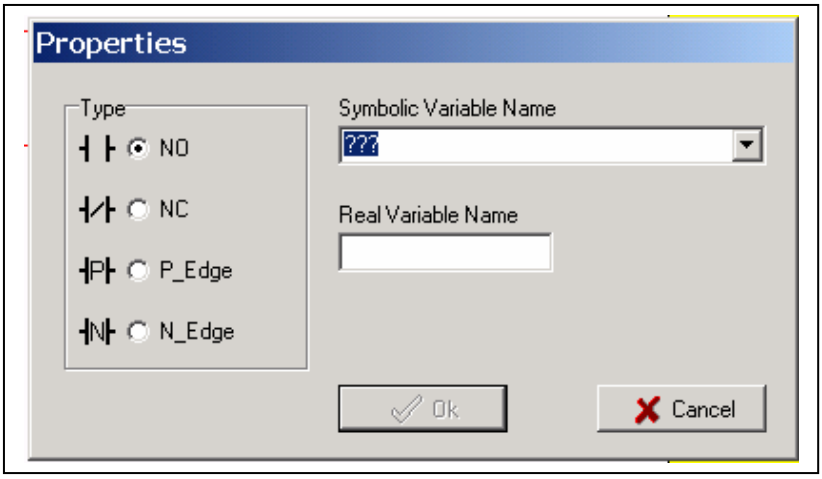
The highlighting of the network means editing mode. Each network has to be edited separately.



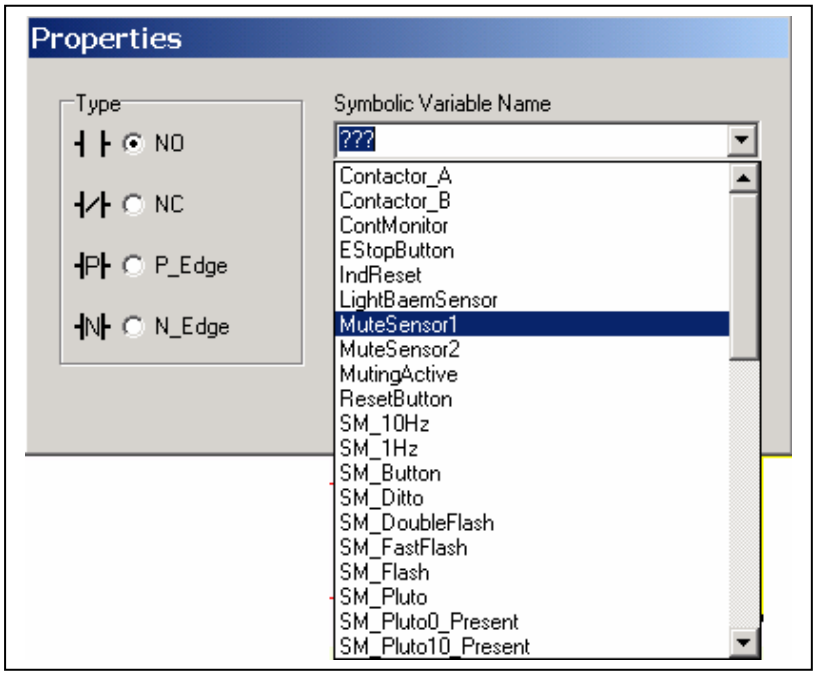
The ladder components which are marked ‘???’ must now be defined or in some cases deleted. By a right mouse click on a component three options are available. ‘Component Properties’ leads to the next dialog box, and ‘Disconnect Component’ disconnects it from the red connection lines.
By a left mouse click the ‘Component properties’ box is selected directly.



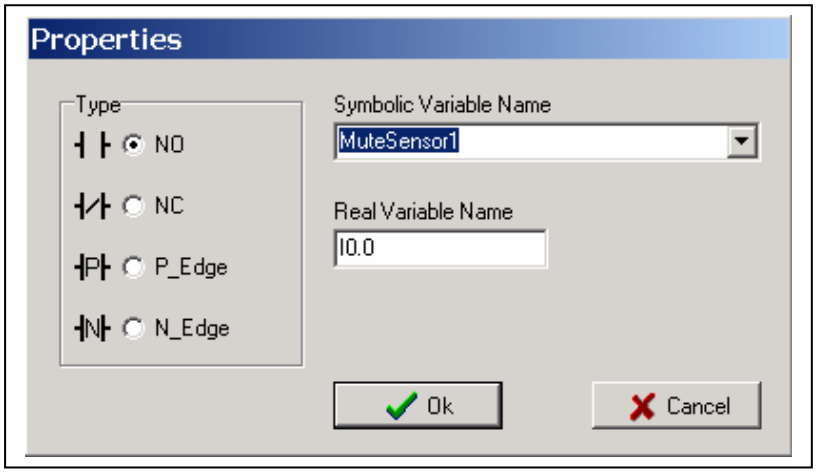
The property box for a contact symbol gives the choice for normally open, normally closed, positive or negative edge pulse function. There are two ways of entering a variable name, either giving the 'Real variable name', e.g. I0.0, I0.1, M0.3.., or by opening of the list under 'Symbolic Variable Name'.



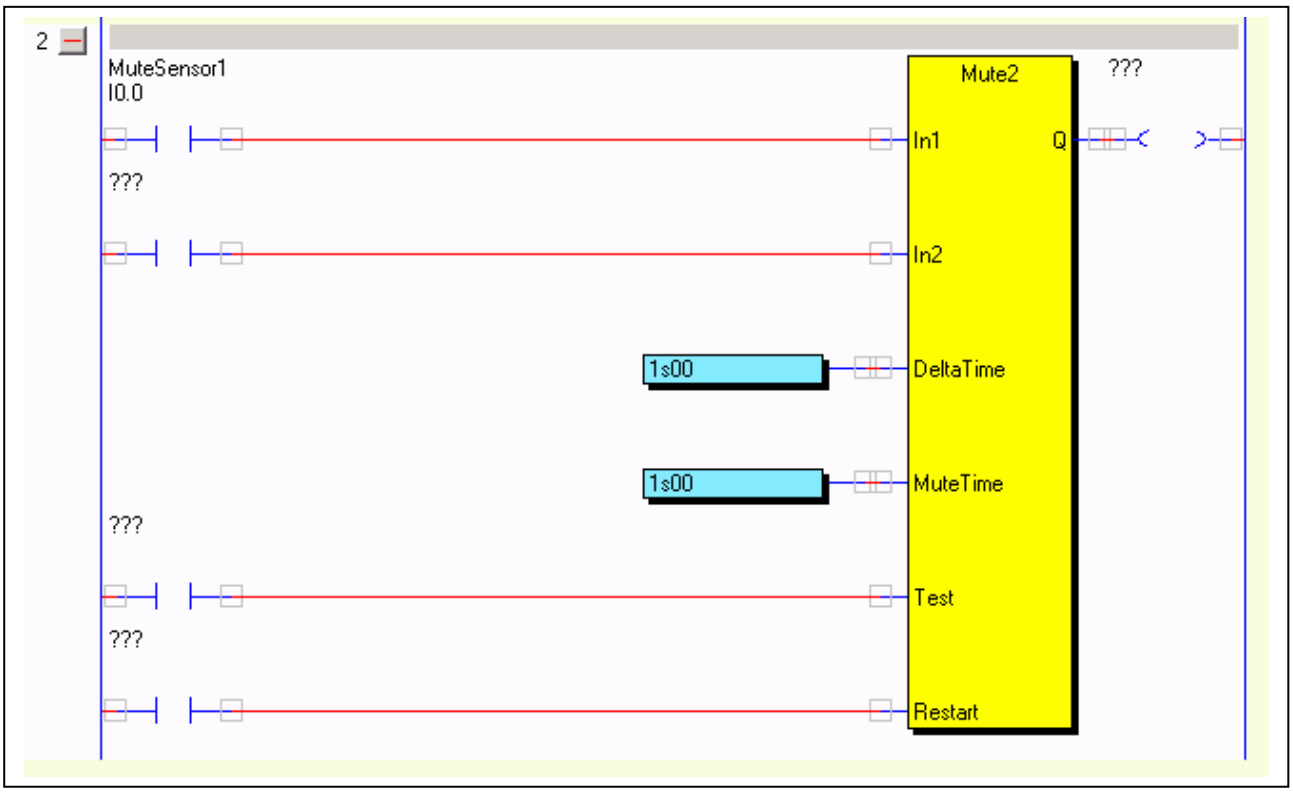
In the list under 'Symbolic Variable Name' all variables which have been given a name can be found.



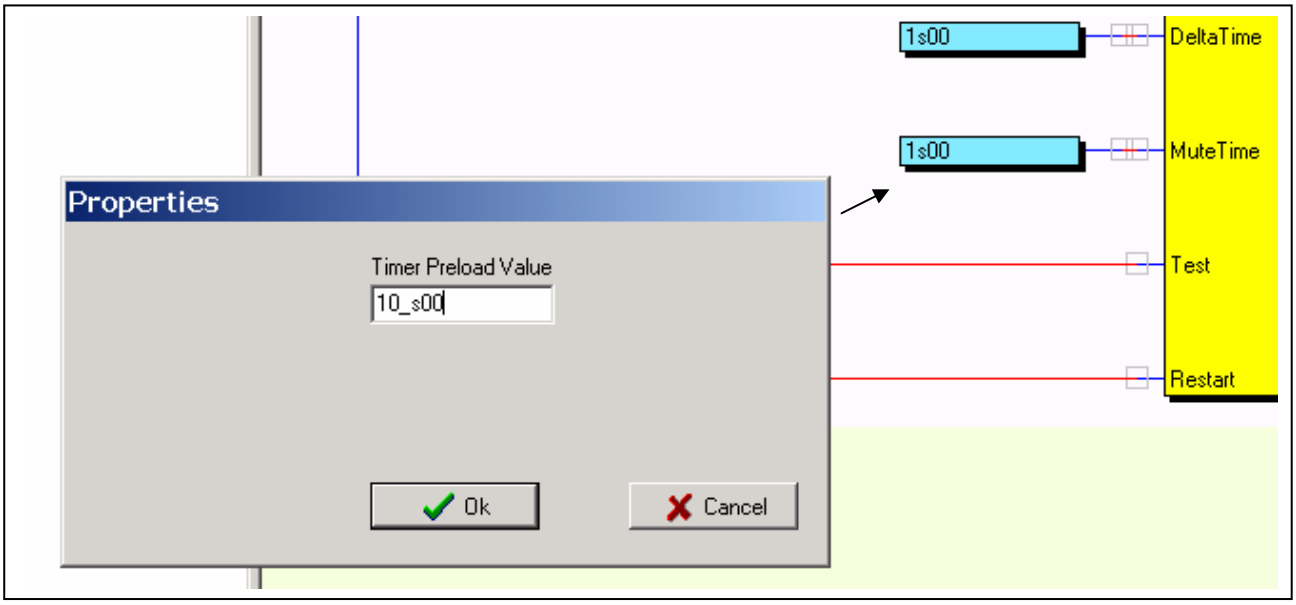
Confirm with a click on 'OK'.



After selection, the component is labeled with both symbolic and real variable name.

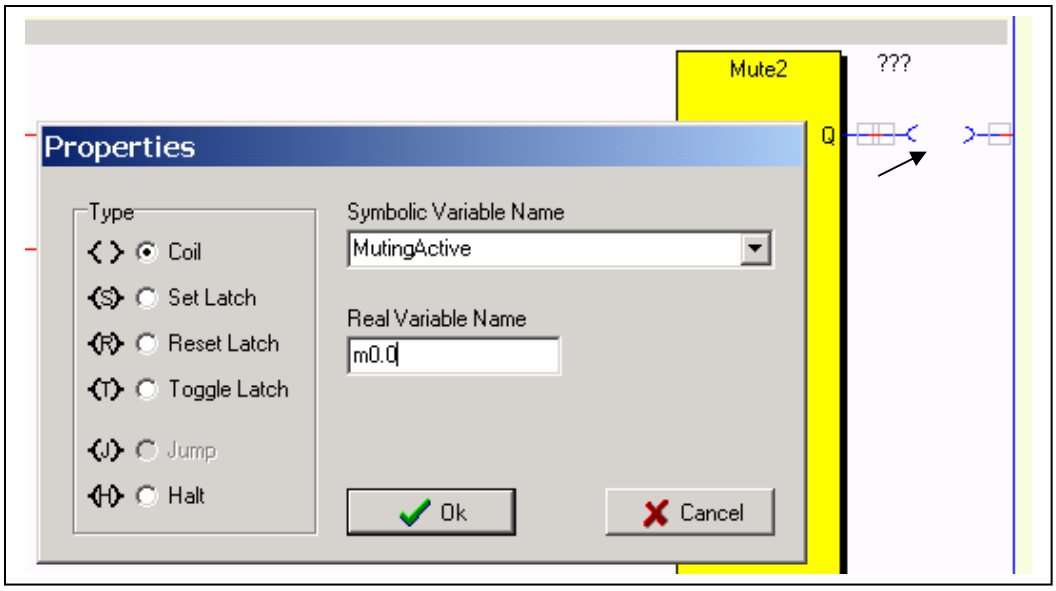


The timer values can be changed in the same way but a different dialog box with a timer value is viewed. ‘s’ is used as the decimal point.

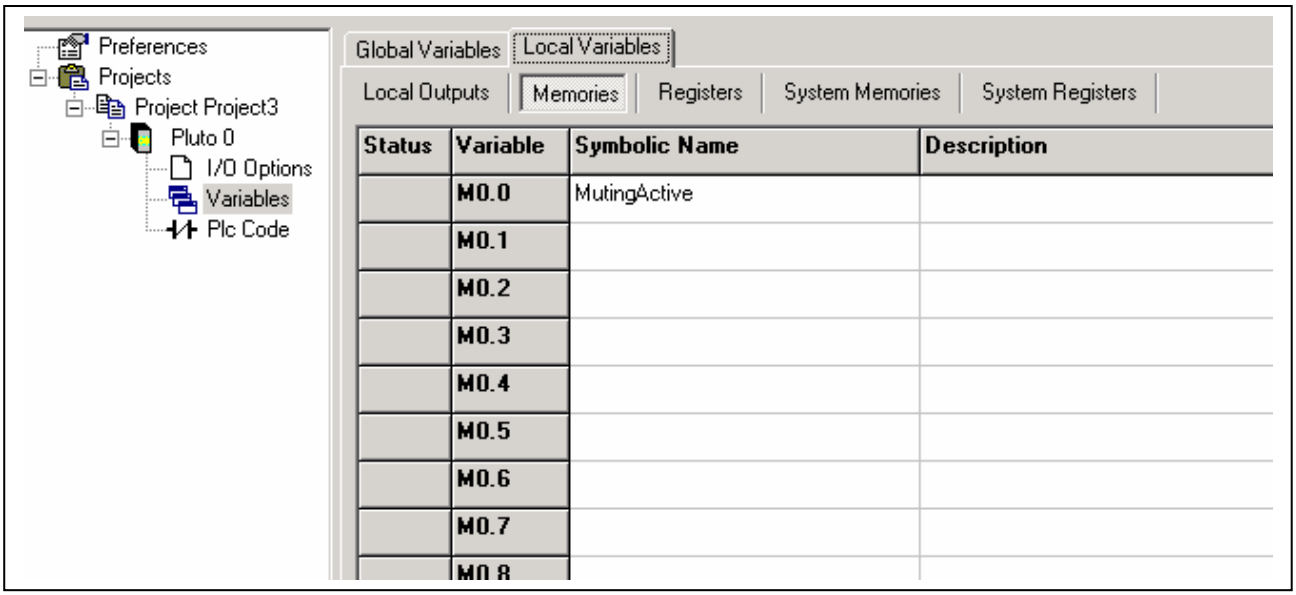


The output from a function block can be connected directly to a physical output (Q), a memory (M or GM) or to an input in another block, in this case a memory (M0.0).

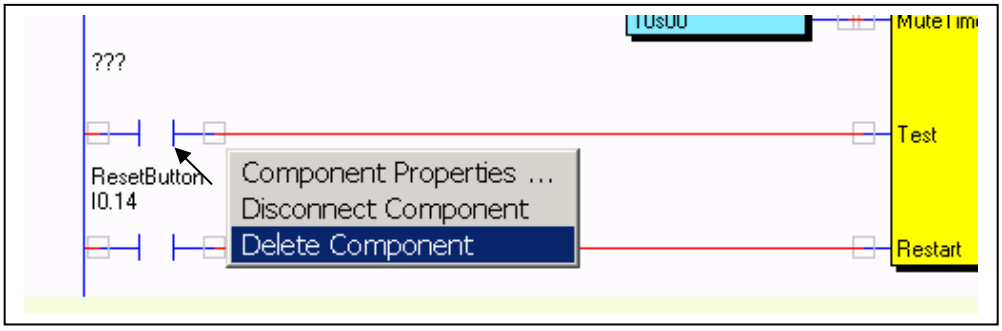
By a double click on the ladder component we get a dialog box with different output functions.



To avoid mistakes the memories should be given a name directly by use. This can be done by opening the window 'Variables' during the editing of a ladder network (except when a dialog box is shown).



The input for Test on the 'Mute2' function block shall not have any input condition in this example. The component is therefore deleted.



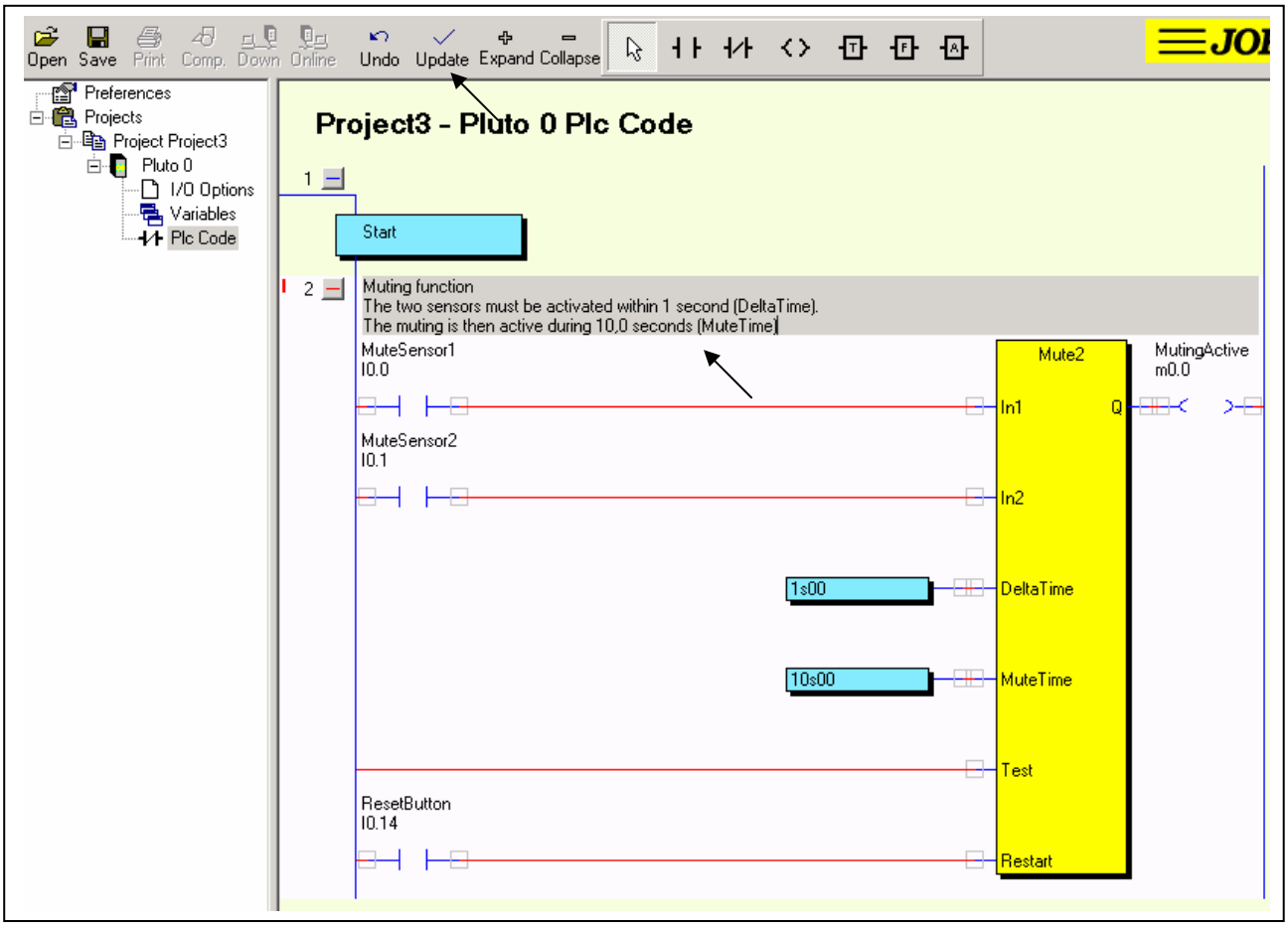
Adding comments and finalising the network

At the top of a network there is a field for comments. Everything that is typed on the keyboard during edit mode is written into this field.

When the editing of the network is completed it can be closed for editing by a left mouse click on 'Update'.
Alternative ways are:

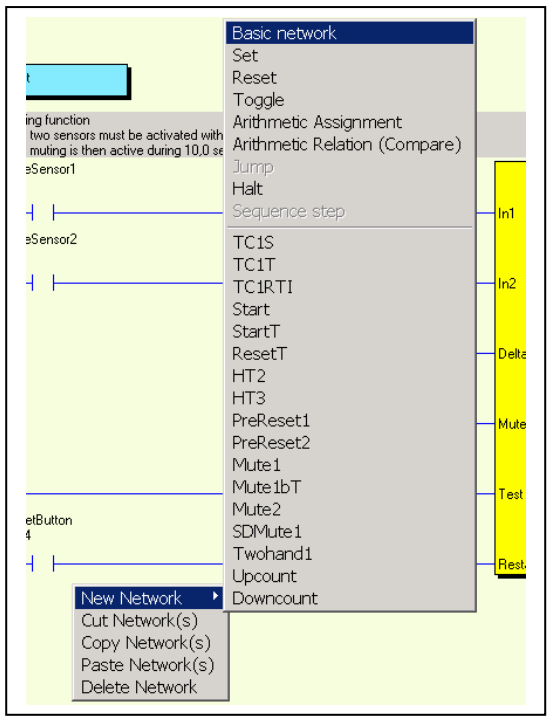
- to press 'F3' key or
- to press 'Esc' followed by answering 'Yes' in a dialog box.

If 'Undo' is pressed, everything in the edited network is restored as it was before it was entered. Instead of 'Undo' F2 can be used.

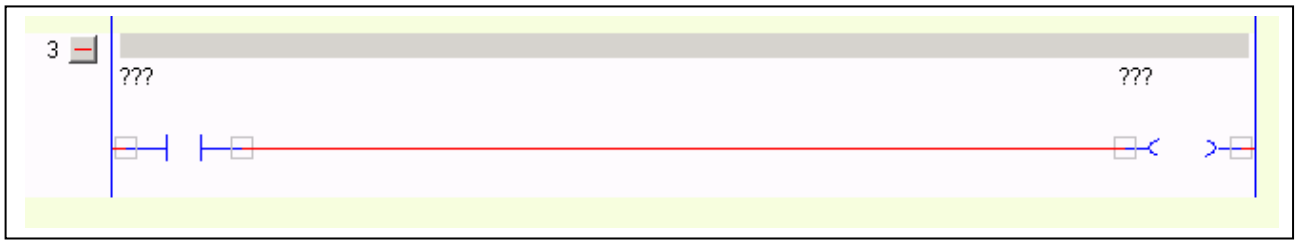


Next network

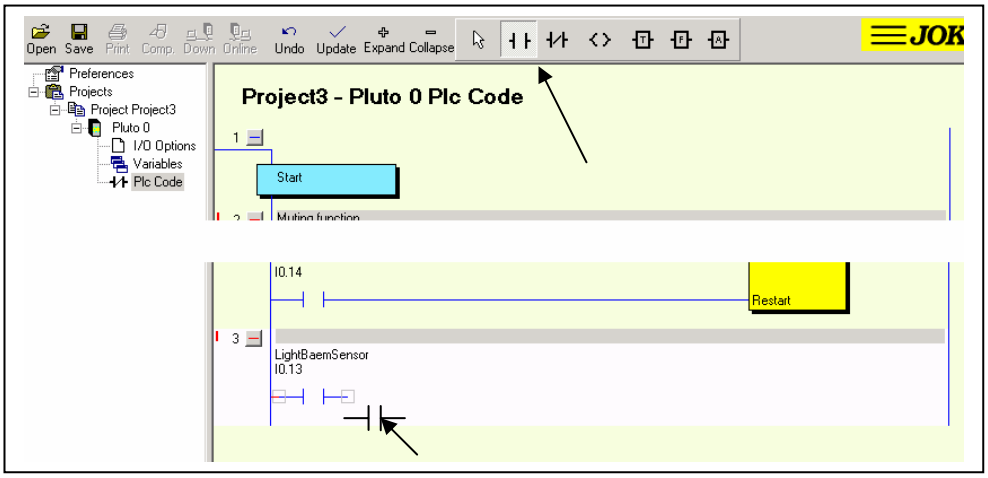
In the next network we shall put together our safety functions and set a safety output. Just for practice we select a ‘Basic network’, instead of a function block this time.
Make a right click somewhere in the first network. Select ‘New network’ and ‘Basic network’.



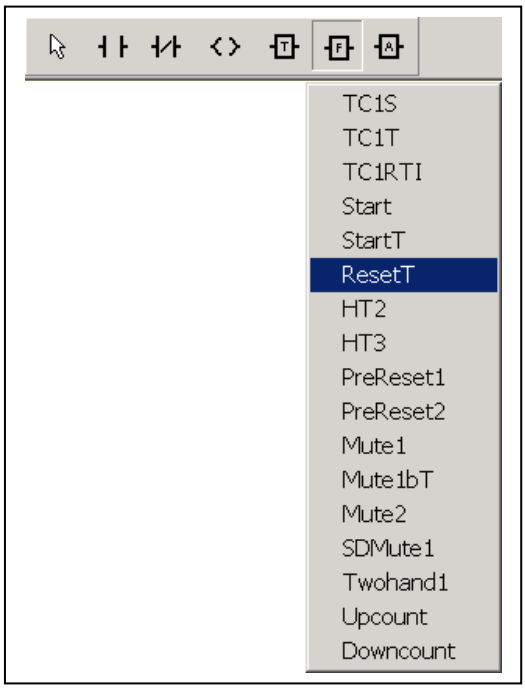
The result is that we get a network with one ladder NO contact and an output.



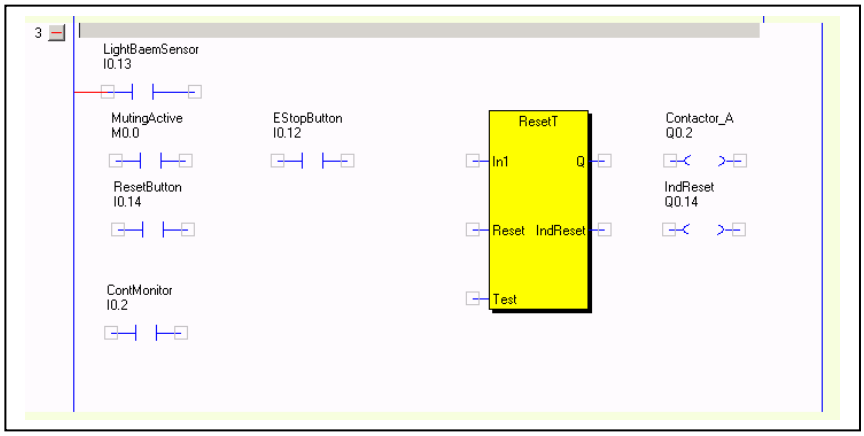
After deletion of the output and changing the properties for first ladder contact to ‘LightBeamSensor’, we start to put in new ladder functions by selecting from the toolbar. Make a left mouse click on the symbol for NO contact. The cursor then takes the form of the NO contact. Place the contact where you want to have it in the network, fix it with a left mouse click and fill in the properties.



In this network we need a function block called ‘ResetT’. This is a block with one safety input which can handle monitoring of a Reset push button with an indicator. By clicking on the F symbol, the list with available function blocks is shown from where ‘ResetT’ can be selected and inserted in the network.



Continue selection of the other components needed in the same way. Function blocks can be found under the symbol F, Timers under ‘T’ and arithmetic functions under ‘A’.



Connecting the components.

When the arrow symbol in the toolbar is highlighted it is possible to draw, delete and change lines between the components. In this mode it is also possible to drag components around.

Operations

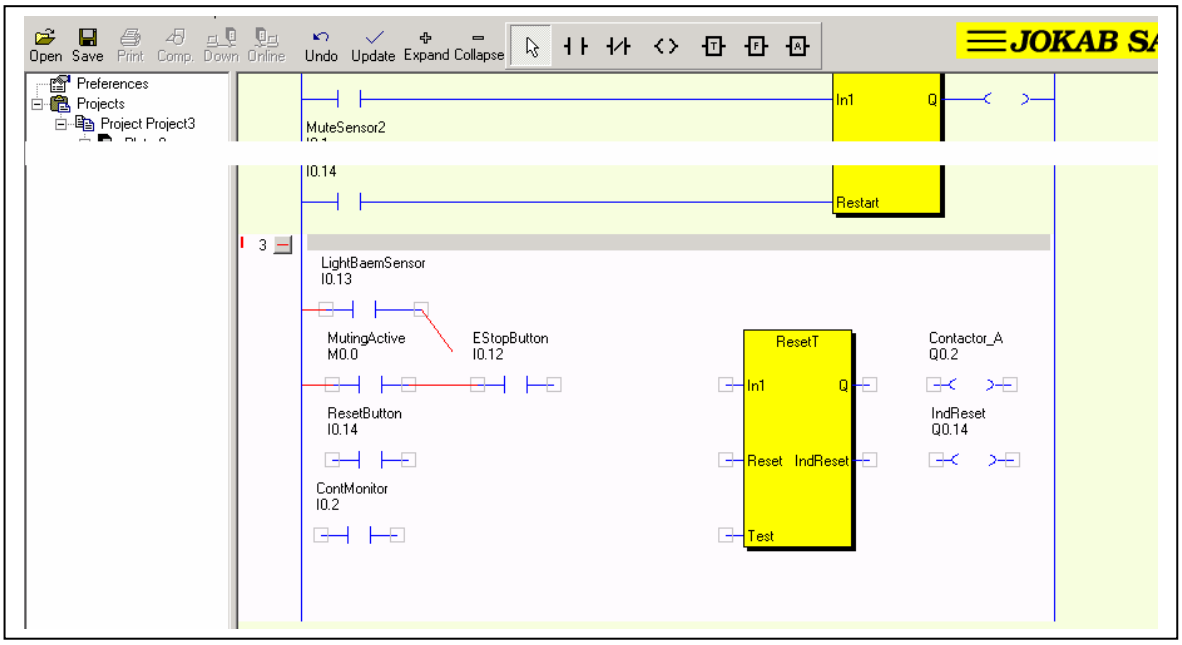
Draw a line: Do a left mouse click (and release the button) in a 'hit box' for a component. The 'hit boxes' show the connection points. Move the cursor to the component where the end of the line is to be connected and fix it with a left click.

Change a line: By clicking the mouse on a line outside the 'hit boxes', the line is grabbed. It is now possible to:
- Stretch it to a third point and fix it with a left mouse click.
- Go to one of the 'hit boxes' and disconnect it with a left mouse click. When the line is detached it can be fixed to another component or deleted with a mouse click outside a 'hit box'.
- Make a right mouse click and a dialog box 'Delete line' is shown.
- Un-grab it with a new left mouse click.

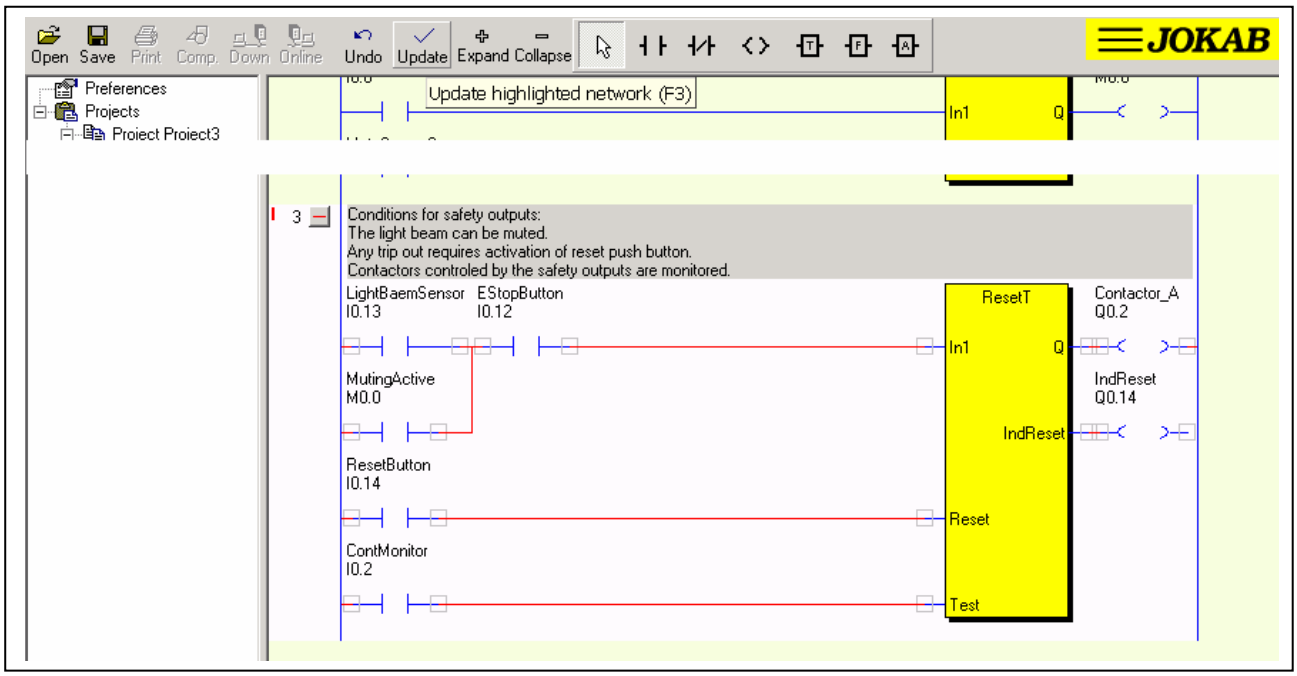
Change components properties: A double left mouse click on a component leads to a dialog box for changing Variable name, NO, NC, Pulse function etc.

Change components: By a right mouse click on a component, a dialog box with three options is shown.
- 'Components properties..' for giving or changing function or name.
- 'Disconnect component' for deleting all connections to the component.
- 'Delete component' for deletion of the component.

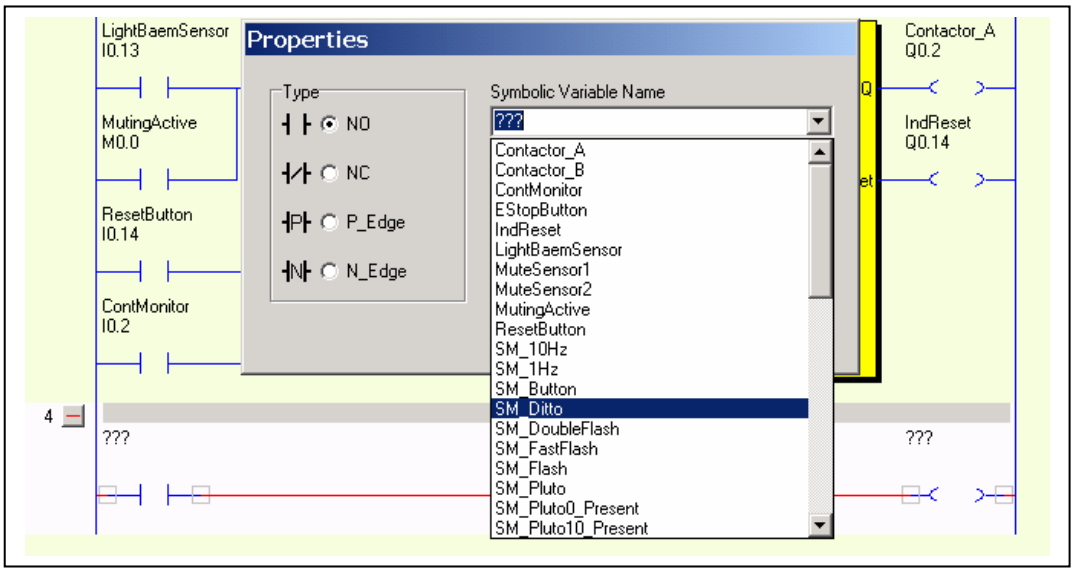
Moving components: Press and keep left mouse button down on a component and drag it. Release the mouse button at the new place required.



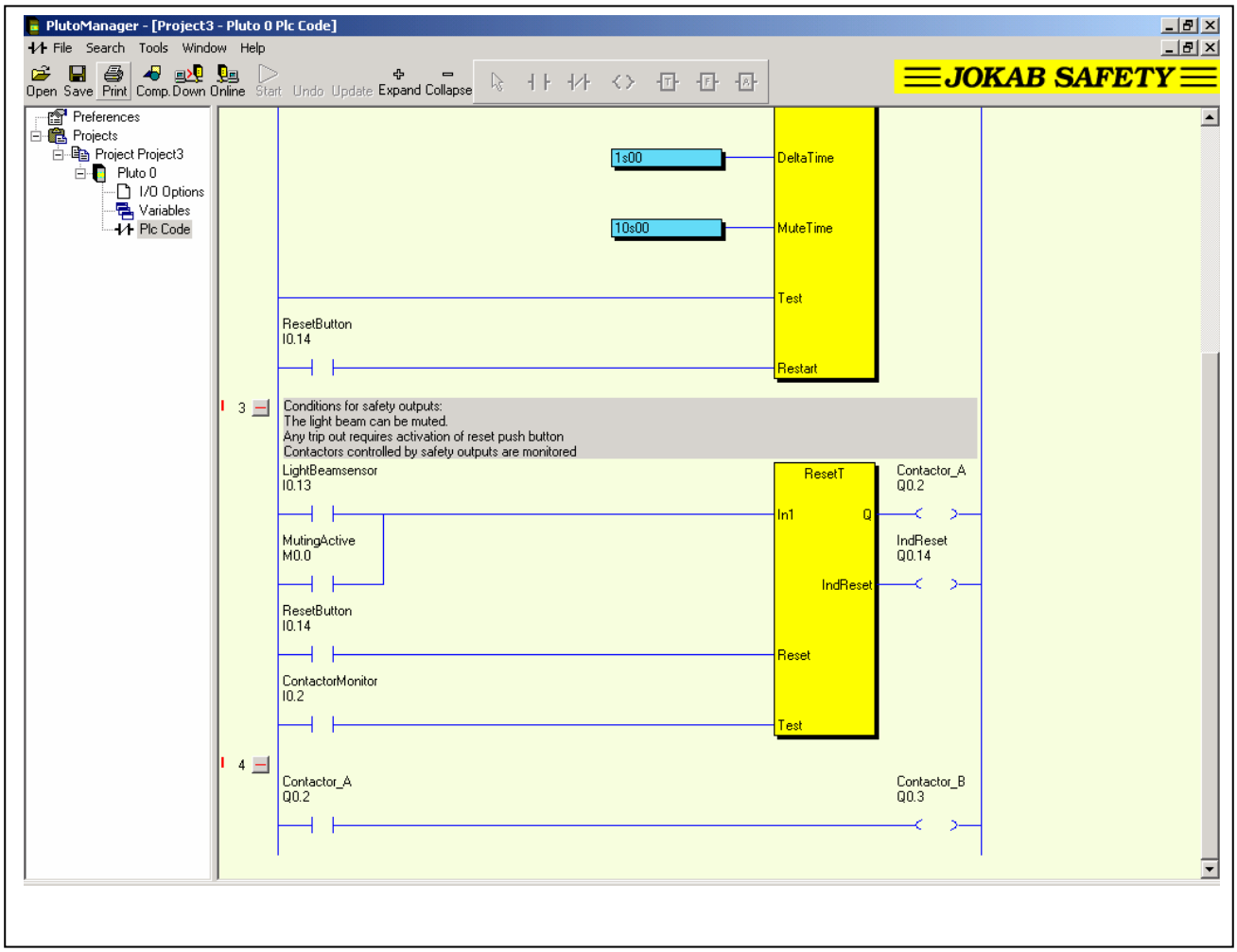
When all components are inserted and connected, press the ‘Update’ button or F3.
Note that the function block output ‘IndReset’ is a secondary output. This block output can be left open if there is no use for it. If a component (Q, M or GM) is connected to it, the right side of this component shall be left open and not be connected to the right common line.



After updating we continue with the last network in this program. The safety function is to control the two contactors A and B, connected to different outputs. We shall program contactor B to work exactly as Contactor A. Instead of making an equal network as for contactor A, we can use the system memory SM_Ditto (SM_.1) which contains the logic result of the previous network.
Open a new basic network, then open the ‘Properties’ dialog box for the first contact. In the list ‘Symbolic Variable Name’, SM_Ditto can be found. Finally set the properties for the output to ‘Contactor_B’

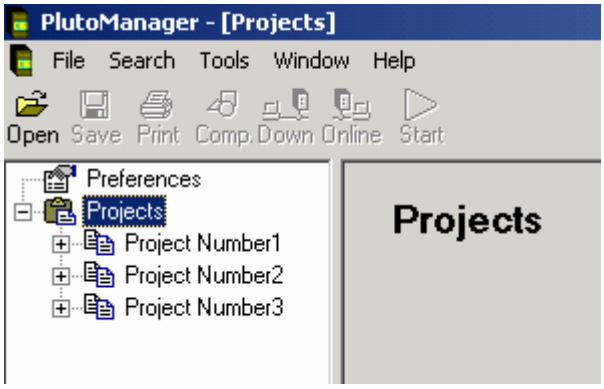


Finished


PlutoManager - [Project3 - Pluto 0 Plc Code]
File Search Tools Window Help
Open Save Print Comp. Down Online Start Undo Update Expand Collapse
JOKAB SAFETY
Preferences
Projects
Project Project3
Pluto 0
I/O Options
Variables
Plc Code
1s00
DeltaTime
10s00
MuteTime
Test
ResetButton
I0.14
Restart
3
Conditions for safety outputs:
The light beam can be muted.
Any trip out requires activation of reset push button
Contactors controlled by safety outputs are monitored
LightBeamsensor
I0.13
ResetT
In1
Q
Contactor_A
Q0.2
MutingActive
M0.0
IndReset
Q0.14
IndReset
ResetButton
I0.14
Reset
ContactorMonitor
I0.2
Test
4
Contactor_A
Q0.2
Contactor_B
Q0.3

4 Projects Open, close, save, ….

After loading the Pluto Manager two fields are shown. The left field contains a tree menu which always is visible and is used to navigate between the different pages. These pages are shown in the right field on the screen. Several projects can be open simultaneously.

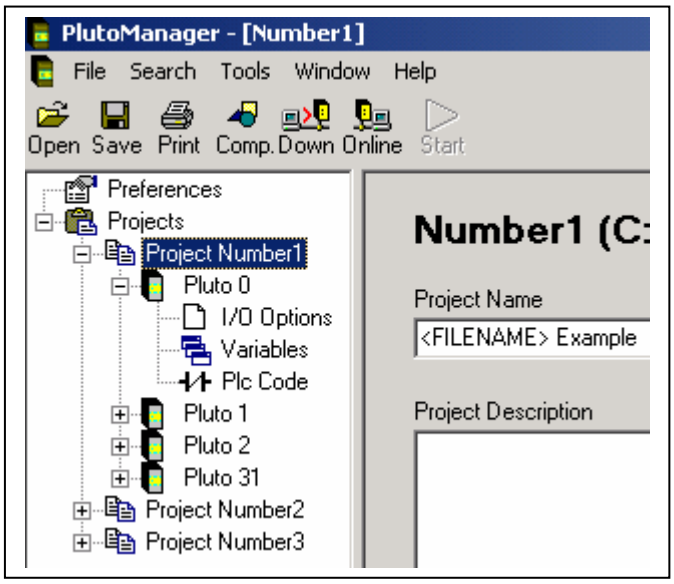


Commands:

Open a new project:	- Right mouse click on ‘Projects’ in the tree menu and select ‘New Project’, or - Open ‘File’-menu → ‘New’
Open an existing project:	- Right mouse click on ‘Projects’ in the tree menu and select ‘Open Project’, or - Use the shortcut ‘Open’ in the toolbar, or - Open ‘File’-menu → ‘Open’
Close project:	- Right mouse click in the tree menu on project name. Select ‘Close Project’, or - Mark one of the open projects in the tree menu. It does not matter if it is at a lower level in the tree below the project name icon. Open ‘File’-menu → ‘Close Project’.
Save:	- Right mouse click in the tree menu on project name. Select ‘Save Project’, or - Mark one of the open projects in the tree menu. Use the shortcut ‘Save’ in the toolbar, or - Mark one of the open projects in the tree menu. Open ‘File’-menu → ‘Save Project’.
Save all:	Open ‘File’-menu → ‘Save All’. All open projects will be saved.

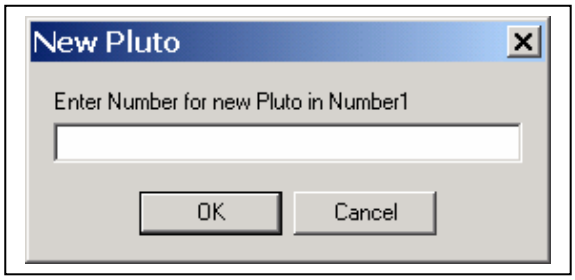
5 Bus configuration

The Pluto units can work as separate units or together on the bus. A project can be set up to contain 1-32 Pluto units. The programs for all these units will then be stored in one .sps-file which is downloaded into each unit.



Command:
Right mouse click in the tree on 'Project [*name*]' → 'New Pluto'

Enter a station number 0-31.

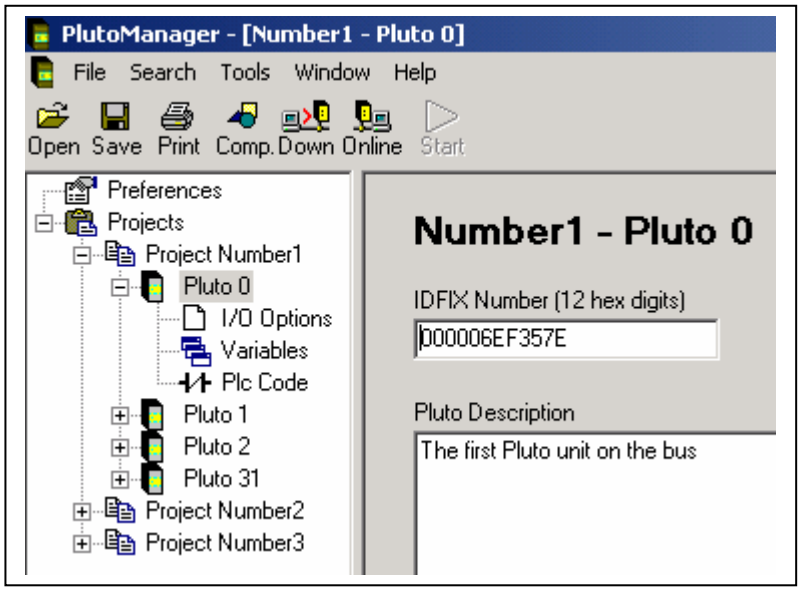


The station number is a part of the I/O addresses. Inputs in Pluto 0 are named: I0.0, I0.1, I0.2,… and in Pluto 1: I1.0, I1.1, I1.2,… etc..

Identifier ‘IDFIX’ number

When there are several Pluto units on the bus, each is equipped with an external identifier circuit containing a unique hexadecimal number. (See also Hardware manual.)

The identifier number shall be filled in to the field ‘Identifier Number’. Since the numbers are not known at this stage of the project, it can be left out until it is time for download and test of the system.

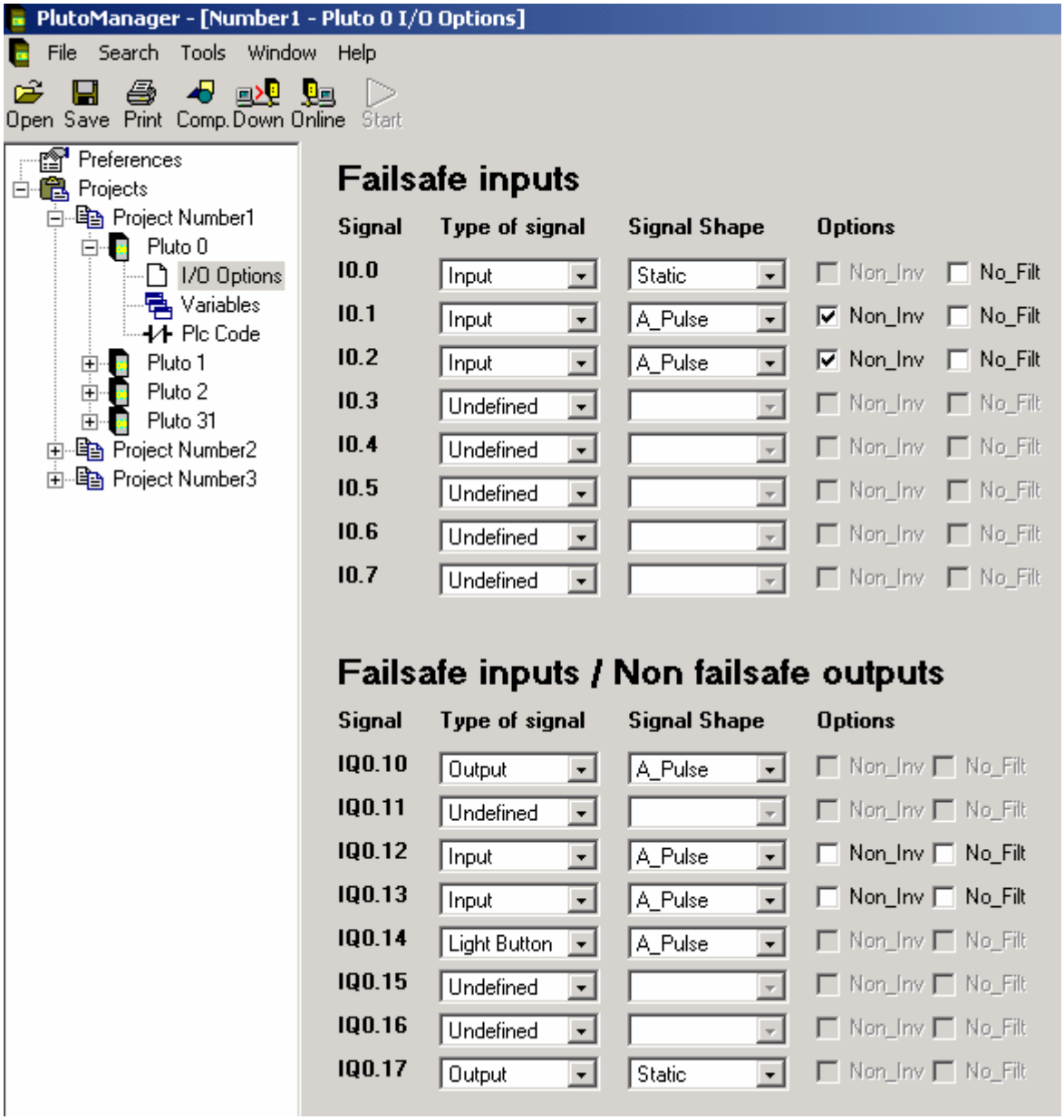


Pluto description

The field is just for comments and descriptions and is not downloaded to the Pluto unit.

6 I/O Options

The 'I/O Option' page is shown by a mouse click on the corresponding icon in the tree menu. The settings are filled in by using the drop down lists and tick boxes. Illegal combinations are automatically blocked.



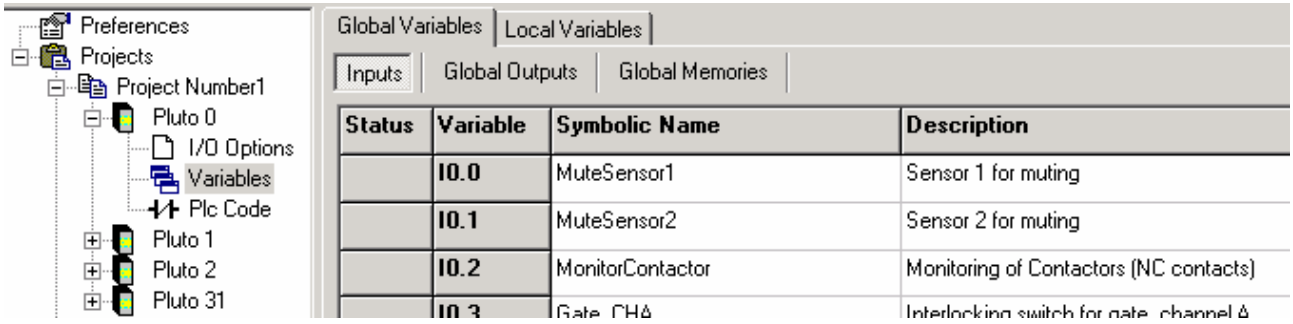
7 Variables

By a mouse click on 'Variables' in the tree menu, pages for each type of variable can be reached. Here it is possible to give an individual name and description for each of the variables.

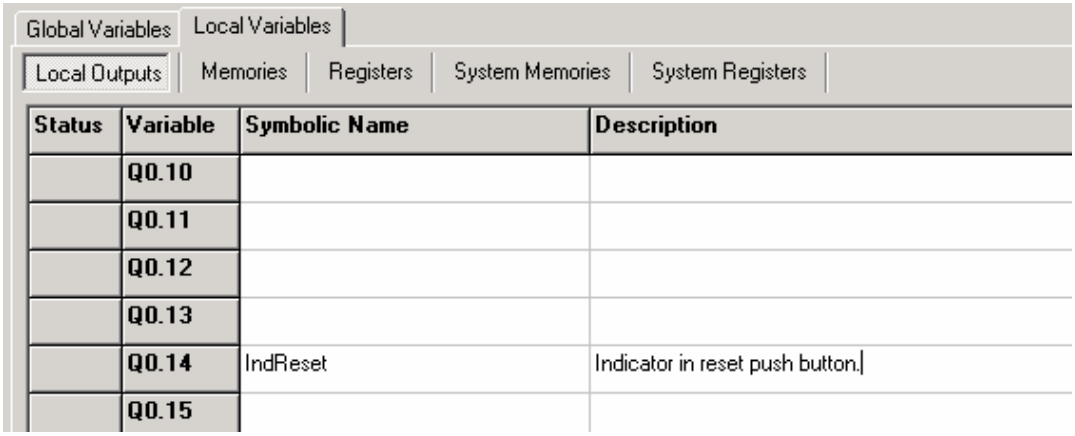
Symbolic Name: A variable can be given a name which can be used instead of the real I/O name further on by ladder logic programming. The naming can be left out or be done later.

Description: The description has no influence on other functions.

At the top of the page there are tabs representing each kind of variable type. First of all there are the two main groups, Global and Local variables. Global variables can be used by all Pluto units connected to the bus, local variables are just for internal use in one Pluto unit.



Global Variables:
- 'Inputs', I_.0 …I_.17 ('_' represents the Pluto station number)
- 'Global Outputs', Q_.0 …Q_.3, (Safety outputs)
- 'Global Memories', GM_.0 … GM_.11

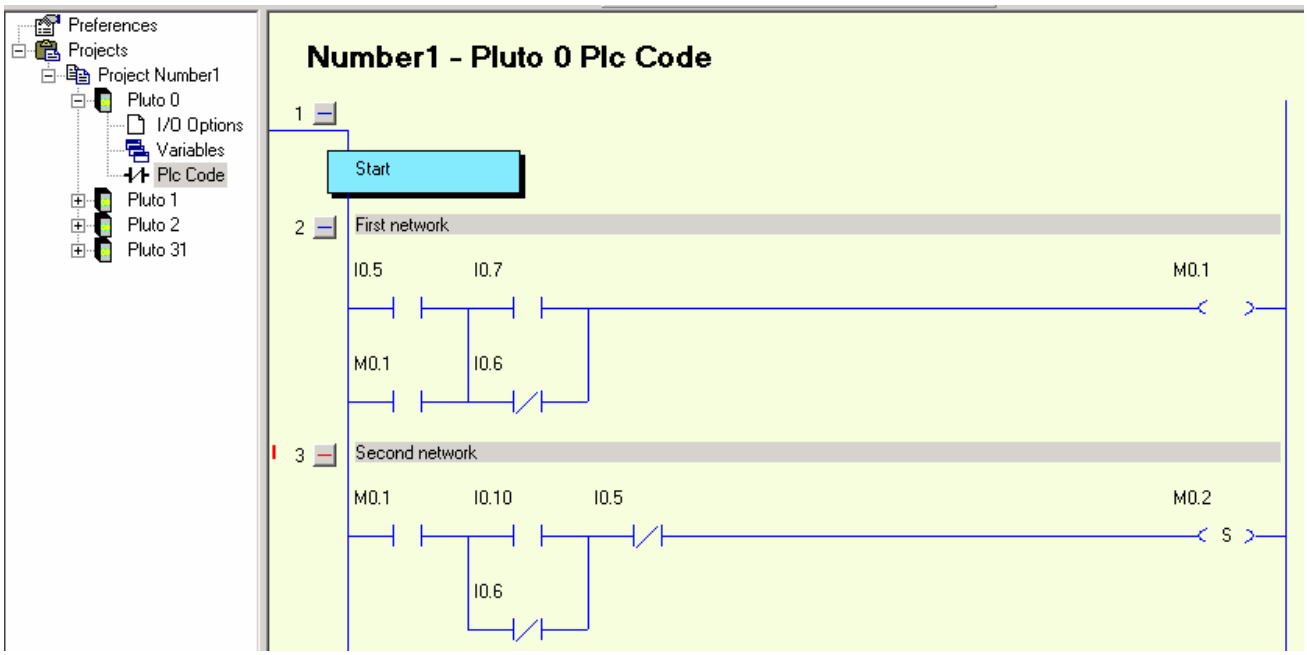


Local Variables:

- 'Local Outputs' Q_.10 …Q_.17, (Non fail safe outputs)
- 'Memories' M_.0 … M_.599
- 'Registers' R_.0 …R_.149
- 'System Memories' SM_.0 … SM_.199
- 'System Registers' SR_.0 …SR_.99

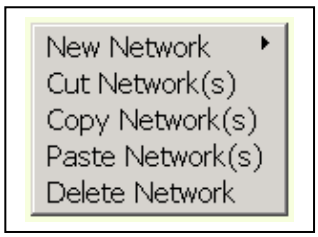
8 Ladder logic programming

By a mouse click on ‘PLC Code’ in the tree menu the page for ladder logic programming is shown.



The ladder logic program is built up with networks, also called rungs. These are numbered on the left side.

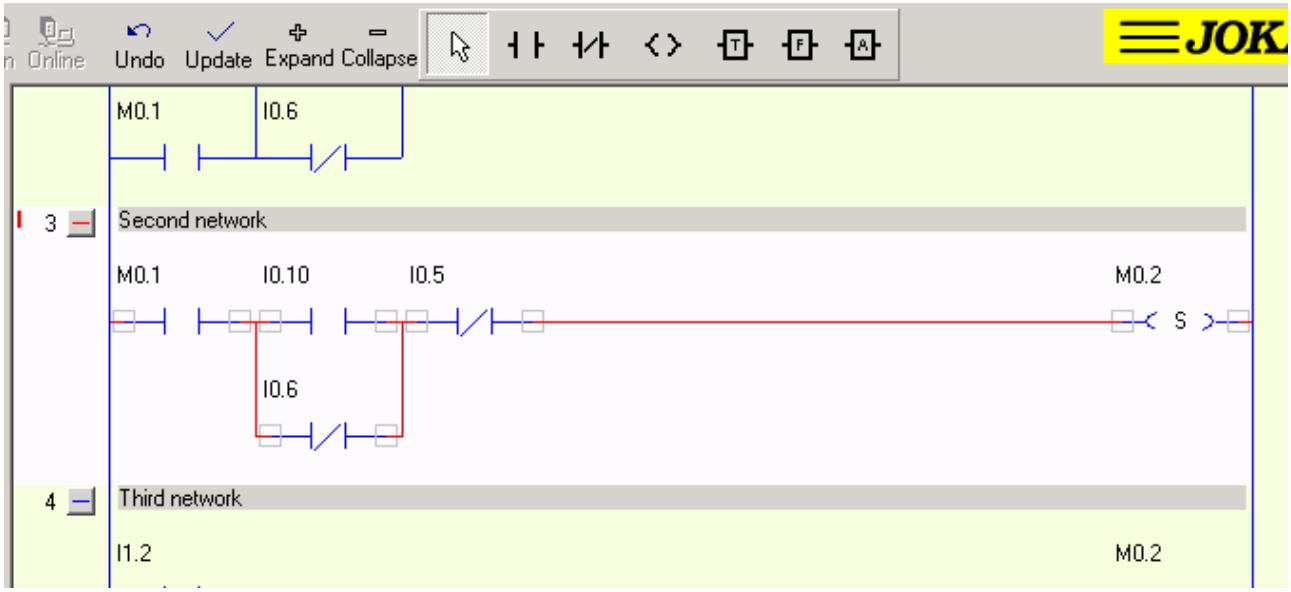
By a right mouse click in a network the following dialog box appears. The options Cut, Copy, Paste and Delete Networks operate as most other windows programs and lead to new dialog boxes.



By selecting ‘New Network’ a new network is opened and inserted below the network were the mouse click is carried out.

Edit mode

Edit mode can be entered in two ways, either by selecting ‘New Network’ as described above or by a left mouse click on an existing network. Only one network can be edited at a time.



A network in Edit mode is high-lighted, the lines between the components are red and hit boxes are shown. The tick boxes show where it is possible to connect a line. In edit mode it is possible to drag around, insert, disconnect, delete, etc. lines and ladder components.

Operations in edit mode:

Draw a line: Do a left mouse click (and release the button) in a ‘hit box’ for a component. The ‘hit boxes’ show the connection points. Move the cursor to the component where the end of the line is to be connected and fix it with a left click.

Change a line: By clicking the mouse on a line outside the ‘hit boxes’, the line is grabbed. It is now possible to:
- Stretch it to a third point and fix it with a left mouse click.
- Go to one of the ‘hit boxes’ and disconnect it with a left mouse click. When the line is detached it can be fixed to another component or deleted with a mouse click outside a ‘hit box’.
- Make a right mouse click and a dialog box ‘Delete line’ is shown.
- Un-grab it with a new left mouse click.

Change components properties: A double left mouse click on a component leads to a dialog box for changing Variable name, NO, NC, Pulse function etc.

Change components: By a right mouse click on a component a dialog box with three options is shown.
- ‘Components properties..’ for giving or changing the name or function.
- ‘Disconnect component’ for deleting all connections to the component.
- ‘Delete component’ for deletion of the component.

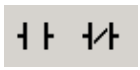
Moving components: Press and keep left mouse button down on a component and drag it. Release the mouse button at the new place required.

Tool bar

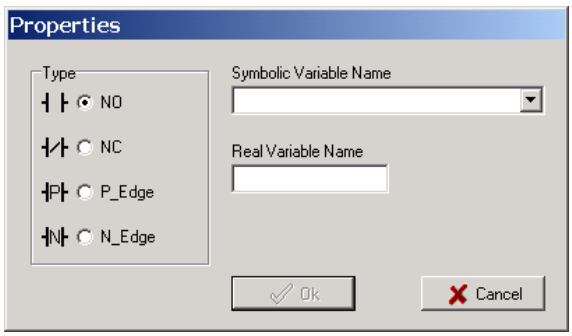
The tool bar is shown in edit mode and is used for the insertion of ladder components

To insert a component, click on the corresponding symbol. The cursor then takes the form of the symbol. Place it where you want to have it in the network, fix with a left mouse click and fill in the properties.

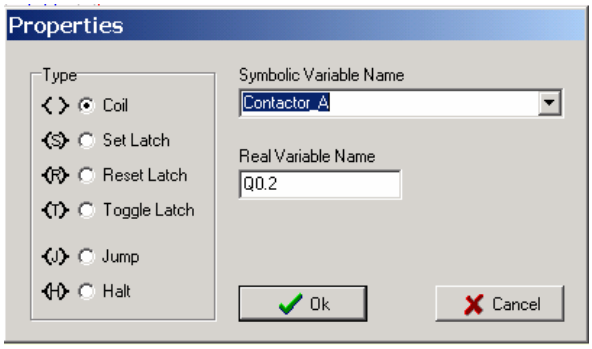
Tool bar components:



Standard ladder contact components. (Leads to the dialog box below.)

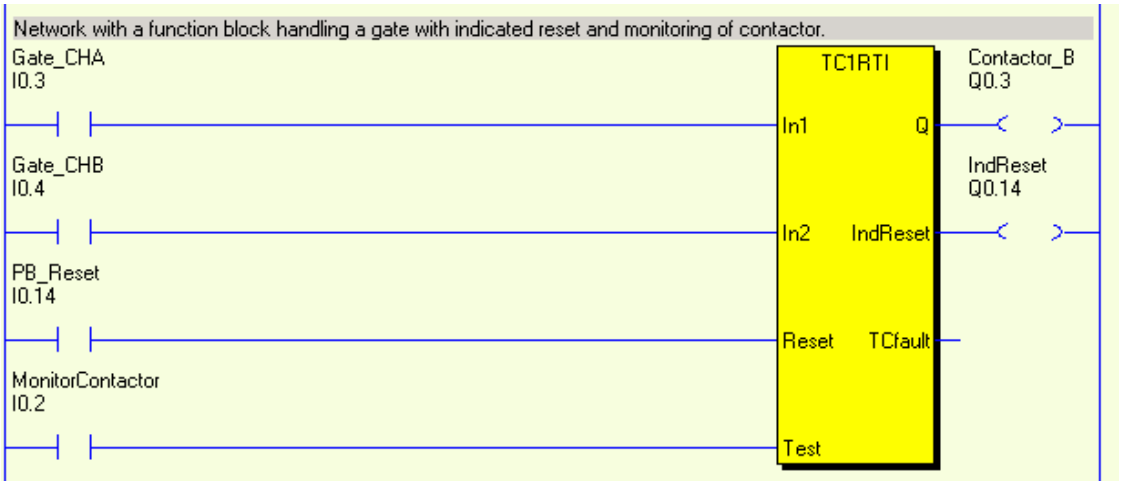


Standard ladder output components.
Leads to the dialog box below.



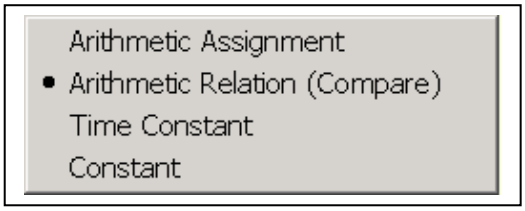
Timers
Leads to a dialog box for selection of two different types of timers.

Function blocks.
By a clicking on 'F' a list with available function blocks appears. This list is however dependant on if a function block library is selected. See 'Selection of function block library'. The function blocks are described in separate documentation.



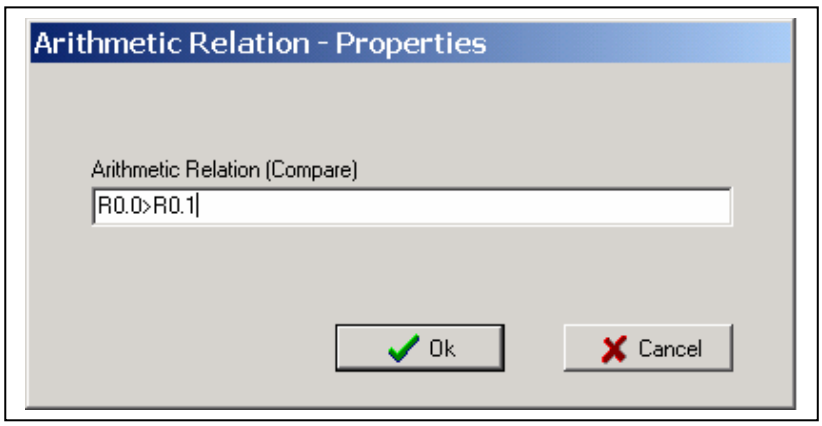
Example of network with function block

Arithmetic functions and constants.
A click on the symbol leads to the following drop down list.

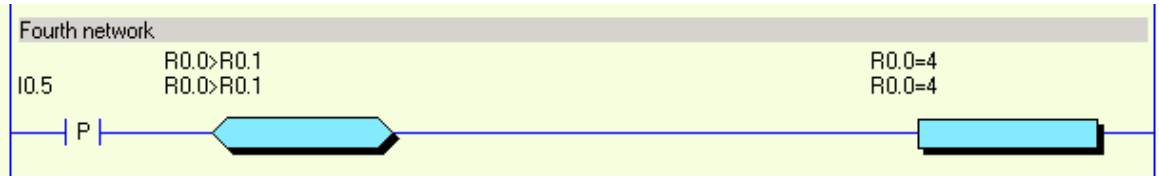


- 'Arithmetic Assignment' assigns a value to a register. This assignment can contain a mathematic operation (+, -) as well as a direct assignment of a value.
- 'Arithmetic Relation' makes a comparison of a register.
- 'Time constant' is used for function blocks requiring a timer value as input.
- 'Constant' is used for function blocks requiring a constant value as input.

By selection of one of these options a new dialog box is shown where the value, comparison etc. is written in text form. (See also Part2 Programming manual)



In the ladder diagram the arithmetic function looks as follows.



By positive edge on input I0.5 and register R0.0 is greater than R0.1, R0.0 is set to 4.

Update / Undo

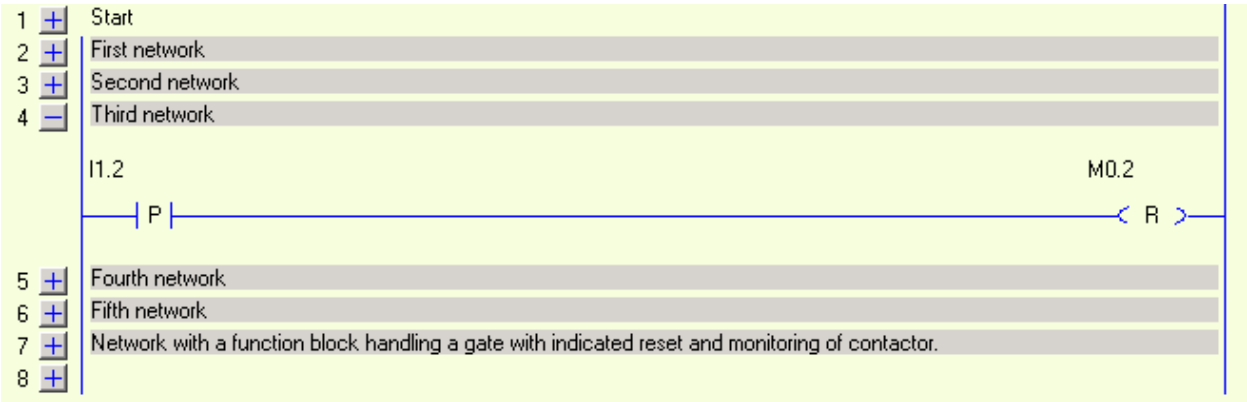
To exit edit mode either the 'Update' or 'Undo' buttons can be used. Update confirms the changes and Undo restores everything in the edited network as it was before entering it.

Instead of 'Update' button:
- 'F3' key or
- 'Esc' followed by answering Yes in a dialog box, can be used

Instead of 'Undo' button:
- 'F2' key or
- 'Esc' followed by answering No in a dialog box, can be used

Expand / Collapse networks

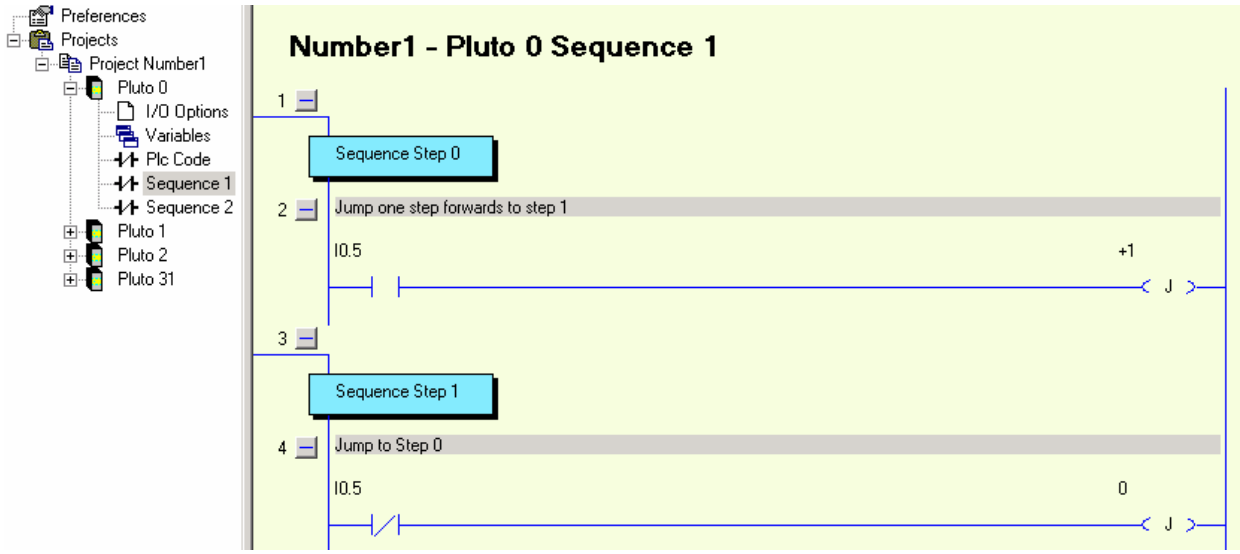
The ladder diagram can be controlled to be in either in expanded or collapsed form. In collapsed form only the comment for a network is shown and the ladder logic is not visible. The buttons in the tool bar controls all networks in the whole ladder diagram.
To control each network separately there are '+' and '-' buttons on the left side of each network which can be used.



Ladder diagram with collapsed and expanded networks

Sequences

In addition to the ordinary PLC code it is possible to have 9 Sequences with a maximum of 254 steps in each sequence. See Programming manual.
To open a new sequence:
Right click on the Pluto symbol in the tree menu → Select 'New Sequence' → Enter a sequence number 1-9 in the next dialog box.

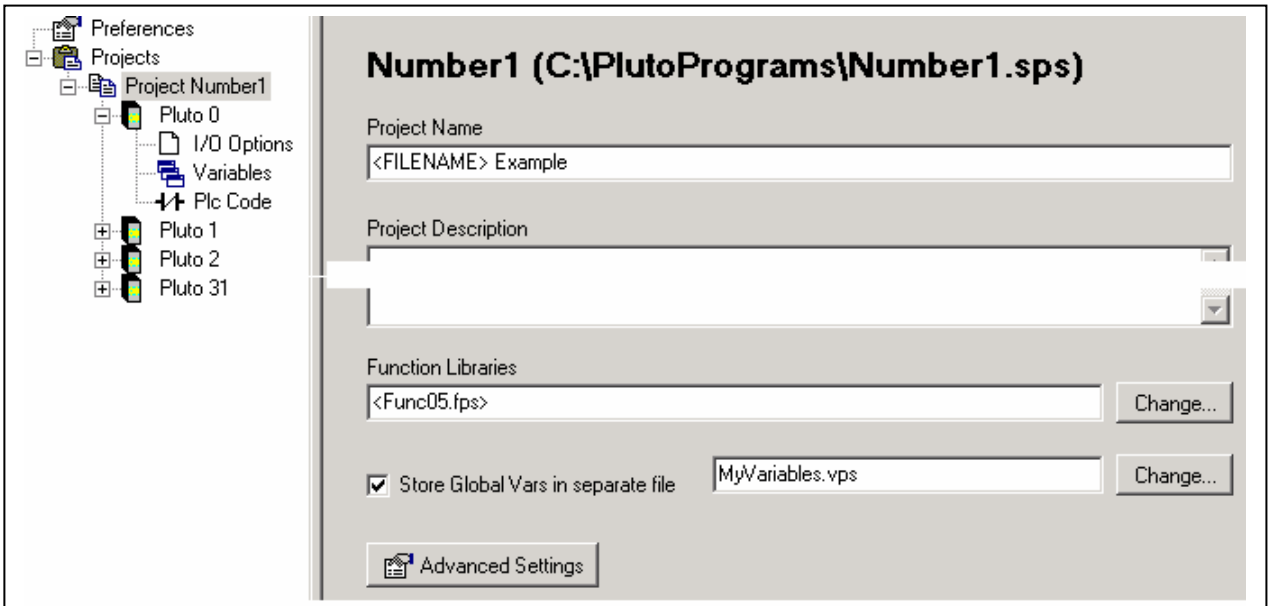


A very simple sequence.

9 Project setup

Include files

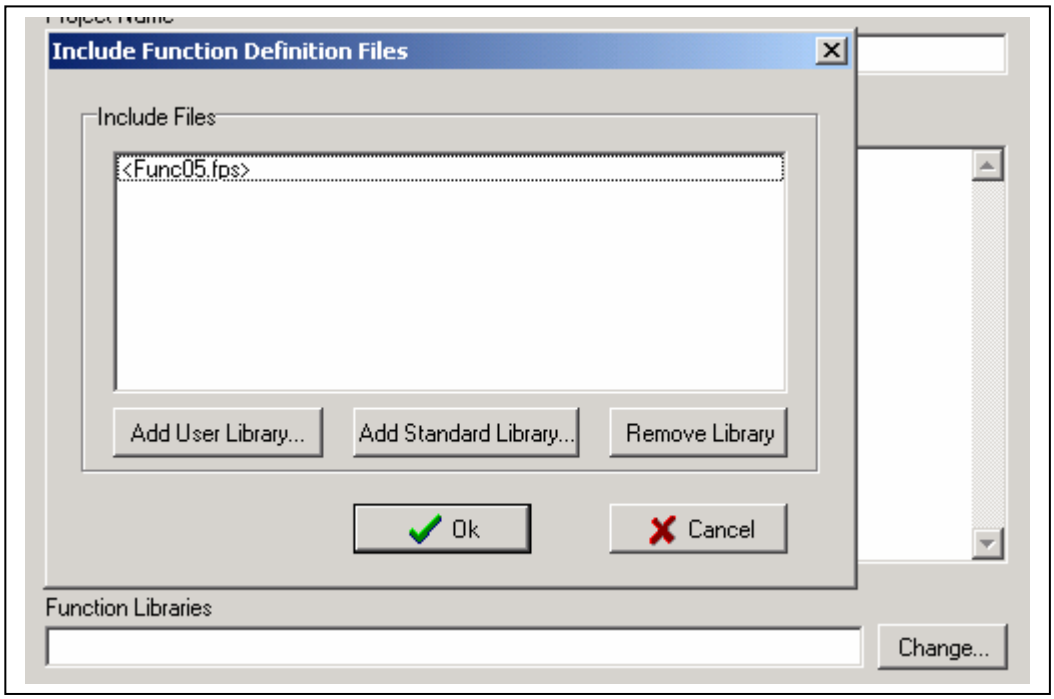
On the page Project [*Name*] there are two settings to make, selection of library file for function blocks and the possibility for storing global variables in a separate file.



Function library Func05.fps is selected and global variables are stored in MyVariables.vps

Function libraries

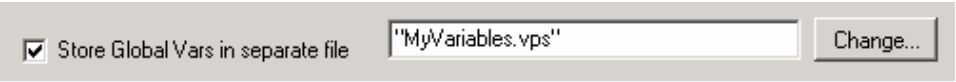
The Pluto system offers the possibility for using pre-programmed function blocks / macros for different safety functions and safety devices. These function blocks are stored in separate library files with file extension .fps. Standard libraries are provided by Jokab but it is also possible to make user specific libraries. Several library files can be loaded in one project.



By a first mouse click on ‘Function libraries’ ‘Change’ a dialog box with three options appears.

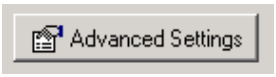
- ‘Add standard Library’: Pluto Manager looks for Jokab made files at ‘..\PlutoManager\Library’ where they are normally stored by the installation program.
- ‘Add User Library’: Pluto Manager looks for the files in the directory where the project files are stored. User libraries are files with user specific function blocks.
  For making a function block see special manual.
- ‘Remove Library’ is used for deleting a file in the list.

Variables in a separate file



In some cases there can be the need for storing the global variable names in a separate file which makes it possible to use them in other programs. One case can be if a network is divided into several programs. For example a network with 20 Pluto units divided in two program files (.sps), one for Pluto 0…9 and one for Pluto 10-19. Since they can use each others global variables it is practical/logical that they use the same names and that these names are taken from the same source. By creation of the first program the global variables can be named and stored in a separate file and when it is time to create the second program the file can be imported and used.

Advanced settings



The advanced settings on this page have influence on the response times of the system and require an in depth knowledge to understand and to modify.

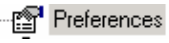
For further information see System Maintenance manual.

10 Compilation

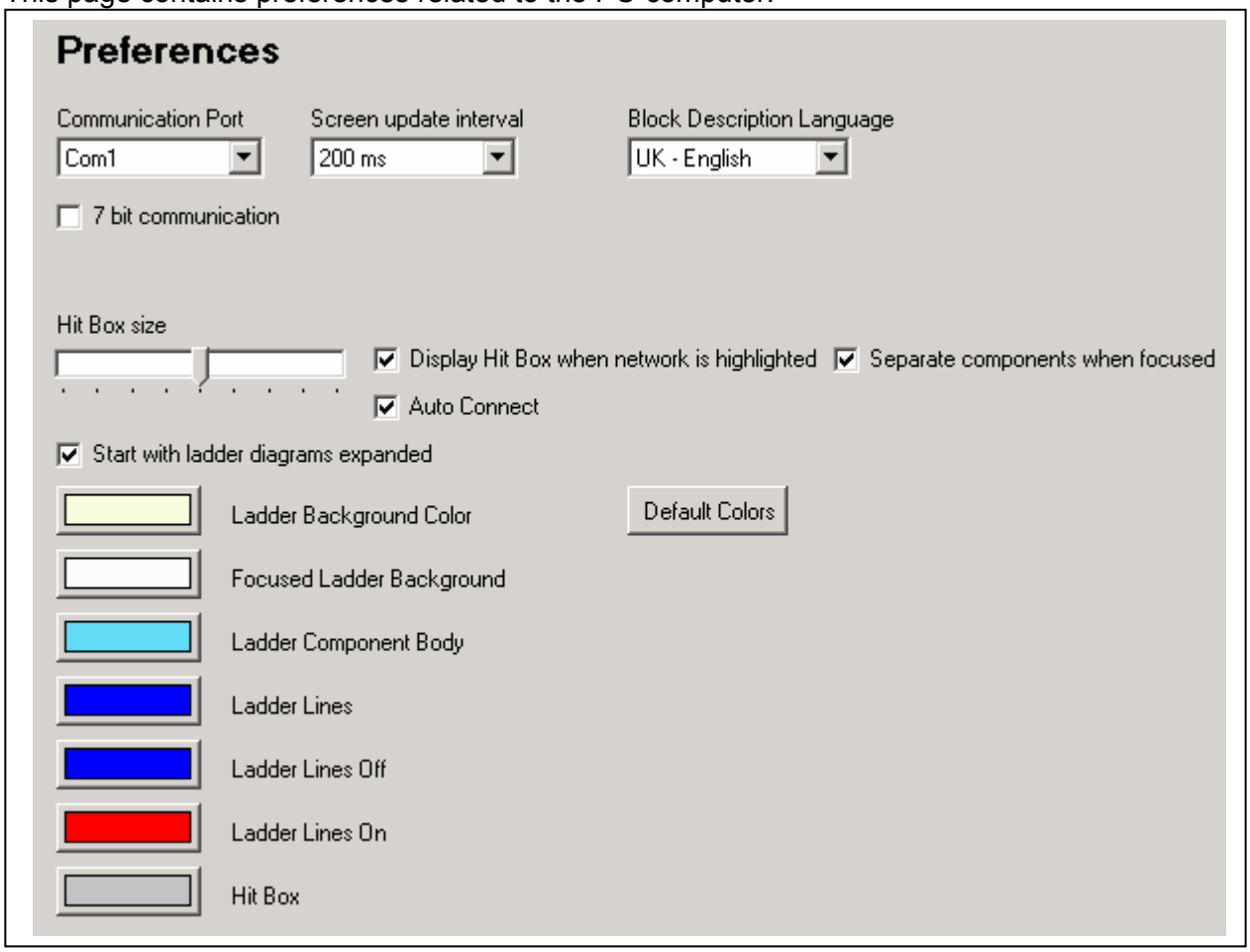
Pluto Manager saves the program in a file with extension ‘.sps’, but this cannot be downloaded to a Pluto unit before being compiled. The compiler checks the program code in the sps-file against syntax faults and produces a file in hex format (.hps), which can be downloaded.
By clicking on the ‘Comp’ button the compilation is started and a text window appears on the screen. At the end of the compilation the message ‘0 Error (s) detected’ …… Result=OK’ appears, if everything is passed. Pluto Manager prevents most syntax faults but not 100% and it can therefore happen that the compiler gives fault messages.

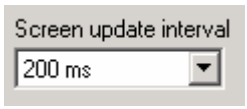
Note. Pluto Manager and the compiler just checks for syntax faults, when the code is not corresponding with rules of the language. Logic faults, like an emergency stop that controls an incorrect output cannot be detected by the software tools. Programs must therefore be reviewed and safety applications carefully tested before the use.

11 General Preferences

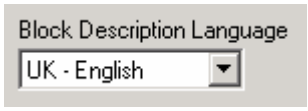


This page contains preferences related to the PC-computer.





Update interval in on-line mode. Lower update interval makes the computer slower.



The function blocks (described under 9) have a description, visible by a bubble click on them in edit mode. The language of this description can be selected here.

The size of Hit Boxes and if they shall be shown in the ladder diagram can be set.

When Auto Connect is ticked, ladder components are automatically connected when they are inserted on a line.

In Edit mode the ladder components are separated from each other.

As default the ladder diagrams are opened in expanded from.

Colours
The colours in Pluto Manager can be changed by the user.

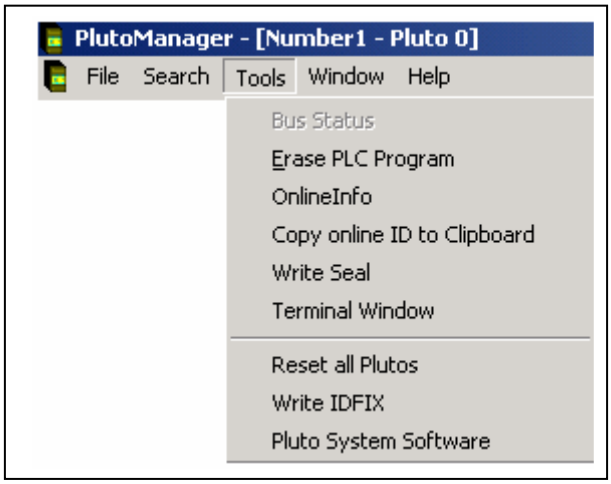
12 On-line operations

Communication

Communication with the system requires an infrared adapter connected to one of the COM ports on the PC. Go to the page 'Preferences' and select COM port.

'Tools' menu

Most of the on line functions can be found under 'Tools' menu



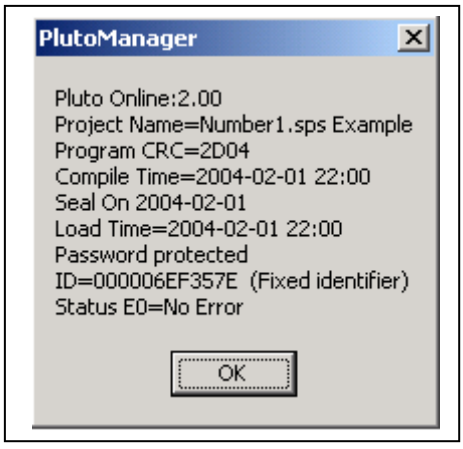
Erase PLC Program / Change of password

Under 'Tools' → 'Erase PLC program' it is possible to erase the PLC program.
This function can also be used in order to change pass word. By down load of PLC program into an erased Pluto the user can select a new password.

On line info

Under 'Tools' → 'Online Info' it is possible to read data in 'real time' from a Pluto unit.

For the normal user, Project Name and Compile time is the most important data.
To go on-line, the Project name must match with the Project name of the opened project in Pluto Manager.

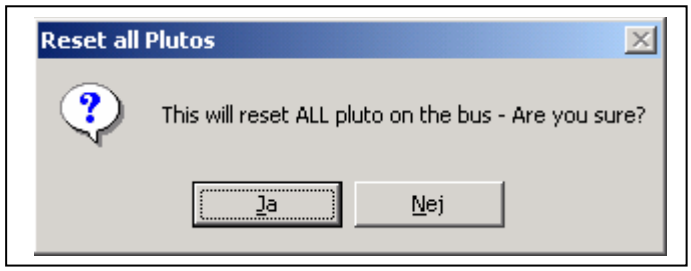


Copy online ID to Clipboard

The identifier circuit 'IDFIX' is read and automatically copied to the clipboard. By a Ctrl+V it can then be pasted into the field 'Identifier Number'

Reset all Plutos

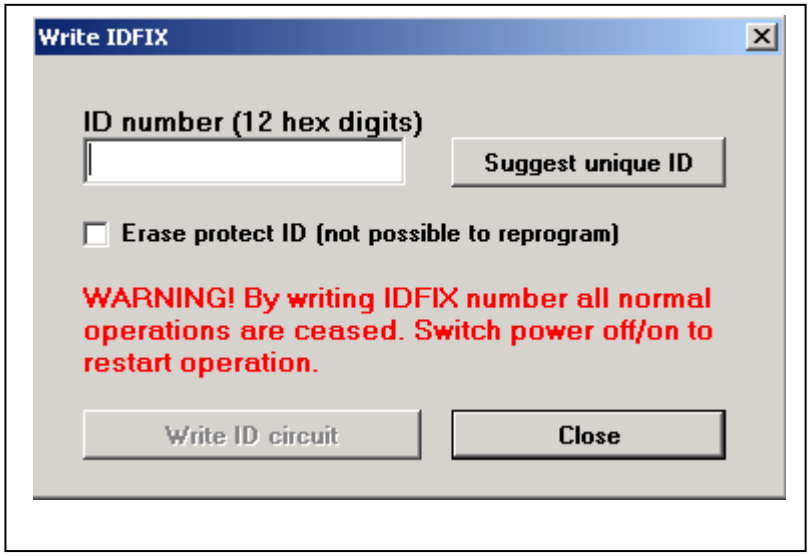
The command will Reset / (reboot) all units connected to the bus. The Reset has the same function as power off/on and can be necessary in situations as after change of baudrate or reset of some faults.



Write IDFIX-RW circuit

Function for programmable identifier circuits 'IDFIX-RW'.
It is possible to put in the number manually for example in order to make a copy of an existing, or let the system suggest a number.
By selection of 'Erase protected ID' the circuit can never be changed again.

Note that after writing ID the Pluto must be reset (power off/on) to enter to normal operation again.



Pluto System Software

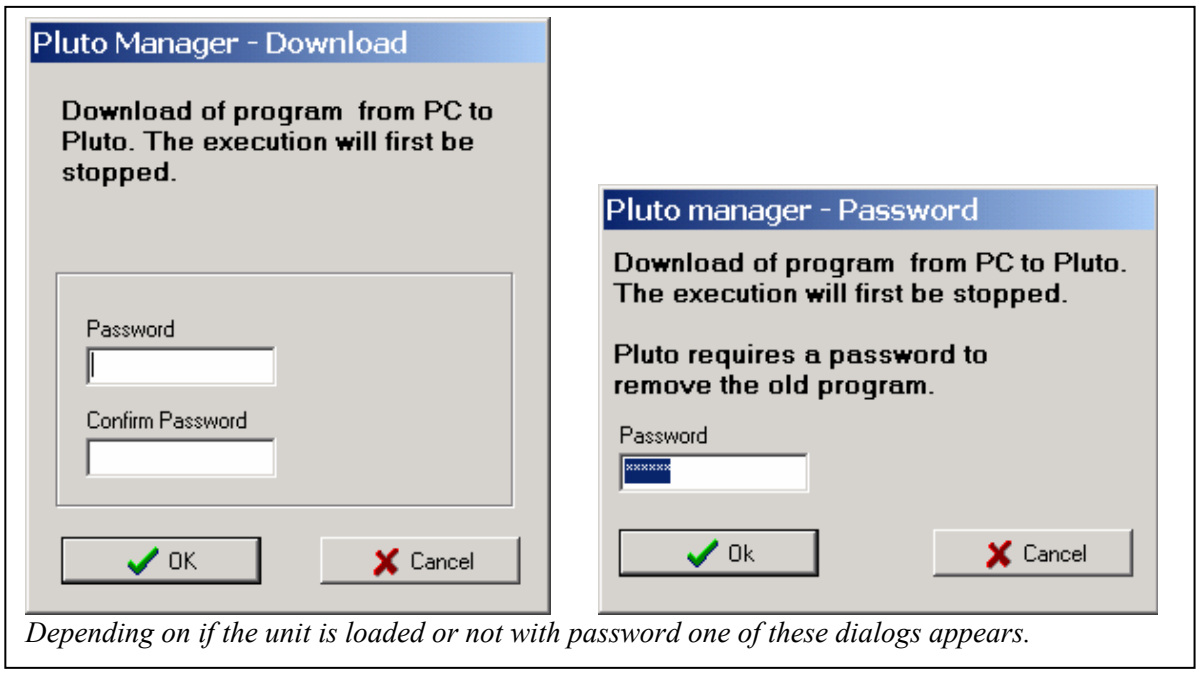
This function is to be used by upgrade of the system software (operating system). To use this function the Pluto must be started in a special mode and the user must have two files with extension '.mhx' available. Instructions in detail follow together with the upgrade files.

Program down load

To down load a program from a PC to a Pluto unit press the 'Down' button in the tool bar.
Note that before a program can be downloaded it must be compiled. A fault message will tell if not. See Compilation.

After pressing the Down button, dialog boxes requiring passwords appear.
The password must be 4-6 characters long.
If not fault messages appear:

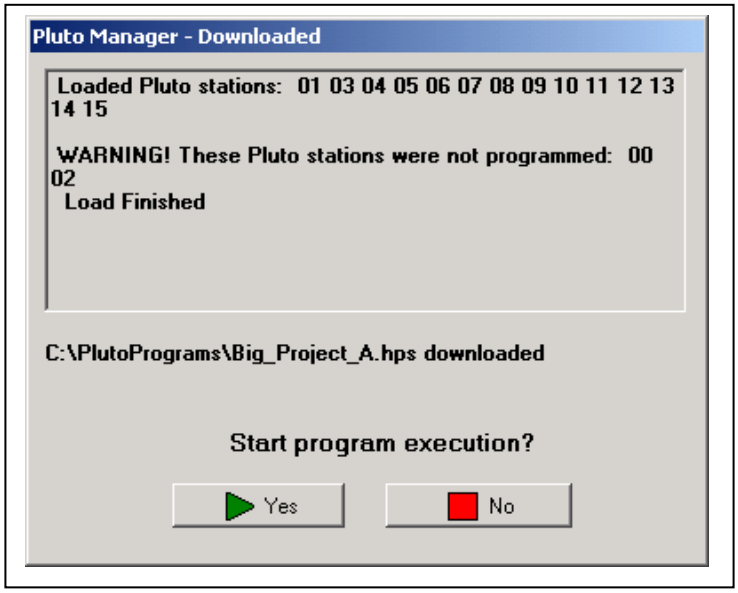
'Couldn't establish connection…' - No connection at all.
'Connection time out' - The communication is interrupted



Depending on if the unit is loaded or not with password one of these dialogs appears.

If everything works the message appears that the file is downloaded together with a selection if execution of program shall be started or not. If 'No' is selected it is possible to start execution by pressing the Online button and then Start.

If the program project is for several Pluto stations and all are not connected to the bus a warning is given.



Insert of Pluto unit in existing project afterwards.

When Pluto units are loaded with program for several units they check each other so they have exactly the same version of the program code. By mismatch they do not accept each others I/Os.

If a unit belonging to a program project is connected to the bus afterwards the following situations can appear depending on what PLC program it is loaded with:

Alt. 1 - The new Pluto is empty of PLC program (message code E20) and is fitted with correct IDFIX circuit.

The new Pluto can be loaded with program by a new download from the PC to any of the Pluto:s of the same program project.
It can also be programmed by using self programming without PC. By pressing the 'K' button in the Pluto front panel in 2 seconds the display flashes 'L' which indicates that it is ready for self programming. By another activation of the 'K' button the program load is started indicated by a steady 'L' on the display.

Alt. 2 – The new Pluto is fitted with correct ID-FIX circuit but loaded with wrong version of the program.

By connection all units of the project will give error code E27 because they detect units belonging to their own program project but with mismatching program as the new unit has wrong version. The units will run the PLC program but will not accept I/Os in Pluto units with mismatching program.
By a new download to any of the units in the project all of them will be updated with the same version.

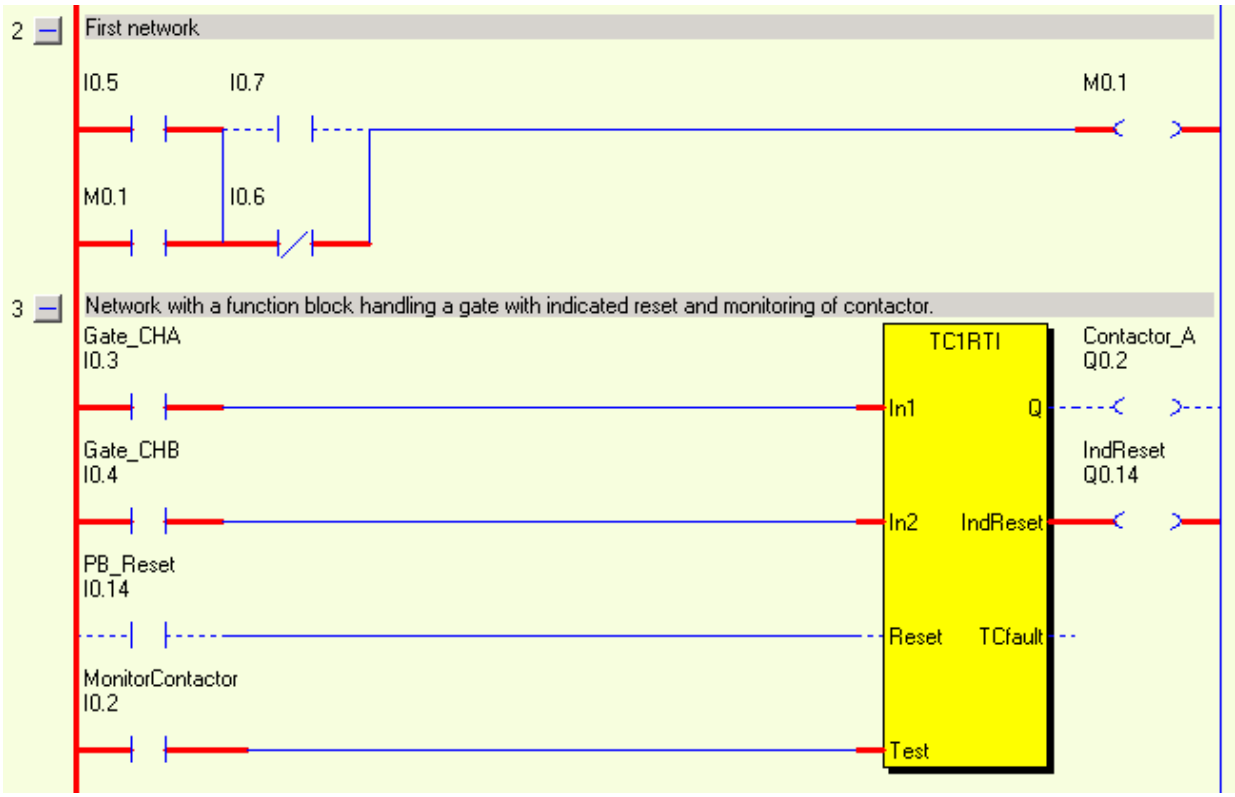
Change of baud rate, error code E26.

A Pluto unit can not change baud rate during operation. If then a unit is loaded with a program with new baud rate it will continue with the old baud rate and indicate E26. E26 indicates that a unit runs with another baud rate than it is programmed for.
By reboot either by power off/on or via Pluto Manager 'Tools' → 'Reset all Plutos' the unit can change baud rate.

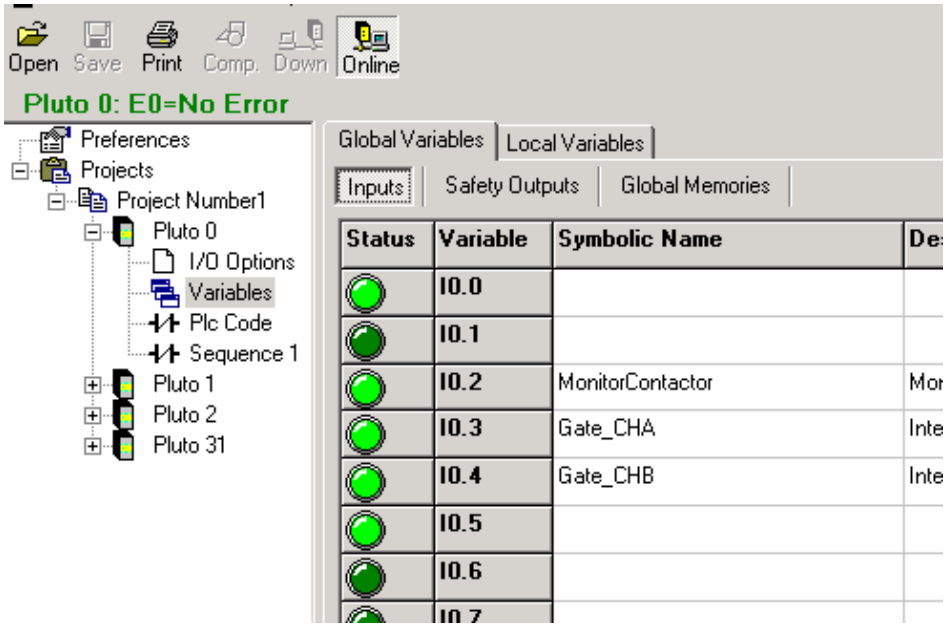
Also if an empty (E20) stand alone unit is loaded with program it will indicate E26 and has to be rebooted in order to start with programmed baud rate.

On-line

Using the button in the tool bar, the on-line mode can be switched on and off.
In on-line mode the I/O status can be monitored either by opening a variable page or a ladder diagram.



In ladder diagram, all 'true' components are marked with red lines in online mode.



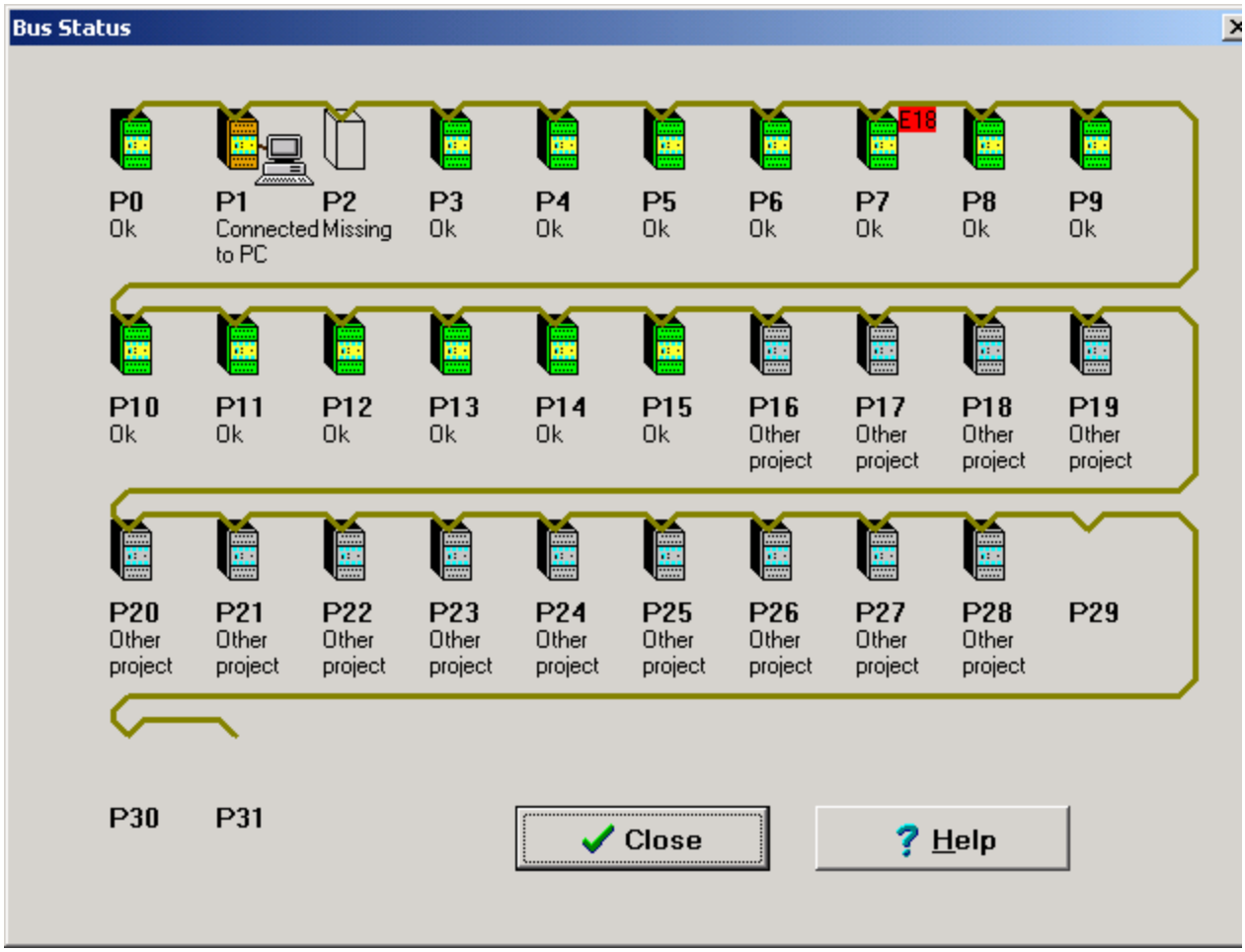
In the windows for variables, a column with status indicators is viewed in online mode.
In the tool bar, the error codes for the Pluto unit is showed with green text.

Start and stop of program execution.

In on-line mode the program execution can be controlled.

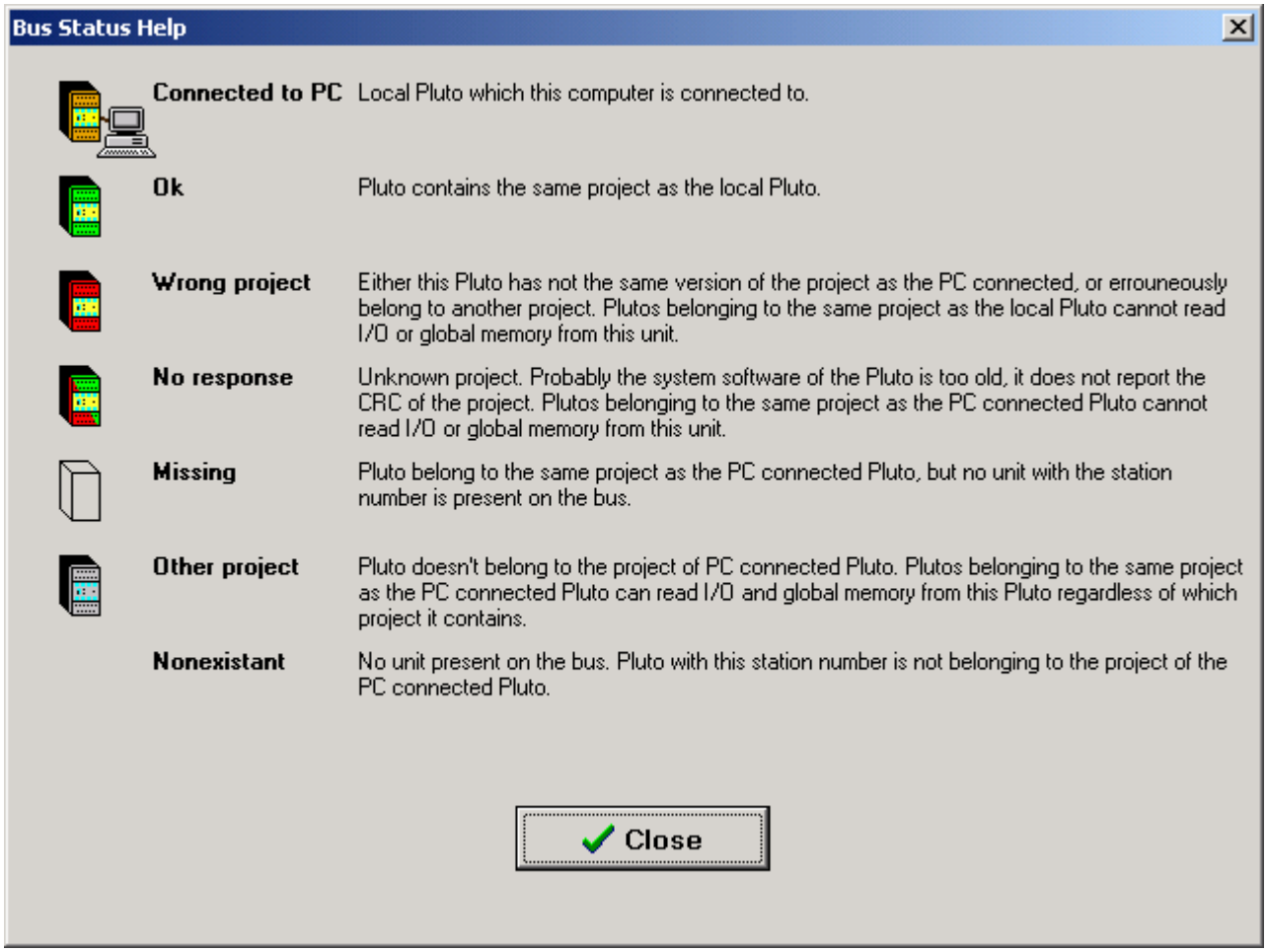
Bus Status

In on-line mode it is possible to get an overview of the Pluto units connected to the bus via selection of 'Tools' → 'Bus Status'



The picture shows a bus with 29 Pluto:s divided in two or more project files.
One project contains Pluto nodes 0…15 meaning that these units are loaded with the same program. Pluto no:1 is connected to the PC. Pluto no:2 is in the project but has no connection with the bus. Pluto no:7 has an error.
Pluto no: 16…28 are connected to the buss but are not in the same program project as the Pluto connected to the PC.

Explanation of Bus Status



Under Help button the following picture with explanation is displayed.

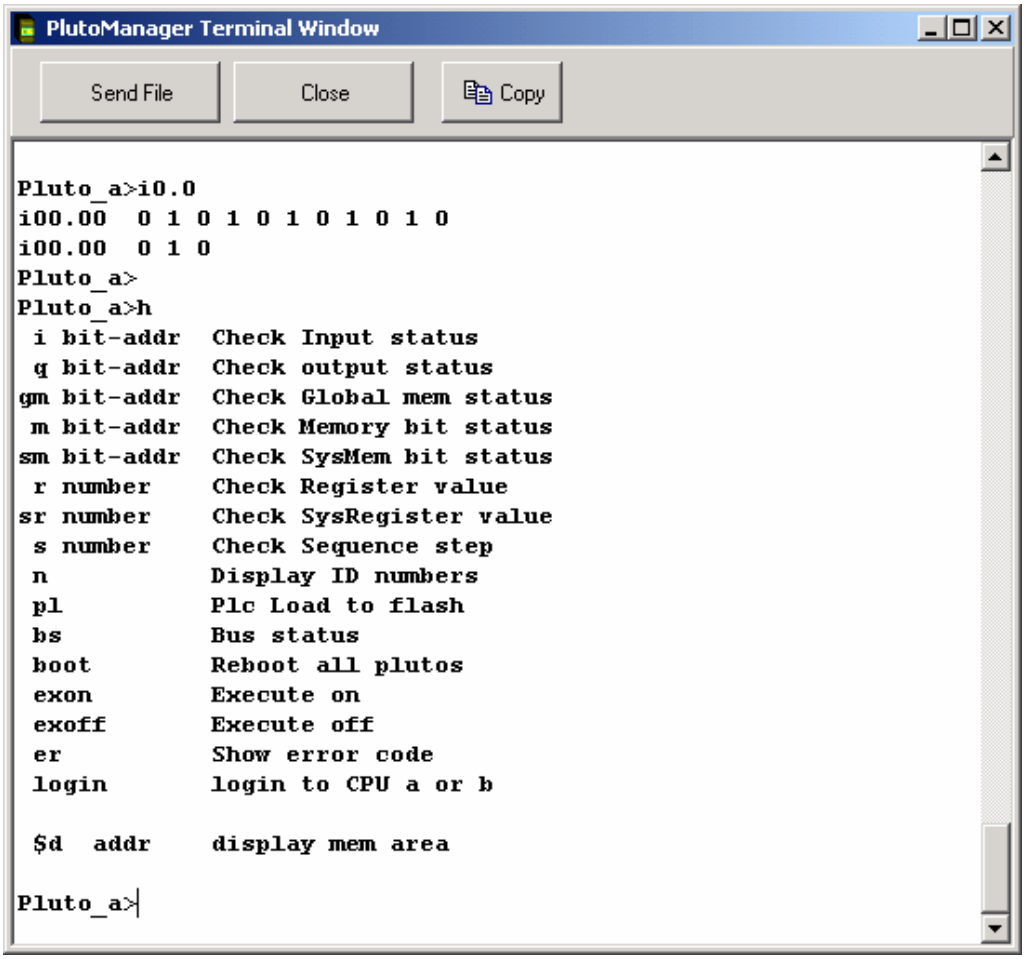
Terminal window

Another way to communicate with a Pluto unit, is to open a terminal window.

'Tools' → 'Terminal Window'

In this mode the PC is just a terminal. Everything typed on the keyboard is sent to the Pluto unit and everything written in the terminal window is written by the Pluto unit.
A lot of things can be monitored via the terminal like I/O:s, compile date, program name etc.. It is also possible to load new programs by typing 'pl' followed by use the 'Send File'-button.
By typing 'h' (help) available commands are listed.

Instead of this terminal a standard terminal program can be used such as HyperTerm in Windows.



Example of terminal communication. Pluto_a> is prompt. Input i0.0 is monitored, all changes of state are logged. Next command is 'h' which lists all available commands.

Seal

In the dialog box ‘On line info’ (see above) there is a text line telling ‘Seal On’ or ‘Seal Off’.
After download of a program the text ‘Seal off’ is shown. This indicates that the program is changed but not sealed.
The purpose of the seal is just to give an indication that the program is changed and has no influence on the function.

Depending on the licence code, Pluto Manager can be set up with three different alternatives with or without the possibility to write seal.
Alternative 1: Seal function is not available for the user.
Alternative 2: Seal can be loaded separately after program load.
Alternative 3: Seal is automatically loaded by program load.
A user company can then make a system there some people are authorized to review programs and confirm by downloading a seals.

To write a seal: ‘Tool’ → ‘Write Seal’ → A message ‘Seal written’ indicates if success.

Part 2
Programming language

1 Bit-instructions

1.1 Addressing of bit-operands

In PLUTO programming language I/O and memories are addressed as [*I/O-type*][*unit no*].[*I/O no*].
At most 32 PLUTO-units, numbered 0 – 31, can be interconnected via the Bus.

I/O type:	I/O designation Pluto 0	I/O designation Pluto 1	……	I/O designation Pluto 31
Inputs	I0.0 – I0.17	I1.0 – I1.17	……	I31.0 – I31.17
Outputs	Q0.0 – Q0.17	Q1.0 – Q1.17	……	Q31.0 – Q31.17
Memories	M0.0-M0.599	M1.0-M1.599	……	M31.0-M31.599
Global memories	GM0.0-GM0.11	GM1.0-GM1.11	……	GM31.0-GM31.11

Example:
Q10.1 ⇔ Addressing of output 1 on PLUTO no. 10

Following alternatives are also accepted: Q10.01 or Q10.0001 or Q10.1

1.2 Boolean instructions

PLUTO programming language follows the rules for ladder programming of IEC 1131-3 when programming with Pluto Manager.

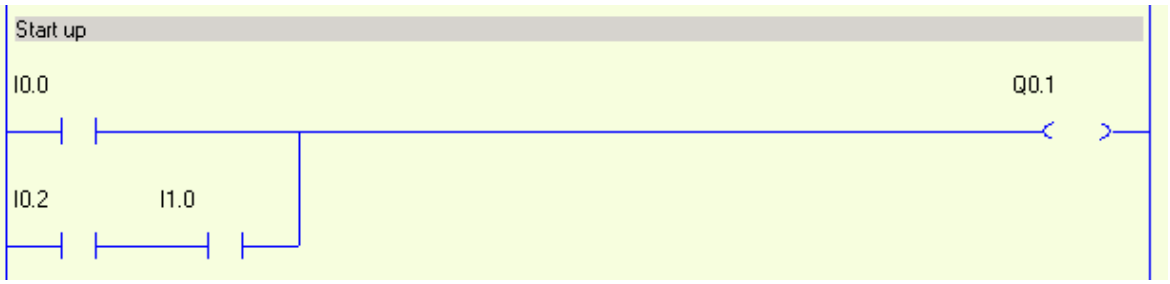
By programming in text form using an text editor the programming language follows the Boolean laws and utilises AND, OR, NOT and EXECUTION -commands.

Program syntax in text form:

Instruction:	Program syntax:
AND	*
OR	+
NOT	/
EXECUTION	=

Example:

In ladder form:



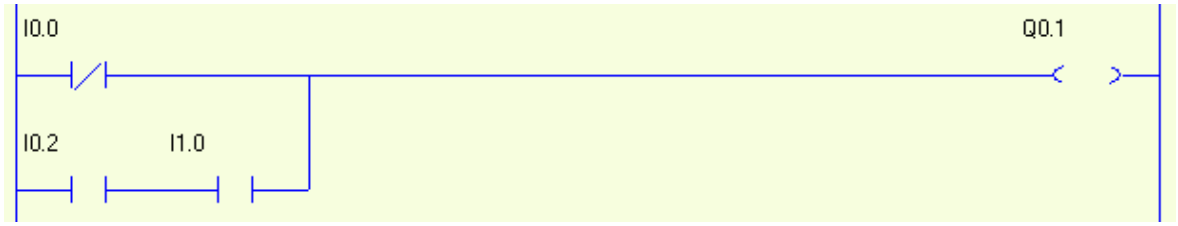
Equivalent text form:

```
Q0.1 = I0.0 + I0.2 * I1.0              ; Start up
```
; (semicolon) defines start of program comments.

Explanation: Output Q0.0 is on when input I0.0 or both of I0.2 and I1.0 is on (‘1’).

Example with negation:

In ladder form:



Equivalent text form:

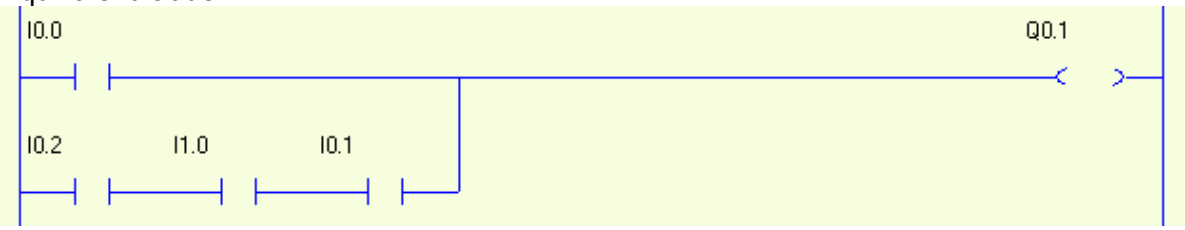
```
Q0.1 = /I0.0 + I0.2*I1.0
```

According to the boolean laws AND-instructions (*) are executed before OR-instructions (+). By using brackets the instruction order can be changed.

Examples:

```
Q0.1 = I0.0 + I0.2*I1.0*I0.1
```

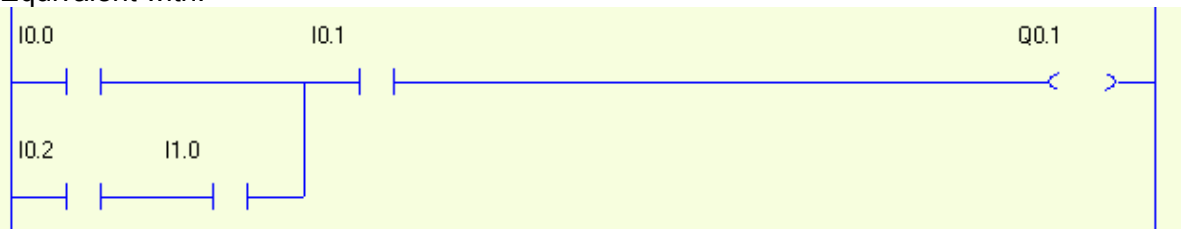
Equivalent ladder:



Example with use of brakets

```
Q0.1 = (I0.0 + I0.2*I1.0)*I0.1
```

Equivalent with:

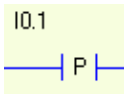


NOTE: In text form the use of spaces have no influence.

1.3 Edge detection

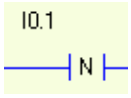
Edge detection can be used on single operands. The EDGE-function enables detection of both positive and negative edges. Relevant program syntax follows in the table below:

Positive edge:



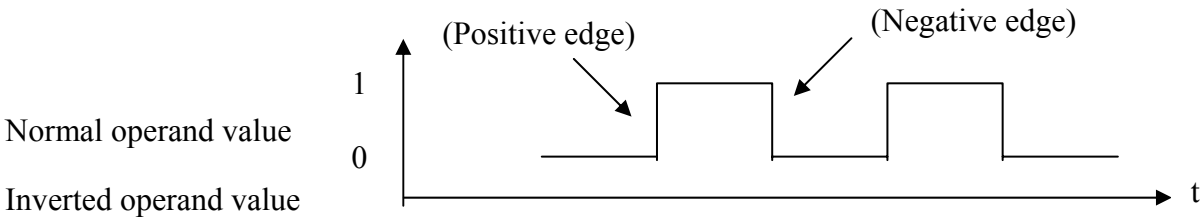
⇔ P(I0.1)

Negative edge:

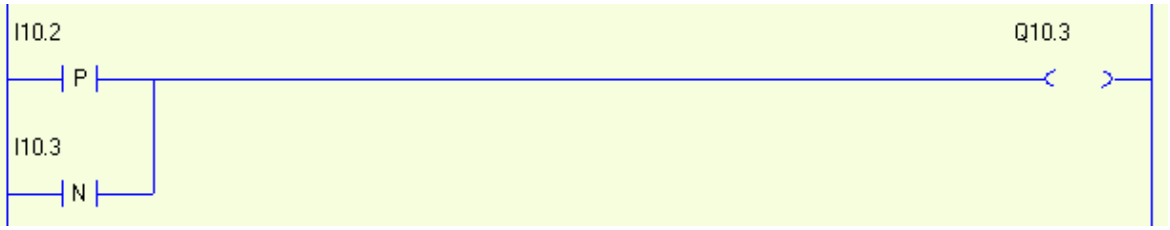


⇔ P(/I0.1)

Function: When an edge is detected a logical “1” is held during a complete program scan cycle.



Example:



`Q10.3 = P(I10.2) * P(/I10.3)` ⇔ Output 3 on PLUTO no. 10 is set HIGH when positive edge is detected on input 2 on PLUTO no. 10

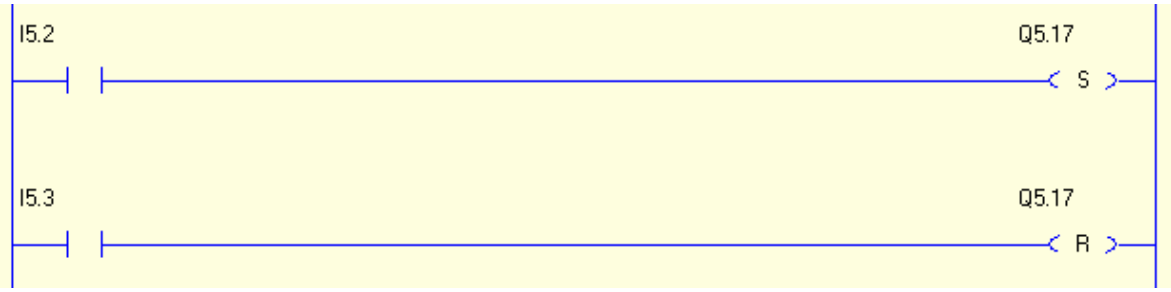
1.4 Latch function

By use of the Latch function an output or a memory-cell is given a self-hold/memory function.

Latch function:	Program syntax:
SET/Latch on	S(Q0.1)
RESET/Latch off	R(Q0.1)

When an output/memory-cell is set HIGH by the SET-instruction, the output/memory-cell will remain HIGH although the previous condition-statement no longer is TRUE. The output/memory-cell can be set LOW by use of the RESET-instruction.

Example:



Equivalent text form:

```
S(Q5.17) = I5.2
R(Q5.17) = I5.3
```

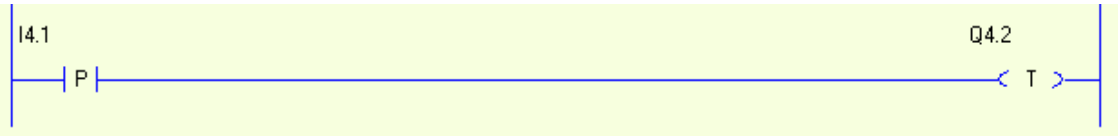
Function: Output 17 on PLUTO no. 5 is set HIGH when input 2 on PLUTO no. 5 is set HIGH. The output remains HIGH until it is RESET by setting input 3 on PLUTO no. 5 HIGH.

1.5 Toggle function

The Toggle function toggles the state of a operand (Q, M or GM).

Toggle function:	Program syntax:
Toggle state	T(Q0.1)

Example:



Equivalent text form:

```
T(Q4.2) = P(I4.1)
```

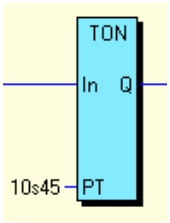
Function: Toggle of output 2 on PLUTO no. 4 changes state from 0 -> 1 or 1 -> 0 on positive edge of input 1 on PLUTO no. 4.

NOTE: In this example edge instruction is used to avoid that Q4.2 toggles more than once. Otherwise the output will toggle ON/OFF every PLC cycle.

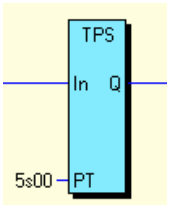
1.6 Timers

PLUTO has 50 timers that all can be used simultaneously in an active sequence steps (see sequences). The timers have a resolution of 10 ms and can be defined in the time-interval 0.01 – 655.35 s.

Timer:	Value	Program syntax:
50	0,01- 655,35 s.	**T**(*nn***S***nn*)



⇔ T(10s45)



⇔ /T(5s0)

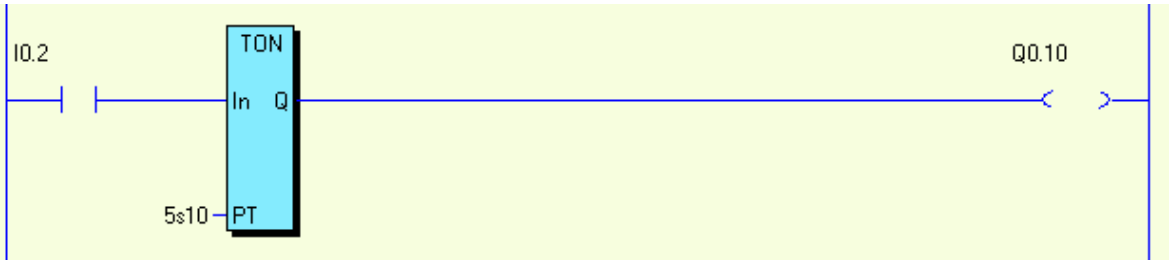
ON-delayed timer (10,45 sec.) Pulse timer (5,0 sec.)

The “s” -symbol corresponds to decimal sign

Function: There are two types of timers: ON-delayed and pulse timers.

ON-delayed timers (TON) starts when the boolean instructions on the left side of the timer instruction is TRUE. When the specified time is elapsed also the timer is TRUE (“1”).
Pulse timers (TPS) are activated in the same way but they are TRUE (“1”) from start and gets FALSE (“0”) when the time has elapsed.

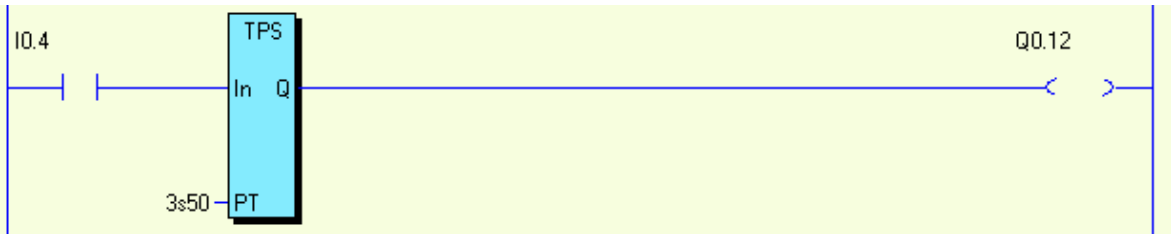
Exemple:



Equivalent text form: `Q0.10 = I0.2 * T(5s10)`

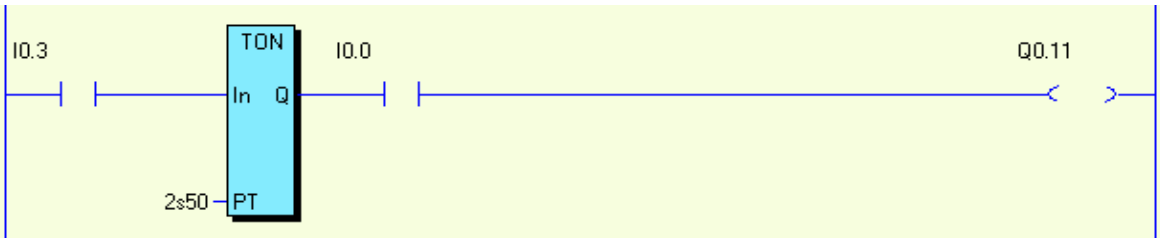
Function: When input I0.2 is set HIGH the timer with time-delay of 5,10s is activated. Output Q0.10 is set HIGH when the time is elapsed.

Example:



Equivalent text form: `Q0.12 = I0.4 * /T(2s5)`

Function: When input I0.4 is set HIGH the timer output and then output Q0.12 is immediately set.
After a delay of 3,5 s the timer switches output Q0.12 off.



Equivalent text form: `Q0.11 = I0.3 * T(2s5) * I0.0`

Function: When input I0.3 is set HIGH the timer is activated.
After a delay of 2,5 s and if input I0.0 is HIGH, output Q0.11 switches on.
Note that the expression after to the right of the timer (I0.0) has no influence on the timer.

2 Memories

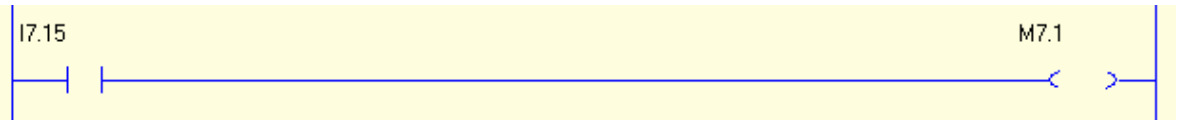
2.1 Local memories (M)

PLUTO has 600 memories free to use in the application program. These memories are local which means that they can just be used in the own Pluto unit. Example memory M0.10 can just be set and read in the application program in Pluto unit no: 0.

The memories are addressed as shown below:

Workmemory-cell:	Program syntax:
600	M_.0 – M_.599

Example:



Equivalent text form: `M7.1 = I7.15`

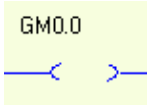
Function: Memory M7.1 is HIGH (1) when input I7.15 is HIGH.

NOTE: Although workmemory-cells are local within one PLUTO PLC, identity of the PLUTO-unit must be set as shown above.

2.2 Global memories (GM)

Global memories can be used in the same way as local memories but with the difference that they are transmitted on the bus and can be read by other Pluto units and used in their application programs as input condition.
One example for use of the global memories is to make it possible to have a memory which is the summary of a complex program function. Instead of making the same complex program function in many Pluto:s it can be programmed in just one unit and the result can be stored in a global memory which can be read by all Pluto:s on the bus.

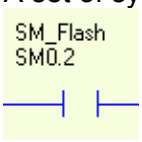
The global memories are addressed as shown below:



Global memory:	Program syntax:
0-11	GM_.0 – GM_.11

2.3 System memories (SM)

A set of system memories with different functions are available in PLUTO.

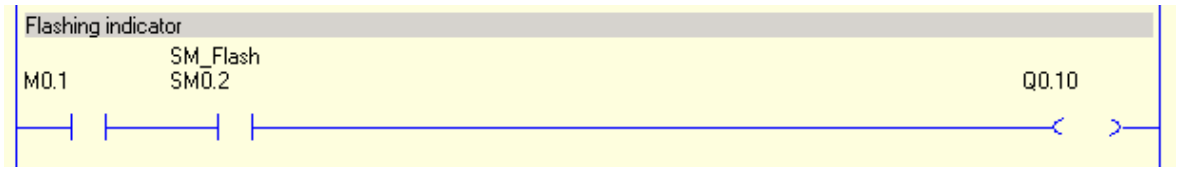


Syntax: SM[*unit*].[*no*]

I/O-address	Symbolic name	Function:	Type:
SM_.0	SM_StepNew	On at first scan in new sequence step.	R
SM_.1	SM_Ditto	Result of last logic operation.	R
SM_.2	SM_Flash	Flash: 0.4 / 0.6 sek. (on/off)	R
SM_.3	SM_1Hz	Pulse 1 Hz	R
SM_.4	SM_10Hz	Pulse 10 Hz	R
SM_.5	SM_FastFlash	Flash: 0.17 / 0,33 sek (on/off)	R
SM_.6	SM_DoubleFlash	Double flash: 0,11 / 0,2 /0,11 / 0,67 msec	R
SM_.9	SM_SysInit	On at first scan after power on	R
SM_.39	SM_Button	Button in front panel	R

(Type: R = Read, W = Write)

Example:



Equivalent text form: `Q0.10 = M0.1 * SM0.2 ;Flashing indicator`

Function: System memory SM0.2 is flashing with an on/off rate of 0.4/0.6 seconds.
If M0.1 is set, output Q0.10 flashes with the same rate as SM0.2.

3 Sequences

PLUTO has 9 sequence registers with 254 steps each available for use. The sequences operate in parallel and independent of each other.
In a sequence only the code in one step is executed. The transition from one step to another is conditional via jump-instructions. The result of the previous step is reset when the next step is entered. By start up of the system, sequence step 0 is automatically executed which means that a sequence must contain step 0.

3.1 Addressing

A sequence step starts with an instruction as below declaring sequence number and step number.

Sequence/Step:	Program syntax:
1-9/0-254	S*n*.1_00 – S*n*.9_254 (*n*=Pluto unit no.)

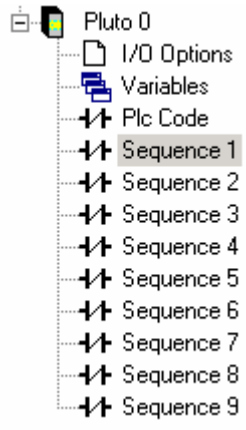
The program syntax in text form is interpreted as follows:

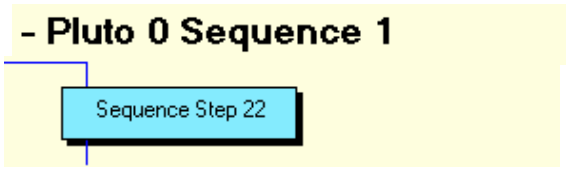
- The first letter concerns sequence register (S)
- The first number sets the identity of the PLUTO-unit where sequence register is to be addressed
- The second number (placed after dot-symbol) sets sequence register to be addressed
- The third number (placed after underscore) sets sequence step to be addressed

Example:

S0.1_22 ⇔ Start of step 22 in sequence 1 on PLUTO no: 0.

Sequence programming in Pluto Manager:

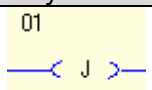
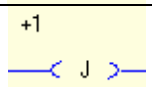
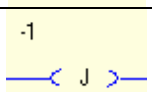




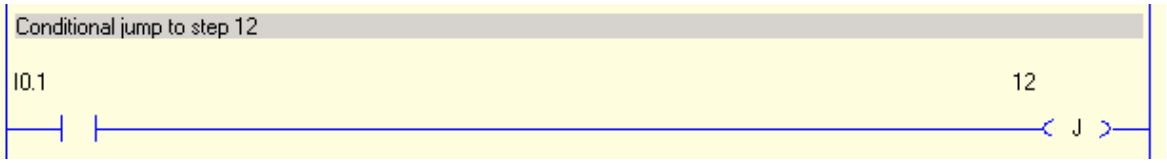
NOTE: By programming in text form Sequence 0 (S*n*.1_0) must be declared. This corresponds to what is named “PLC Code” in ladder programming and is intended to contain the PLC code when sequence programming is not used. Even if it is declared as a sequence it does just contain step 0, the compiler does not accept jumps and declaration of other sequence steps.

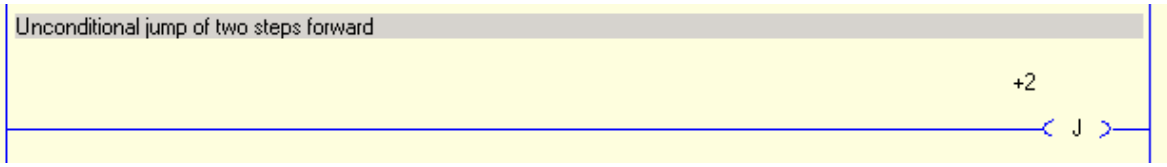
3.2 Jump

The jump instructions are used in sequences in order to jump from one step to another. Jump between sequences steps within a sequence can be performed either absolute or relative to the current active step.

Jump function:	Syntax in text form:	Ladder symbol
Absolute: to step 1	J(01)	
Relative: one step forward	J(+1)	
Relative: one step backward	J(-1)	

The jump can be either condition or unconditional.





Example of a sequence in text form:

```
S0.1_00                    ⇔
Q0.1 = I0.2
J(+1) = Q0.10*M0.7
```

Pluto 0, sequence 1, step 0:
Q0.1 is operated by I0.2
Jump to the next step (step 1) when output

Q0.10 and M0.7 is HIGH

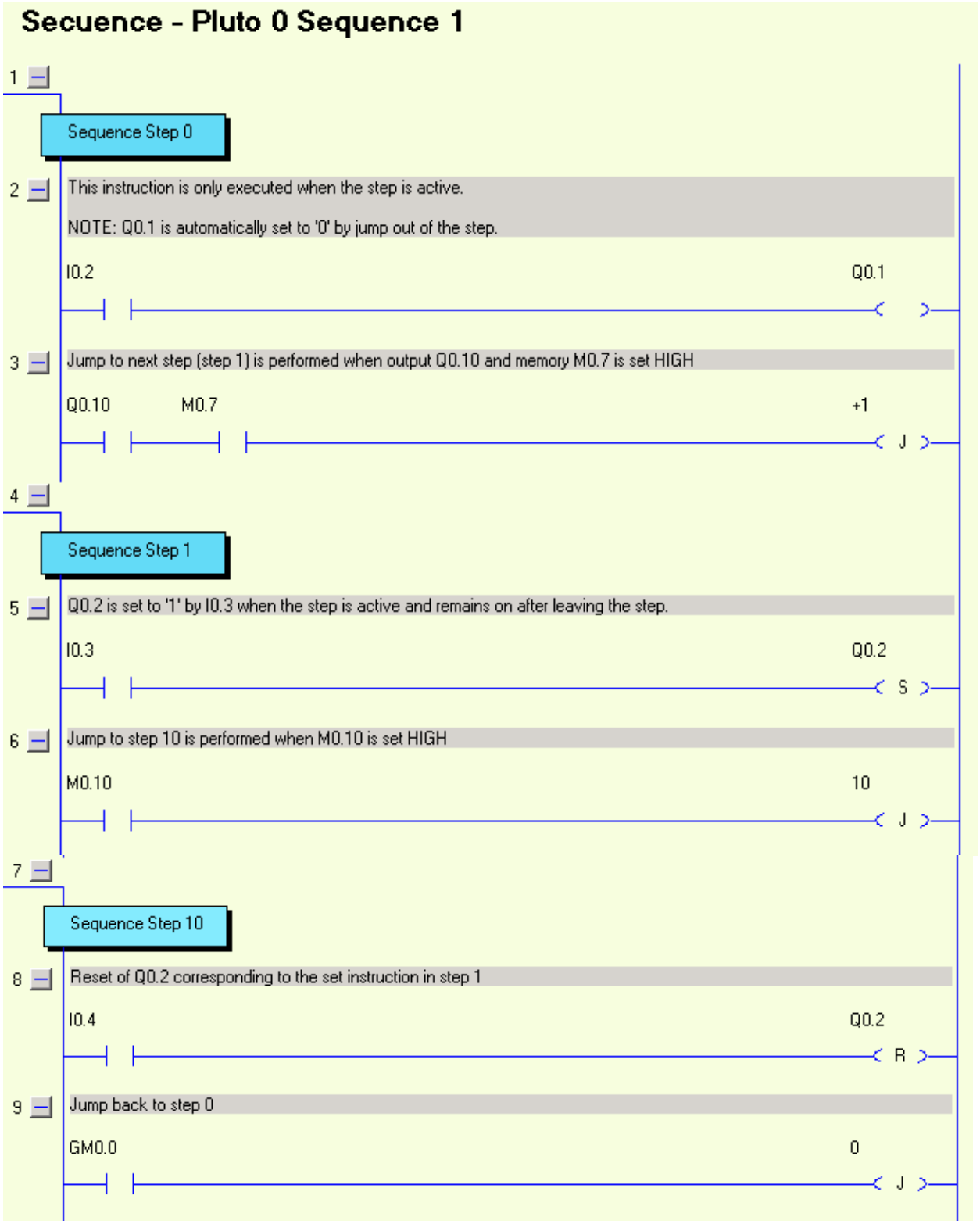
```
S0.1_01                    ⇔
S(Q0.2) = I0.3
J(10) = M0.10
```

Pluto 0, sequence 1, step 1:
Output Q0.2 is set HIGH by I0.3
Jump to step 10 when M0.10 is HIGH.

```
S0.1_10                    ⇔
R(Q0.2) = I0.4
J(0) = GM0.0
```

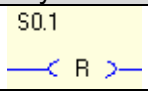
Pluto 0, sequence 1, step 10:
Output Q0.2 is set LOW by I0.3
Jump to step 0 when GM0.0 is HIGH.

The equivalence in ladder



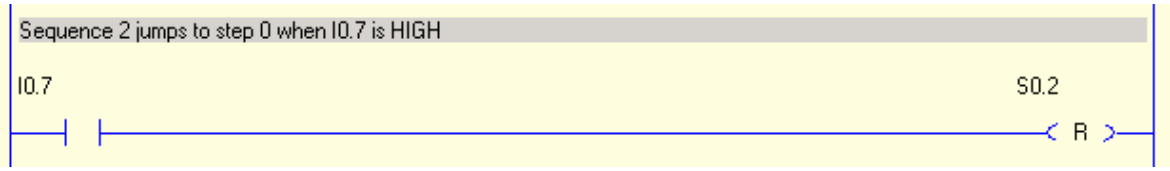
3.3 Reset sequence

It is possible to reset a sequence with code in another sequences.

Function:	Syntax in text form:	Ladder symbol:
Reset sequence	R(S0.1)	

Function:
Reset forces another sequence to jump to step 0, irrespective of the ordinary jump instructions. The sequence remains in step 0 as long the conditions for the reset instruction is TRUE

Example:




```
S0.1_05                    ⇔   In sequence 1 step 5 on PLUTO no: 0.
R(S0.2) = I0.7                 Reset of sequence 2 is demanded
                               when input I0.7 is set HIGH.
```

NOTE: Reset must be performed from another sequence.

4 Word-operands

4.1 Registers

4.1.1 Addressing

PLUTO has 150 16-bit registers where i.e. calculation results can be stored. The registers have the following number range: -32 768 … +32 767

Register are addressed as shown below:

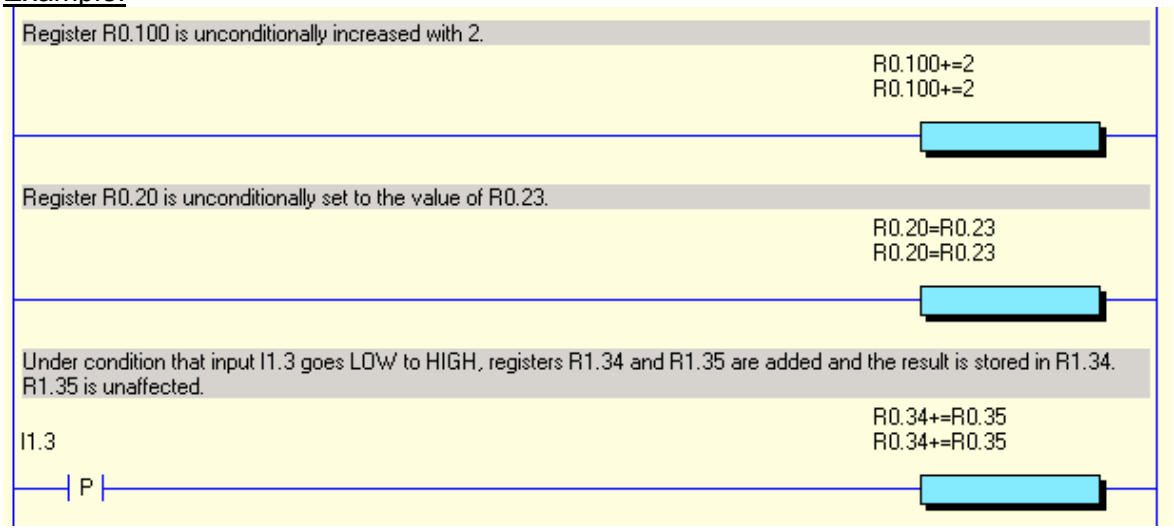
Register:	Syntax:
0-149	R0.0 – R0.149

4.1.2 Operations

Assignment of register

Operation:	Syntax:	
Increment by 1	(R0.100++)	
Decrement by 1	(R0.100--)	
Add constant	(R0.100 += 77)	
Subtract constant	(R0.100 – = 77)	
Assign with absolute value = 1	(R0.100 = 1)	
Addition with other register (R0.100 = R0.100 + R0.101)	(R0.100 += R0.101)	
Subtract with other register (R0.100 = R0.100 – R0.101)	(R0.100 – = R0.101)	
Assign with other reg. value	(R0.100 = R0.101)	

Example:



Equivalence in text form:

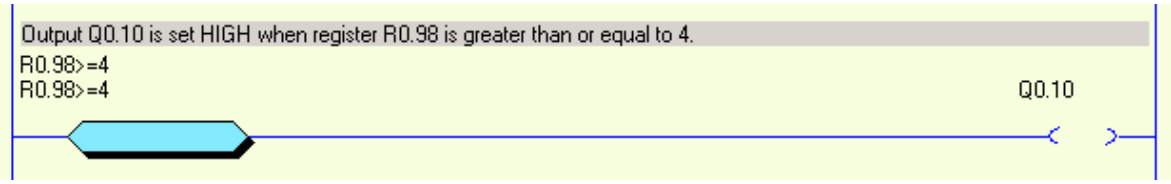
```
(R0.100+=2)
(R0.20=R0.23)
(R1.34+=R1.35) = P(I1.3)
```

Function: By increment of register the increment stops when the register value reaches the limits (32 767 or -32 768)

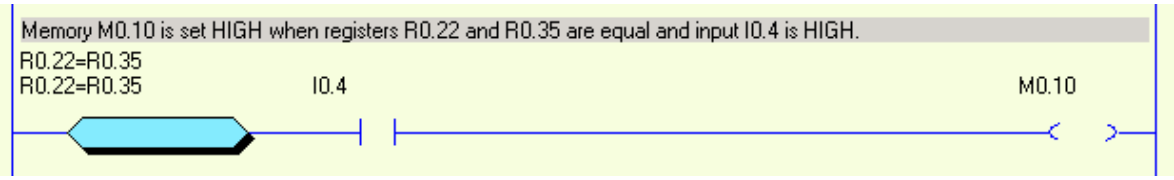
Comparison of register

Comparison	Syntax:
Equal to (constant)	(R0.100=1)
Greater than	(R0.100>1)
Less than	(R0.100<1)
Greater than or Equal to	(R0.100>=1)
Less than or Equal to	(R0.100<=1)
Equal (two registers)	(R0.100=R0.101)
Greater than	(R0.100>R0.101)
Less than	(R0.100< R0.101)
Greater than or Equal to	(R0.100>= R0.101)
Less than or Equal to	(R0.100<= R0.101)

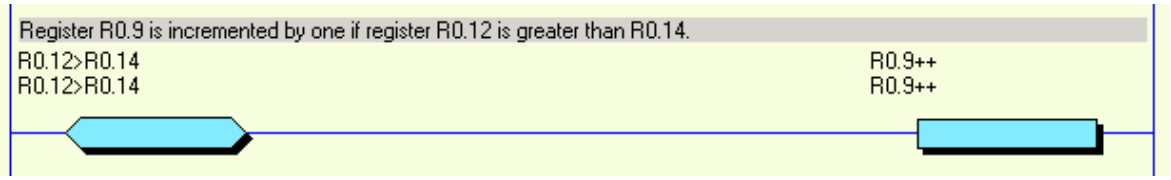
Example:



In text form: `Q0.10 = (R0.98>=4)`



In text form: `M0.10 = (R0.22=R0.35)* I0.4`



In text form: `(R0.9++) = (R0.12>R0.14)`

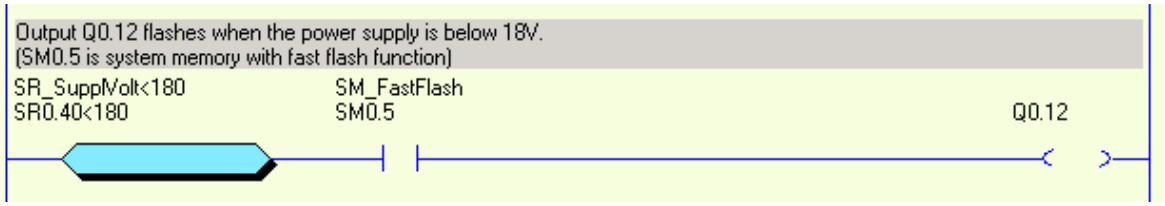
4.1.3 System registers

PLUTO has a set of system registers with different functions.

System registers
Syntax: SR[*unit*].[*no*]

I/O-address	Symbolic name	Function	Type
SR_.10	SR_PlutoDisplay	Display figure	W
SR_.40	SR_SupplVolt	Supply volatge, 10 x Volt	R
SR_.41	*SR_I5_Volt*	*Voltage analogue input I5, 10 x Volt*	*R*
SR_.42	SR_Q16_Current	Current (mA) output no.Q16	R
SR_.43	SR_Q17_Current	Current (mA) output no.Q17	R

Example:



In text form: `Q0.12 = (SR0.40<180) * SM0.5`

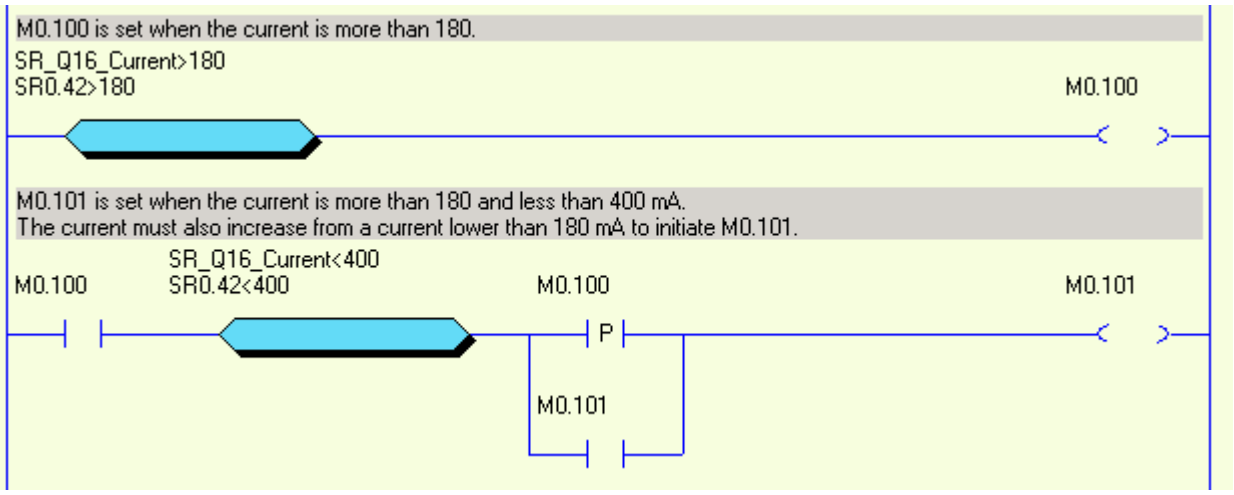
4.2 Use of analogue values

The analogue values are available by reading the system registers SR40…SR43. There are some requirements for the use these functions.

Analogue input I5:
The terminal for input I5 is also an analogue input measuring the voltage at the terminal. In SR41 the value can be read in tenths of volts, (240 = 24.0 volt).
By use in safety applications a 0-value may not be used as safe condition unless it is used in a dynamic monitored way (the program must monitor that the input value changes). This requirement is because the value in SR41 will be set to 0 if an internal fault in the system occurs.

Current monitoring of Q16 and Q17:
The output current from Q16 and Q17 is available in SR42 and SR43, the value represents mA. The function is intended for monitoring the current in a muting lamp, but other usage is not excluded. As the hardware for measuring the current is not fully redundant the values must be used in a dynamic way. For example if a current to a muting lamp shall be monitored the program must be written so that the change of current by switching the input on and off is observed.

Example:



In text form:

```
M0.100 = (SR0.42>180)
M0.101 = M0.100 * (SR0.42<400) * (P(M0.100) + M0.101)
```

5 Program declaration in text form

In the beginning of the program file different declarations are made. These declarations describe the hardware environment for the Pluto unit.

For more information about the function of the different hardware options see the ‘Operating instructions, Hardware’

5.1 Identity, station number

Each unit unit must have a station number 0-31.
It is also possible to connect an external identifier cicuit containing a unique 12 figure hexadecimal number.
These two settings are declared as:

! id_pluto:[*stn.number*]=[*identifier number*]

If identifier is not connected the system will accept this if the identifier number is declared as 000000000000 (12 zero).

Example:

! id_pluto:00=ffff00007FA3 ⇔ The Pluto-unit is given station number 0 and an identifier with number ffff00007FA3 must be connected to the unit.

! id_pluto:23=000000000000 ⇔ The Pluto-unit is given station number 23
and the unit shall run without identifier.

5.2 Declaration of program code

Since it is possible two have program code for several units stored in one unit it must be declared to which Pluto unit a code part belongs to.

Syntax:

! pgm_pluto:*[station no.]*

5.3 Declaration of I/O

All inputs and the non failsafe output Q10…17 must be declared since they can de used in different ways. The tables below show the options.

Inputs
Syntax: ! I[no],[pulse type],[switch 1],[switch 2]
Example: ! I0.5,c_pulse,non_inv,no_filt

Inputs	Pulse types (Dynamic sign.)	Switch 1 (optional)	Switch 2 (optional)
I_.0 - I_.17	a_pulse b_pulse c_pulse	non_inv	no_filt
I_.0 - I_.17	static∗)		no_filt

∗) I_.10-I_17,**static** does not fulfil cat. 4 according to EN954-1, as stand-alone input

Dynamic outputs
Syntax: ! Q[*no],[pulse typ]*
Example: ! Q0.10,a_pulse

Inputs	Pulse types
Q_.10 – Q_.17	a_pulse, b_pulse, c_pulse

Non failsafe-outputs Syntax: ! Q[*no],static*
Example: ! Q0.10,static

Inputs	Pulse types
Q_.10 – Q_.17	static

Special function, Illuminated push button
Syntax: ! IQ[*no],[pulse type]*
Example: ! IQ0.12,a_pulse

Inputs	Pulse types
IQ_.10 – IQ_.17	a_pulse, b_pulse, c_pulse

Example:

```
! i0.1,a_pulse              ; Input is supplied with dynamic A signal via inverter.
! i0.2,a_pulse,non_inv      ; Input is supplied with dynamic A signal.
! i0.3,static               ; Input is supplied with +24V.

! q0.10,a_pulse    ; Output generates dynamic A signal for supply of inputs.
! q0.11,static     ; Input is supplied with dynamic A signal.
```

Special function, Cross talk

Syntax: ! I[*no],[pulse type],x_talk*
Example: ! I0.12,a_pulse,x_talk

Inputs	Pulse types
I_.10 - I_.17	a_pulse b_pulse c_pulse

5.4 Symbolic names

The variables can also be named with a symbolic name which can make a program easier to understand. In Pluto Manager it is declared on a separate page, see Pluto Manager manual.

By programming in text form it is declared. Where in the code the declaration is made depends on whether it is a global or local variable. Global variables I_._, Q_.0...4 and GM_.0..11 are declared before the program code for the first Pluto since the variable can be used in all Pluto:s. Local variables are named in the beginning of the of the program code for the corresponding Pluto, after the I/O declarations. See example.

Example:

```
! I0.0=MuteSensor1                ; Symbolic names global variables
! Q0.1=MuteSensor2
! GM0.1=MuteSensor2

! Q0.14=IndReset                  ; Symbolic names local variables
! M0.0=MutingActive
! R0.0=Counter1
```

6 Program example in text form

This program example is the program for the installation example showed in ‘Operating instruction, Hardware’

$name Example, manual

```
! id_pluto:00=000034AD4AE1

! pgm_pluto:00


! q0.10,a_pulse                ; Dynamic output A

! i0.00,static                 ; Muting sensor 1
! i0.01,a_pulse,non_inv        ; Muting sensor 2
! i0.02,a_pulse,non_inv        ; Test Contactors
! i0.12,a_pulse                ; Emergency stop PB
! i0.13,a_pulse                ; JSL Lightbeam
! iq0.14,a_pulse               ; Reset with indicator

;*****************************************

s0.0_0                         ; Main sequence start

q0.2 = i0.12 * (i0.13 + m0.0) * ( (p(i0.14) * i0.02) + q0.2)
q0.3 = q0.2
                               ; All safety outputs active when Emergency stop(I0.12)
                               ; and JSL(I.13) or muting(M0.0) are active.
                               ; Reset(I0.14) and Test(I0.02) are also needed in the
                               ; start condition.

q0.14 = /q0.2                  ; Reset indication active when outputs not active


;*****************************************

s0.1_0                         ; Muting Sequence
j(+1)=/i0.00*/i0.01*(SR0.43<100)     ; Start condition: both sensors not active

s0.1_1
q0.17 = i0.00 * i0.01 * i0.13
j(+1) = q0.17 * (SR0.43<100)  ; Muting start when both sensors and JSL active
s0.1_2
m0.0                           ; M0.0, Memory muting active
q0.17                          ; Indicator muting activ
j(0) = /i0.00 + /i0.01         ; Muting stopped by either sensor not active
```