

CyPro User Manual

version 53

applies to CyPro v3.2.4 and later



© 1998-2024 Cybrotech Ltd.

Cybrotech Ltd.
68 St Margarets Road, Edgware
Middlesex HA8 9UU
London, UK
info@cybrotech.com
www.cybrotech.com

Index

Index.....	3
Introduction.....	4
Installation.....	4
User interface.....	5
Online monitor.....	9
Identify modules.....	10
Multisend.....	11
Programming.....	12
Hardware.....	12
Variables.....	14
Refresh processing.....	19
Structured text.....	20
Operator panel.....	24
General.....	24
Print functions.....	24
Panel buttons.....	25
Panel masks.....	26
Program interface.....	29
Serial interface.....	31
Features.....	31
Free-programmable mode.....	32
Free-programmable radio.....	36
Free-programmable TCP/IP.....	37
Free-programmable SMS.....	39
Networking.....	40
Ethernet setup.....	40
Connection options.....	41
Socket interface.....	43
Features.....	46
Real-time clock.....	46
NAD alias.....	47
Password protection.....	47
Modbus slave.....	48
Mobile application.....	49
Command line options.....	55
Getting started.....	57
Appendix.....	59
Data type summary.....	59
Structured text summary.....	60
Program examples.....	63
Function library.....	64
Instruction list summary.....	65
Mobile app tags.....	67
Mobile app icons.....	68
Operator panel characters.....	69
Keyboard shortcuts.....	70

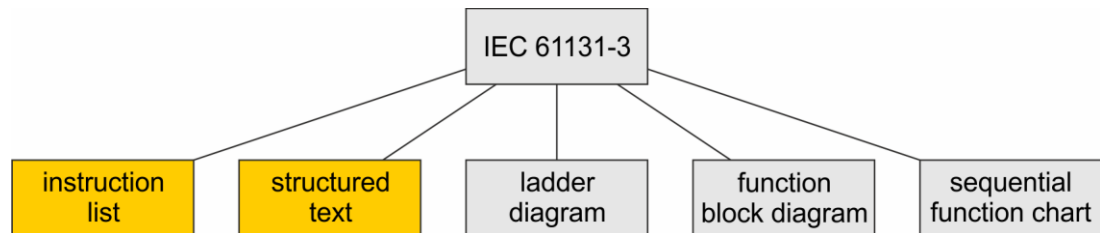
Introduction

Installation

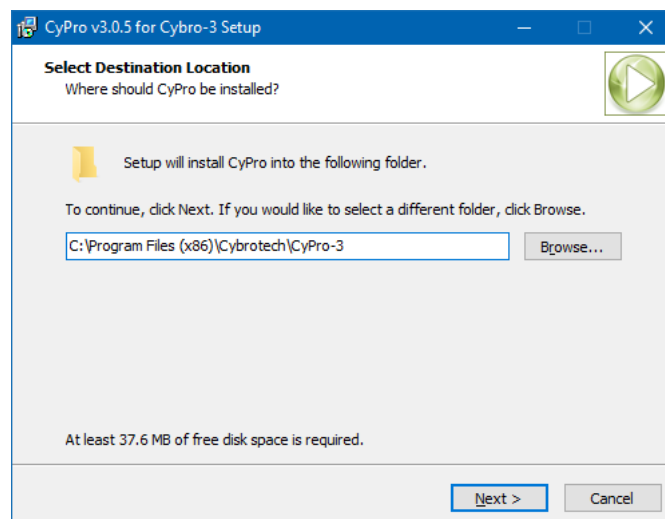
CyPro is integrated development environment for Cybro controllers, with text editor, compiler and on-line monitor. It's running on Windows 7/8/10/11 or Linux/Wine.

Each controller has unique 5-digit serial number, also used as communication address (NAD).

Compiler implements [structured text](#) (ST) and [instruction list](#) (IL) from IEC 61131-3 standard for programming logical controllers. Other languages are not supported.



Hardware requirements are modest, any PC capable of running MS Windows should be sufficient. Installation uses cca. 40Mb, default directory is C:\Program Files (x86)\Cybrotech\CyPro-3.



Installation does the following:

- copy files into the specified directory
- create start menu icons
- create desktop icon (optional)
- set association to .cyp file type (optional)

No file is copied to windows directory, no system files are replaced or changed. Default directory is:

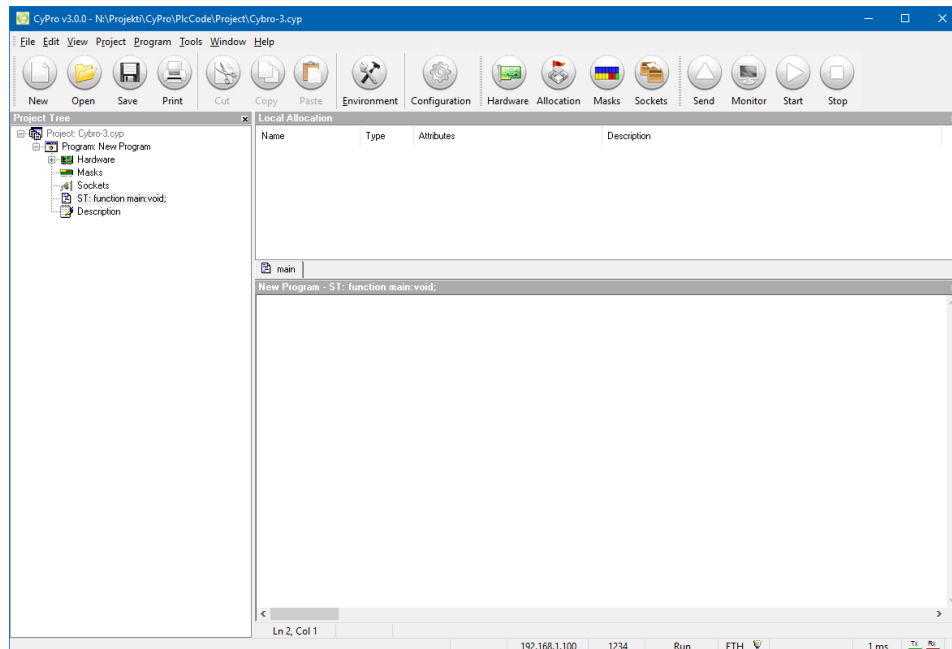
- C:\Program Files (x86)\Cybrotech\CyPro-3 (program and binaries)
- C:\Program Files (x86)\Cybrotech\CyPro-3\Examples (plc programs and function library)
- C:\Program Files (x86)\Cybrotech\CyPro-3\Projects (user projects)

To upgrade CyPro, install a new release into the same directory. User settings will be preserved. With new CyPro, it is required to also upgrade firmware (kernel). To do this, open [Tools/Kernel Maintenance](#) and send the new kernel.

User interface








Main window

CyPro consists of editor, toolbars and status bar. Default window is shown below:



Each component can be docked or floating. To undock, drag the component by the left vertical line over the edit area. To dock it again, drag window to main window border.





Standard toolbar

-  **New** Create a new empty project
-  **Open** Open an existing project (**Ctrl-O**)
-  **Save** Save current project (**Ctrl-S**)
-  **Print** Print current project (**Ctrl-P**)
-  **Cut** Remove the selection and place it on the clipboard (**Ctrl-X**)
-  **Copy** Copy the selection onto the clipboard (**Ctrl-C**)
-  **Paste** Insert the content of the clipboard at the cursor, replacing any selection (**Ctrl-V**)

Program toolbar

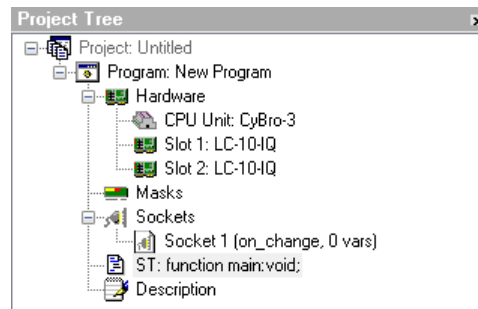
-  **Hardware** Open the [Hardware Setup](#) dialog box (**F5**)
-  **Allocation** Open the [Allocation Editor](#) dialog box (**F6**)
-  **Masks** Open the [Mask List](#) editor (**F7**)
-  **Sockets** Open the [Socket List](#) editor (**F8**)

Communication toolbar

-  **Send** Send current project to Cybro (**F9**)
-  **Monitor** Open the on-line [Variable Monitor](#) (**F10**)
-  **Start** Start program (**F11**)
-  **Stop** Stop program and turn off all outputs (**F12**)

Project tree

Displays project hierarchically.



Right clicking any component opens its context sensitive pop-up menu. Depending on type, it is possible to [Add](#), [Edit](#), [Delete](#) or change [Properties](#) of the selected component.

Status bar

Status bar shows various information about communication and connected Cybro.



System message (left side) show result of the preceding operation.

[Project status](#) indicate that current project is not saved. It reflects changes in any part of the project, such as source, allocation, mask, socket, data manager or monitor list.




[IP address](#) shows IP address of connected controller.

[A-bus address](#) shows Cybro A-bus address (NAD). Right click to select another or enter a new one.

[PLC status](#) shows:

Off-line	Cybro is not responding.
Run	Cybro is on-line and running.
Stop	Cybro is on-line, stopped. Outputs are inactive and program is not executing.
Pause	Cybro is on-line, paused. Outputs remain active, but program is not executing.
Error	Cybro is on-line, some error occurred. Error codes are listed in the appendix. To clear the error press Stop .
Loader	Cybro is on-line, but system software (kernel) seems to be damaged. Start Kernel Maintenance and send a new kernel.

[Com port status](#) indicates whether communication cable is properly connected:

	OK
	cable not connected
	communication port used by another application

[Delay](#) shows roundtrip time, from message sent to message received, in milliseconds.

[Communication indicators](#) show activity, green is transmit (Tx), red is receive (Rx).

Pull-down menu

File

New	Create a new project
Open	Open an existing project
Load From PLC	Load project from controller
Save	Save current project
Save As	Save current project under new name
Save alc File	Save allocation file in text format
Save csv File	Save allocation file in csv format
Printer Setup	Set printer options
Print	Print current project
Close	Close current project
Recent Projects	Open recently opened project
Exit	Exit program

Edit

Undo	Cancel the last action
Redo	Cancel the last Undo operation
Cut	Delete the selection and put it on the clipboard
Copy	Copy the selection onto the clipboard
Paste	Insert text from the clipboard to the insertion point
Delete	Delete the selection
Select All	Select the whole document
Find	Find specified text
Find Next	Find next occurrence of the specified text
Find Previous	Find previous occurrence of the specified text
Replace	Find specified text and replace it
Go to Line Number	Move insertion point to specified line number
Indent Block	Move selected lines right by inserting leading spaces
Unindent Block	Move selected lines left by deleting leading spaces
Comment/Uncomment	Insert or delete comment ("//") before selected lines
Insert Identifier	Display list of functions and global variables
Properties	Show properties of the selected project module

View

Project Tree	Show Project Tree
Local Allocation Editor	Show Local Allocation Editor
Editor Tabs	Show Editor Tabs
Compiler Messages	Show Compiler Messages
Standard Toolbar	Show Standard Toolbar
Program Toolbar	Show Program Toolbar
Communication Toolbar	Show Communication Toolbar

Project

New Program	Create a new program in the current project
New Program From PLC	Load program from controller into the current project
Remove Program	Remove program from the current project
Properties	Show properties of the current project

Program

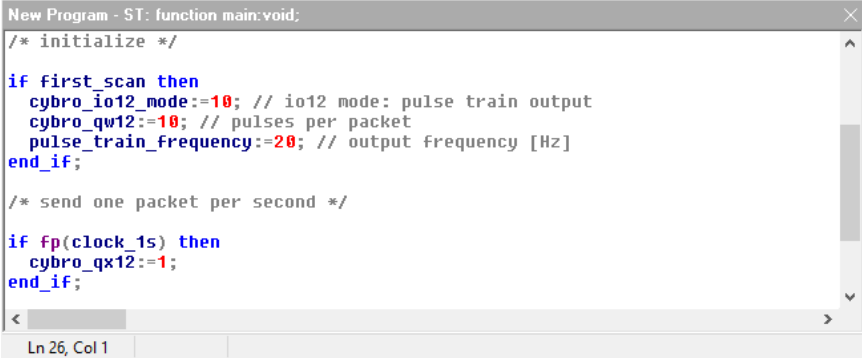
Hardware Setup	Open Hardware Setup dialog box
Allocation Editor	Open Allocation Editor dialog box
Mask Editor	Open Mask List editor
Socket Editor	Open Socket List editor
Syntax Check	Check the current file for errors
Send	Send current program to controller
Send Without Init	Send program without initializing variables, when possible
Start PLC	Start Cybro program
Stop PLC	Stop Cybro program and turn off all outputs
Pause PLC	Pause Cybro program, keep outputs active
Add NAD	Add new network address to the current program
Remove Current NAD	Remove current NAD from the current program
Select NAD	Select current network address for the active program
Connect/Disconnect	Connect/Disconnect communication port
Configuration	Settings related to plc program

Tools

PLC Info	Display various controller-related information
Kernel Maintenance	Update system software
Online Monitor	Online access to plc variables
Identify Modules	Identify IEX modules and individual inputs/outputs
Init all variables	Initialize all variables, including retentive
Multisend	Send program to multiple controllers
Erase Protected Program	Erase password protected program
Communication Monitor	Low-level A-bus communication monitor
Environment Options	Settings related to CyPro environment

Edit window

Edit window is used to type and edit PLC program. Each function has its own window.



```

New Program - ST: function main:void:
/* initialize */

if first_scan then
  cybro_io12_mode:=10; // io12 mode: pulse train output
  cybro_qw12:=10; // pulses per packet
  pulse_train_frequency:=20; // output frequency [Hz]
end_if;

/* send one packet per second */

if fp(clock_1s) then
  cybro_qx12:=1;
end_if;

```

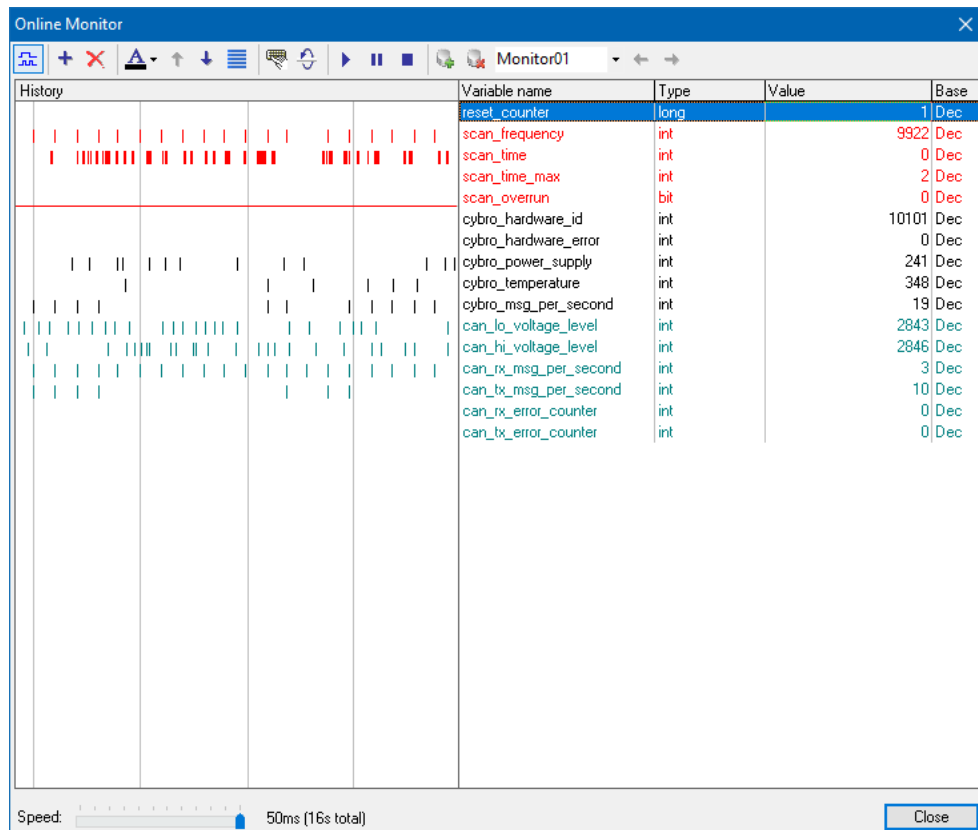
Ln 26, Col 1

Editor uses syntax highlight - variables, constants, functions and other language elements are displayed in different colors. To customize colors, open [Tools/Environment Options/Colors](#).

Insert identifier ([Ctrl-Space](#)) is used to display a list of allocated variables and available functions.

Online monitor

Online monitor is designed to display and change controller variables.



To insert new variables use **Add** button (**Insert**), select desired variables and press **OK**. To rearrange variables, click **Move Up** / **Move Down**, or use **Ctrl-Up** / **Ctrl-Down** (arrow) keys.

Monitor update rate is 20ms (50 times per second). Scroll rate is 50ms, it can be changed with **Speed** slider. First number is time to move a single pixel, second is total time from left to right.

To enter a new value, click **Edit selected variables** (**Alt+Enter**), right-click and select **Properties**, or double-click the variable.

The 'Edit variable values' dialog box shows the following fields:

- Variables:** reset_counter [long]
- Value:** 0
- Base:** Dec (dropdown menu)

Buttons: **OK**, **Cancel**

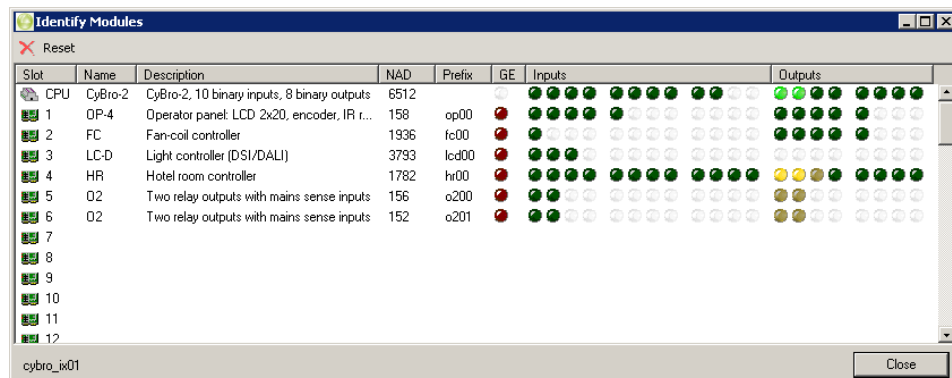
Enter value and press **OK**. Value is sent and immediately read back, monitor always display the actual value. Multiple variables can be set at once.

To toggle a bit variable, press **Space** key.

Monitor supports multiple sets. To create a new set click **Add new varset**, then insert variables. For a quick access press **Alt-1** to **Alt-5**, or **Ctrl-Left** / **Ctrl-Right** (arrow) keys.

Identify modules

Identify is a tool used to identify individual inputs/outputs.



Each LED represents single digital input or output. When mouse is positioned over LED, signal name is shown in the bottom left corner.

Input and output LEDs are defined according to the following table:

LED	current level	changed
	0	no
	1	no
	0	yes
	1	yes

General error (GE) is defined as:

LED	description
	module is operating properly
	error, module is not operative

To identify unknown input:

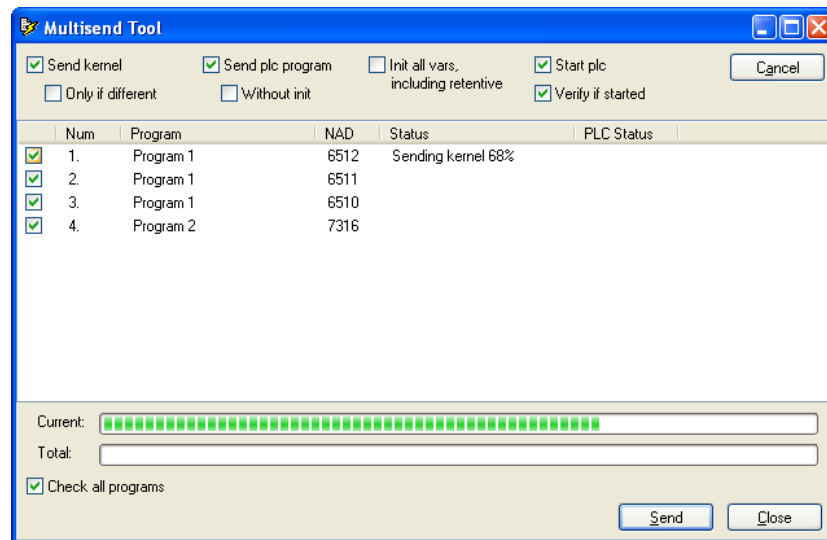
1. Reset all
2. Press and hold unknown input for a second
3. Look for the yellow LED

To identify unknown output:

1. Click LED to toggle the output

Multisend

Multisend is tool to update multiple controllers at once.



All programs and all NAD's are listed.

It is optional to send program either without initialization (only if allocation is not changed), with a standard initialization (retentive variables are preserved), or with forced initialization (all variables are initialized, including retentives).

Option [Check all programs](#) will verify all programs by reading back and comparing to original.

Programming

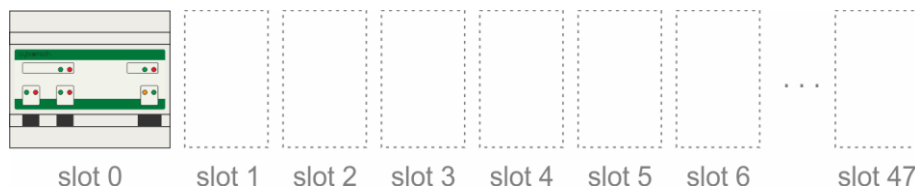
Hardware

Expansion modules

Cybro is expanded with IEX-2 modules. For the complete list, check hardware manual.



Each module occupies a single slot. Slot is logical entity, used to address the module.

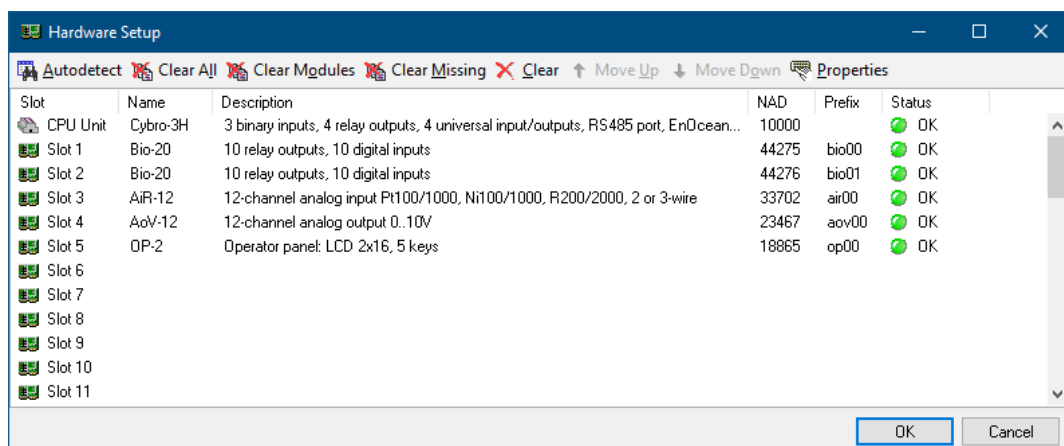


Each IEX-2 module has unique address, equal to serial number. Autodetect will detect module type, address, and assign slot number. Slot 0 is reserved for Cybro internal inputs and outputs.

Some modules implement autoaddress feature, used to fit devices into a predefined hardware list.

Hardware setup

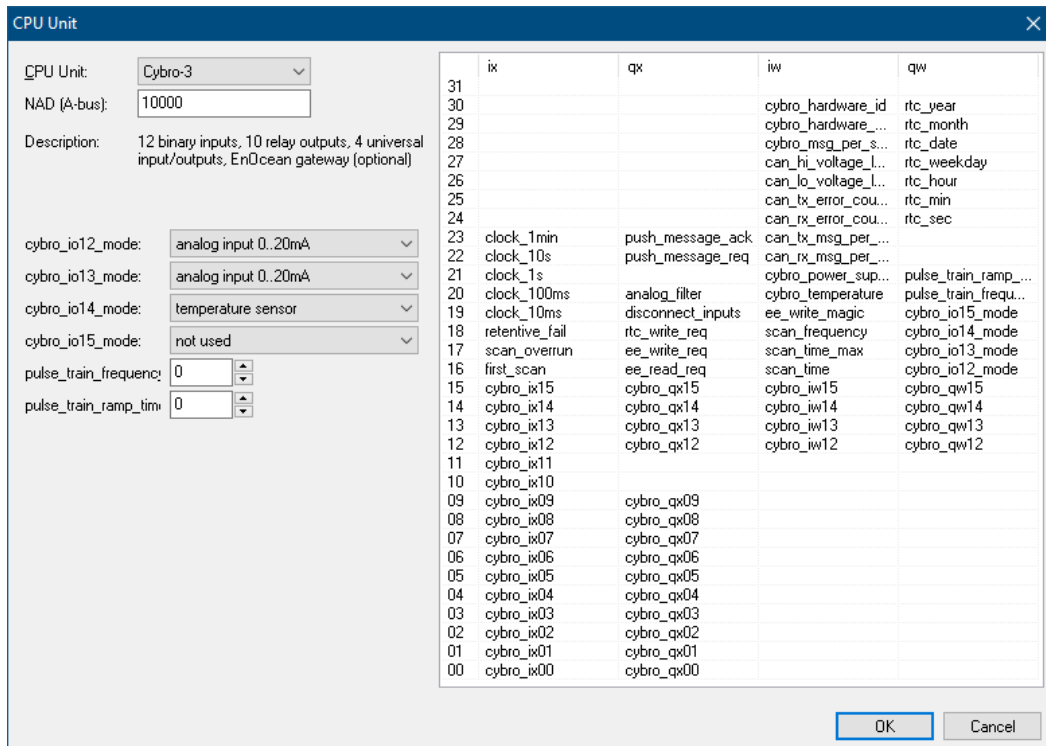
To perform automatic detection of connected modules press [Autodetect](#) button.



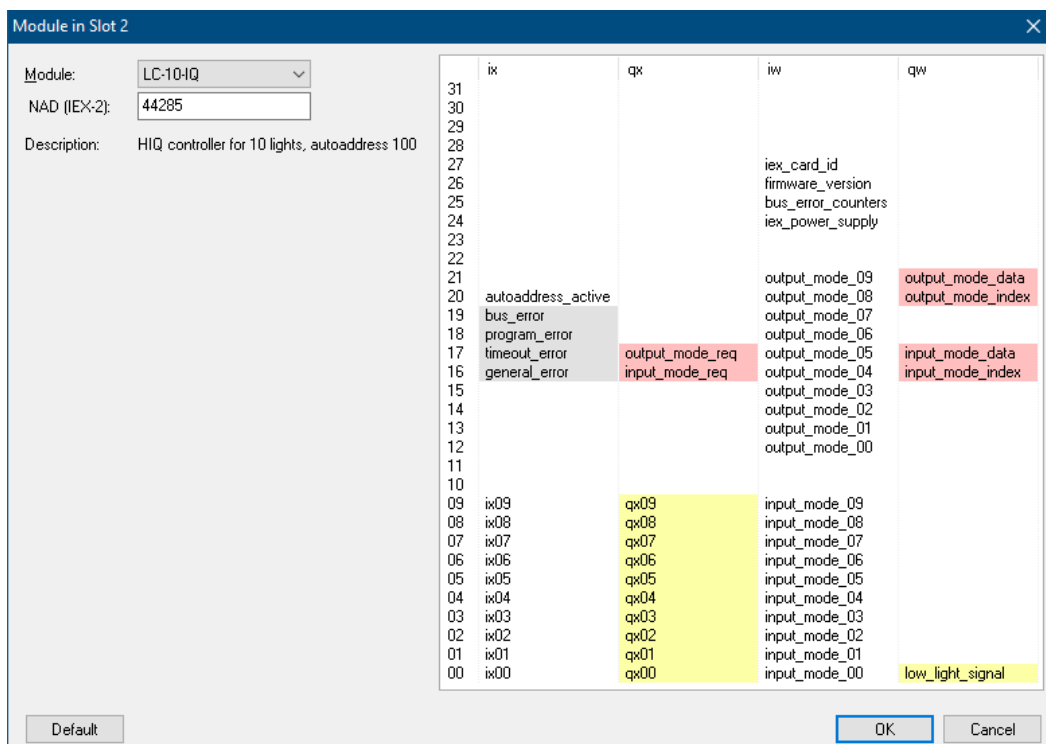
Dialog shows slot number, module type, short description, communication address, variable prefix and status.

Device properties

To open device properties, double click the module, or click right and select [Properties](#).



Dialog shows automatically assigned i/o variables for the module. Everything device does is accessible through this variables.



Variables are sorted in four columns: I-inputs, Q-outputs; X-digital, W-analog (word).

Each module has a four status variables (`general_error`, `timeout_error`, `program_error`, `bus_error`), shaded gray. When `general_error` is zero, everything is ok, module is fully functional.

Yellow shaded variables are sent on change. When changed, it is sent automatically.

Red shaded variables are sent on request. Each group of four has it's own request. To send the group, set request to 1.

To get description of each variable, hover mouse over. The description comes from the `cym` file.

Variables

Naming

Variable name may contain letters, digits and underline symbol. First character must not be a digit. Maximum length is 32 characters. Name is not case sensitive. Special and national characters (ß, ä, ü, ë, č, ć, š, ž...) should not be used.

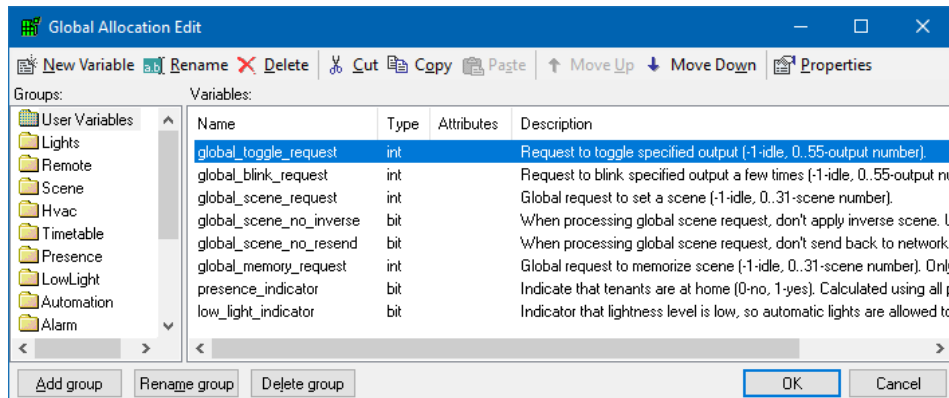
Examples of a valid name:

```
i
caret_position
maximum_water_level
```

Name must not match IEC-1131-3 keyword.

Allocation

Variables are allocated using [Global Allocation Edit](#). To insert a new variable, choose group and click [New Variable](#).



Basic data types

type	size	range
bit	1 bit	0..1
int	16 bits	-32768..32767
long	32 bits	-2147483648..2147483647
real	32 bits	-10 ³⁸ ..10 ³⁸

Bit is a single boolean variable with only two possible states, zero or one. It is used for flags, logical equations, logical states and other. The result of comparison instruction is also bit type.

Int is a 16-bit signed number. It is used for counting, encoding states, fixed point arithmetic and similar.

Long is a 32-bit signed value. It is used when numbers outside of 16-bit range may occur. Processing speed is the same as for the 16-bit integers, but they use more memory.

Real is a floating point number. Float consist of 8-bit exponent and 24-bit mantissa, so the result has 5 to 6 significant digits.

Other data types

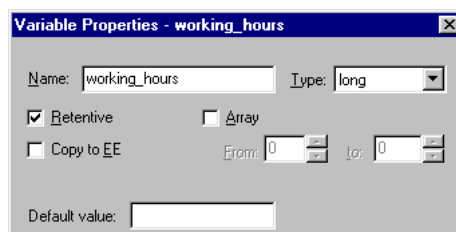
In bit, **out bit**, **in word** and **out word** variables represent physically connected binary (bit) and analog (integer) signals. In bit and out bit are bit type. In word and out word are integer type.

Timer is a structured data type, consisting of a several dedicated fields.

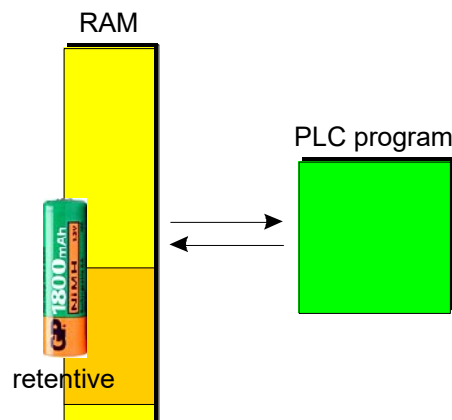
Constant is used to represent a value that will never be changed. For example, $\text{Pi}=3.141592$ can be defined for trigonometrical calculations. Constants are replaced in preprocessing, data type does not apply.

Retentive variables

Retentive variables retain their value when power supply goes down, and also when PLC is stopped. To make variable retentive, set the retentive flag in the global allocation dialog.



Both retentive and non-retentive variables reside in the same RAM, but retentives are automatically copied to battery backup RAM.



Number of retentive variables is not limited. If needed, the whole PLC memory can be retentive.

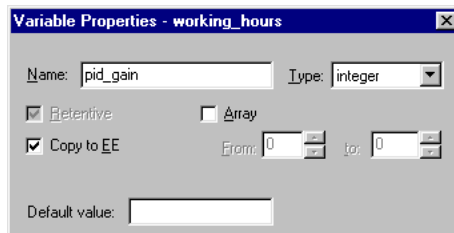
Data retention time is specified in Cybro hardware manual. When power is lost for a period longer than specified, content of retentive memory may be lost.

System bit **retentive_fail** indicates that retentive memory is damaged or lost. It is set automatically after power-on, and cleared next time PLC is started.

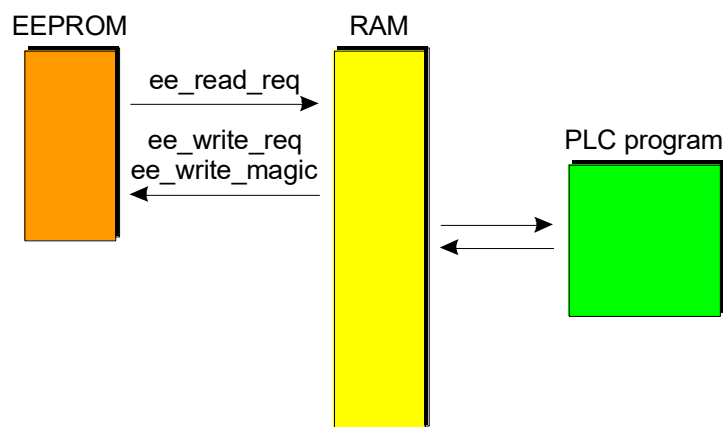
When allocation is changed, sending program to PLC will clear all variables. If allocation is not changed, retentives are preserved. To send program without initialization, use **Send Without Init**.

EE variables

Variables that must be preserved for a long period without electricity are stored in EEPROM. To configure this, set "Copy to EE" checkbox.



EE variables reside in RAM memory as all other retentive and non-retentive variables, but they also have a copy in EEPROM. Because of this, they are used by PLC program the same way as all other variables, but in addition, reading and writing to EE is available.



To read all variables from EE to RAM, set bit `ee_read_req`. Bit will be automatically cleared when copy is finished. Depending on number of EE variables, copy process may last a few seconds.

To write all variables from RAM to EE, set `ee_write_magic` to 31415 and set `ee_write_req`. When copy is finished, both variables will be cleared. Depending on number of EE variables, write process may last a few seconds. The purpose of magic is to protect from accidental writing.

Only the whole EE can be read or written, there is no method to read or write a single variable.

EE variables should not be accessed by program during read or write. The operation is finished when command variable (`ee_read_req` or `ee_write_req`) is returned to zero.

EE variables are automatically retrieved on power-up.

Total number of EE variables is limited by physical size of EE memory, specified by hardware manual. To check memory usage, open [PLC Info](#) dialog box, tab [PLC Program](#), [Total EE size](#).

I/O variables

I/O variables are used to access physical inputs and outputs. Cybro uses four I/O address spaces, two binary and two analog. Binary inputs and outputs are allocated respectively, starting from the ix0 as the first input and qx0 as the first output.

binary inputs	binary outputs
IX2047 slot 31 IX2016	QX2047 slot 31 QX2016
...	...
IX63 slot 1 IX32	QX63 slot 1 QX32
IX31 on-board IX0	QX31 on-board QX0

Analog i/o space has 32 analog inputs and 32 analog outputs for each slot. Slot 0 is reserved for Cybro local inputs and outputs. **In word** and **out word** variables are both integer type (16 bit signed).

analog inputs	analog outputs
IW2047 slot 31 IW2016	QW2047 slot 31 QW2016
...	...
IW63 slot 1 IW32	QW63 slot 1 QW32
IW31 reserved IW0	QW31 reserved QW0

Input and output variables are auto-allocated, their name is in the form:

nnnxx_varname

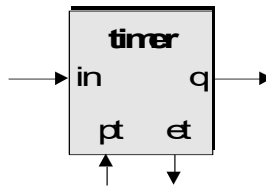
where nnn is prefix (e.g. bio for Bio-24), xx is card number (starting from zero) and varname is the function it performs. For example, operator panel key F is allocated as `op00_key_f`.

Timer

Special structured type, used to determine time interval. To define a new timer variable, open [Insert New Variable](#) dialog box, choose timer type, enter name, adjust preset, type and timer base, then press **OK**.

Timer base is a period in which the timer is incremented, time resolution of the timer.

Timer may be represented as the function block with two inputs and two outputs:



Correspondingly, the timer variable consists of four fields. Each field is an elementary data type.

name	direction	type	description
in	input	bit	input
q	output	bit	output
pt	input	long	preset time
et	output	long	elapsed time

To use timer, the following syntax applies:

```
<timer name>.<field>
```

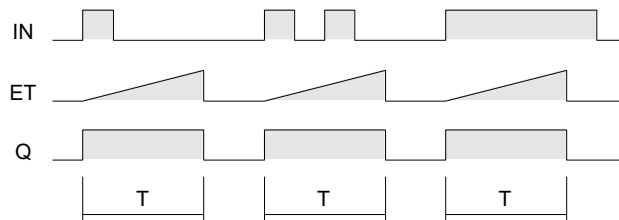
For example, to set the preset of the `wash_timer` to 15 seconds (assuming the base is 100ms):

```
wash_timer.pt:=150;
```

Elapsed time of the `wash_timer` will start at 0 and increment every 100ms until it reaches 150.

Pulse timer

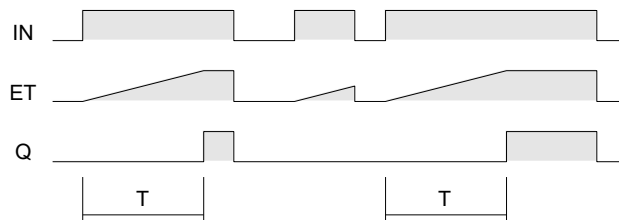
Timer output is activated immediately after the rising edge of input signal. After the specified time, the output will go off. Changes of input signal during active pulse do not affect output.



Typical application is a staircase timer.

On-delay timer

When input is activated, timer starts counting. After specified time output activates and stays high, until input goes low. Available fields are the same as pulse timer.



Typical application is a star-delta switch for three-phase motors.

Visibility

Each variable can be marked as:

User	visible across all tools
System	visible in tools used by administrators (CybroOpcServer, CybroDataTool)
Hidden	not visible outside of CyPro environment

Automatically allocated I/O variables are marked as "System".

Refresh processing

Cybro implements soft refresh processing. In a regular cycle, inputs are sampled immediately before and outputs are refreshed immediately after the execution of PLC program.

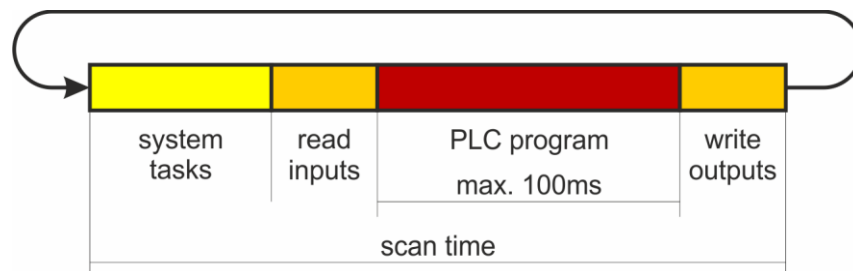
When scan time is very short, inputs and outputs may not refresh in each scan.

When scan time is very long, inputs may update during the scan, to reduce lag.

IEX modules are updated strictly before and after the scan.

Scan overrun

Scan time is defined as a time needed to complete a full program cycle. It consists of system tasks and PLC program.



When scan time exceeds 100ms, controller goes into [scan overrun](#) error and stops program execution (current scan will be finished). Error code is displayed on the status bar. To disable this feature, uncheck [Scan overrun stops program](#) in [Configuration options](#) dialog box.

When scan time exceeds 250ms, program will be interrupted by hardware watchdog, regardless of overrun settings. When this happens 10 times in a row, program will be stopped with [repetitive reset](#) error.

Structured text

Structured text is a high level language similar to Pascal, specifically developed for industrial applications.

Assignment

Assignment is used to store value in a variable. An assignment statement has the following format:

```
variable := expression;
```

The assigned value should be lower or equal data type than the variable.

Expressions

Expressions are used to calculate a value, derived from other variables and constants. Expression may use one or more constants, variables, operators or functions. Using expressions, Cybro can perform complex arithmetic operations, including nested parentheses and mixed data types.

Examples:

```
y_position:=5;
down_timer.pt:=15000;
case_counter:=case_counter+1;
start:=(oil_press and steam and pump) and not emergency_stop;
valid_value:=(value = 0) or ((value > 10) and (value <= 60));
```

Operators

Cybro supports a number of arithmetic and logical operators, listed in the following table:

operator	alias	unary	binary	function	bit	int	long	real	result
+			•			•	•	•	same
-		•	•			•	•	•	same
*			•			•	•	•	same
/			•			•	•	•	same
mod	%		•			•	•		same
not	!	•		•	•	•	•		same
and	&		•		•	•	•		same
or			•		•	•	•		same
xor			•		•	•	•		same
shl, shr			•			•	•		same
rol, ror			•			•	•		same
=	==		•		•	•	•	•	bit
<>	!=		•		•	•	•	•	bit
<, <=			•			•	•	•	bit
>, >=			•			•	•	•	bit
:=			•		•	•	•	•	same

Expression evaluation

Expressions are evaluated in a particular order depending on precedence of the operators and other sub-expressions. Parenthesized expressions have the highest precedence. Top precedence operators are evaluated first, followed by lower precedence. Operators of the same precedence are evaluated left to right.

Consider the following example:

```
Speed1 := 30.0;
Speed2 := 40.0;
Press := 50.0;
Rate := Speed1/10 + Speed2/10 - (Press+4)/9;
```

In this example, evaluation order is:

```
Rate := 30.0/10 + 40.0/10 - (50.0+4)/9
Rate := 3.0 + 4.0 - 54.0/9
Rate := 3.0 + 4.0 - 6.0
Rate := 1.0
```

Evaluation order can be changed using parentheses:

```
Speed1 := 30.0;
Speed2 := 40.0;
Press := 50.0;
Rate := Speed1/10 + Speed2/(10 - (Press+4)/9);
```

In this example, evaluation order is:

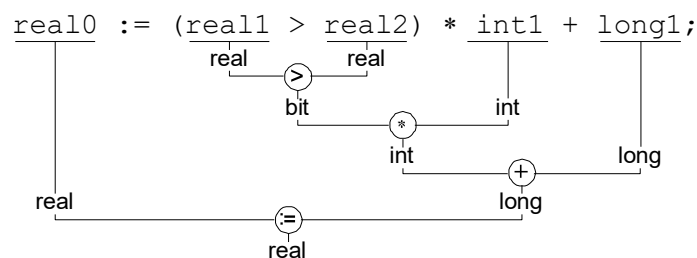
```
Rate := 30.0/10 + 40.0/(10 - (50.0+4)/9)
Rate := 30.0/10 + 40.0/(10 - 54.0/9)
Rate := 3.0 + 40.0/(10 - 6.0)
Rate := 3.0 + 40.0/4.0
Rate := 3.0 + 10.0
Rate := 13.0
```

Type conversion

Lower-to-higher data type conversion is performed automatically:

bit → int → long → real

In the following example, multiple of implicit conversions are performed:



If both arguments are integer, result is also integer, regardless of the operation.

```
i := 25;
r := i/10; // result is r=2
```

To get the expected result, constant should be written as 10.0:

```
i := 25;
r := i/10.0; // result is r=2.5
```

Same result can be obtained by using the cast operator:

```
i := 25;
r := real(i)/10; // result is r=2.5
```

Multiline expressions

In a multiline expression, each line must end with an operator:

```
heater_on := (heater_temperature < 600) and
             ((mode = MANUAL) and start_pressed) or
             ((mode = AUTO) and heater_request));
```

Flow control

This commands define order in which program statements are executed.

if..then..else

Conditionally execute one or another block of statements:

```
if <expression> then
  <statements>;
elsif <expression> then
  <statements>;
else
  <statements>;
end_if;
```

Example:

```
if a>(2*b) then
  d:=3;
elsif a>b then
  d:=2;
elsif a=b then
  d:=1;
else
  d:=0;
end_if;
```

case..of

Conditionally execute one of multiple statements. It consists of an selector and a list of statements, each preceded by a constant. Selector type must be ordinal (boolean, integer or long).

```
case <expression> of
  <value>: <statements>;
  <value>: <statements>;
  <value>: <statements>;
else
  <statements>;
end_case;
```

Example:

```
case material_type of
  1: speed:=5;
  2: speed:=20;
     fan:=ON;
  3: speed:=40;
     fan:=ON;
     cooling:=ON;
else
  speed:=0;
end_case;
```

for..do

The `for...do` construction allows a set of statements to be repeated specified number of times. Counting variable is incremented by 1 at the end of the loop.

```
for <var>:=<expression> to <expression> do
  <statements>;
end_for;
```

The statements within the loop must not contain `fp` or `fn` instructions.

Example:

```
for i:=0 to 19 do
  channel[i]:=TRUE;
end_for;
```

while..do

The `while...do` construction allows one or more statements to be repeatedly executed while particular boolean expression is true. The expression is tested prior to executing the statements. When it becomes false, statements are skipped and the execution continues after the loop.

```
while <expression> do
  <statements>;
end_while;
```

The statements within the loop must not contain `fp` or `fn` instructions.

Example:

```
while value<(max_value-10) do
  value:=value+position;
end_while;
```

Return value

Structured text function may return a single value of one of the basic types (bit, int, long, real). Return value is defined by the following expression:

```
result := expression;
```

Variable `result` is automatically declared when function is configured to return a value (function properties). Data type is the same as the type returned by function. Within a function, `result` may be used more than once:

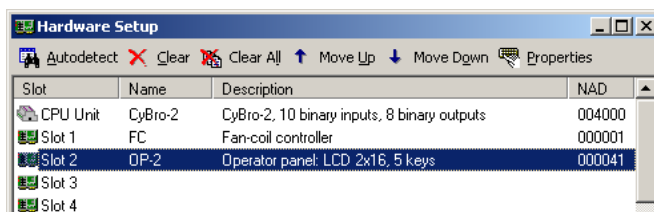
```
if a<=b then
  result:=a;
else
  result:=b;
end_if;
```

Operator panel

General

Operator panel is the optional external device connected to the Cybro via the IEX-2 bus. OP provides LCD display and a few keys readable from the PLC program.

OP has to be defined in the [Hardware Setup](#) dialog box. Configuration is saved within project.



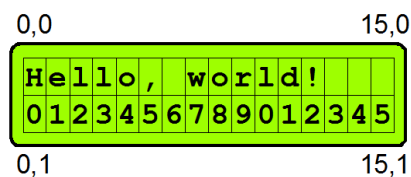
To program operator panel, the following tools are available:

- Print functions Structured text functions typed in the PLC program. Used to display strings and values.
- Panel buttons Bit variables readable from PLC program, represent current button state.
- Panel masks Visual tool for programming operator panel, used to enter parameters. Capable of entering integer values, decimal values and values represented by strings. Parameters may be hierarchically organized.

Print functions

Print functions are structured text functions used to display text messages and values.

First parameter is slot number where display appears in the hardware setup. Two following parameters of all functions are x and y coordinates. They are used to set display position. Print origin is in the upper left corner.



Printing outside visible range may produce unexpected results.

Print functions are:

```
dclr(slot:int);
```

Clear the whole display (fill with spaces).

```
dprnc(slot:int, x:int, y:int, c:char);
```

Print single ASCII character on specified coordinates. Character may be entered directly ('A'), as ASCII constant (65), or as integer variable. Values from 0 to 255 are allowed.

```
dprns(slot:int, x:int, y:int, str:string);
```

Print a string of characters, enclosed in single quotes.


```
dprnb(slot:int, x:int, y:int, c0:char, c1:char, value:bit);
```

Print first or second ASCII character, depending on bit value. If **value** is false, the first character is printed, otherwise the second.

```
dprni(slot:int, x:int, y:int, w:int, zb:bit, value:int);
```

Print integer value to specified coordinates. Parameter **w** defines width. For example, if **w** is 4, print range is -999 to 9999. Parameter **zb** is zero blanking. If **zb** is 1, leading zeroes are replaced with spaces.

```
dprnl(slot:int, x:int, y:int, w:int, zb:bit, value:long);
```

Print long value to specified coordinates. Parameter **w** defines width. For example, if **w** is 6, print range is -99999 to 999999. Parameter **zb** is zero blanking. If **zb** is 1, leading zeroes are replaced with spaces.

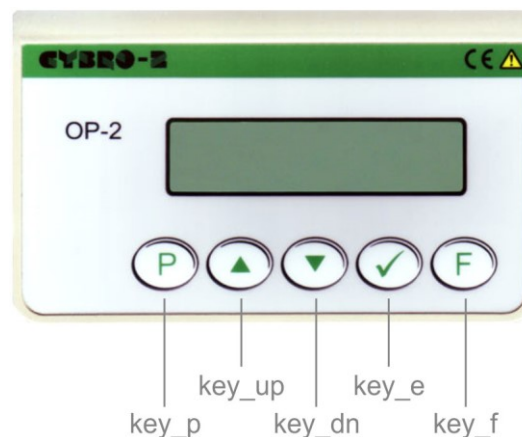
```
dprnr(slot:int, x:int, y:int, w:int, dec:int, value:real);
```

Print real value to specified coordinates. Parameter **w** defines width, parameter **dec** defines number of decimals. For example, if **w** is 6 and **dec** is 2, print range is -99.99 to 999.99. Zero blanking is always on.

Each parameter (except string in dprns) may be constant, variable or expression.

Panel buttons

Operation panel buttons are accessible from PLC program as binary input variables:



Key **P** is used to invoke and exit mask, so it's not available for PLC program (reading is zero). However, if no entry point is defined, it behaves the same as other keys. In such case, mask may be invoked by writing mask number to [op00_next_mask](#).

When mask is active, **up**, **dn** and **e** are not available (readout is zero). Key **F** is always available.

Key variable is true as long as the key is pressed. When key is released, it becomes false.

Any two (or more) keys may be pressed simultaneously. This may be used to initiate a special function. In the following example, pressing up and down simultaneously resets [product_count](#).

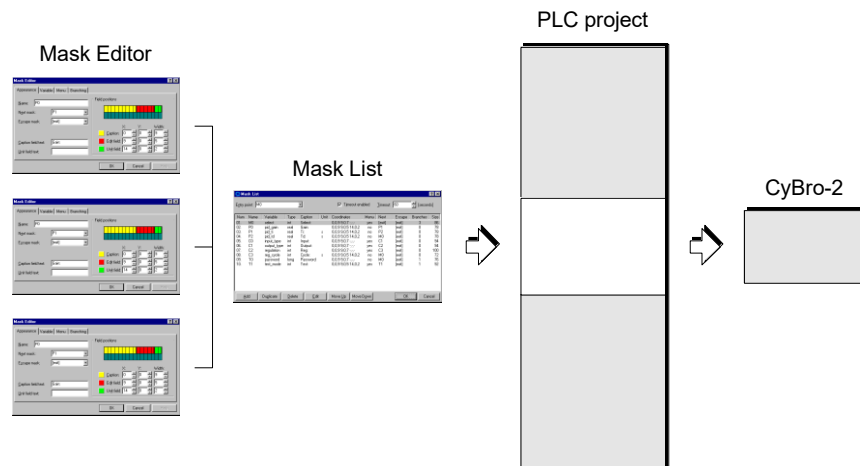
```
if fp(op00_key_up and op00_key_dn) then
  product_count:=0;
end_if;
```

Variables are allocated automatically when OP is defined in Hardware Setup.

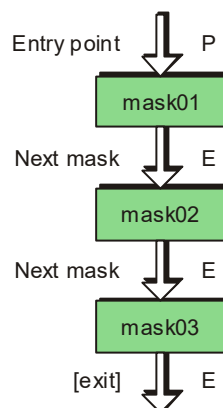
Panel masks

Mask is visual tool for creating user inputs on operator terminal. Masks are transferred to the Cybro together with PLC code.

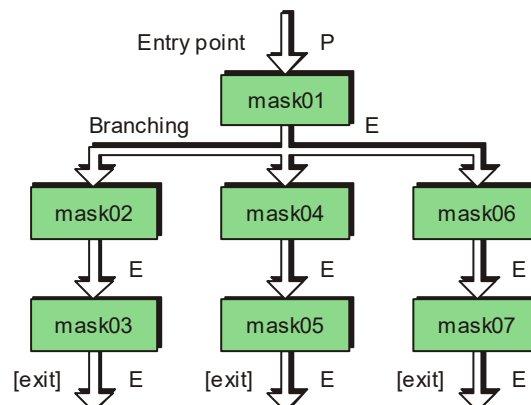
User creates a new mask or edits the existing one by using [Mask Editor](#). Created masks are listed in the [Mask List](#). Masks are integral part of the PLC project, they are saved on the disc and transferred to the controller.



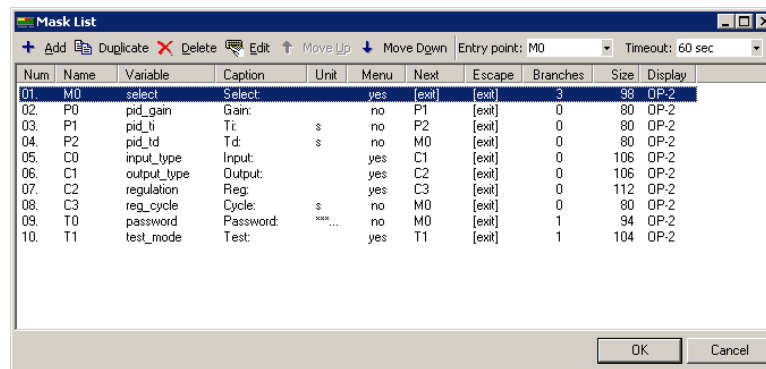
When user presses **P**, Cybro sends first mask to the OP. Pressing **E** advances to the next mask.



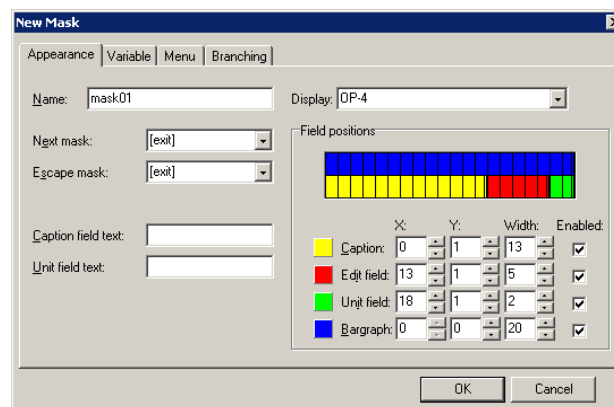
Masks can be organized hierarchically:



To start working with masks, press **Masks** button or **F7**. **Mask List** dialog box will appear.



To create a new mask click **Add** or press **Insert** key. **Mask Editor** dialog box will appear.



Name is a unique string identifier that identifies a particular mask.

Next mask defines a mask that becomes active after **E** key is pressed.

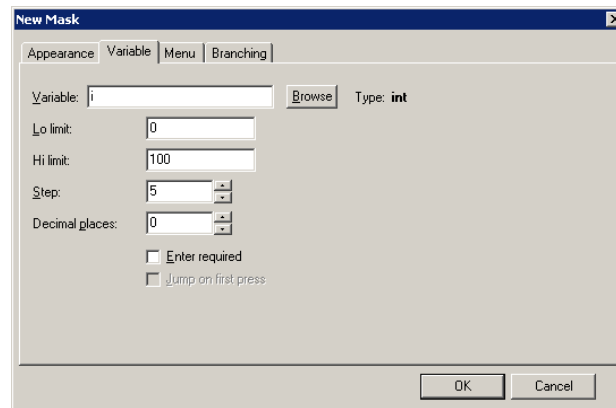
Escape mask defines a mask that becomes active after **P** key is pressed. Usually, this key is used to exit from mask.

Caption field is a short string that will appear on the display to identify the currently edited variable. Caption position is represented by the yellow rectangle. To move the caption, drag the rectangle into the desired position. To resize caption, drag the right edge of the rectangle.

Edit field is a display area in which the value of edited variable is displayed. It is represented by the red rectangle. Edit field should have enough space for editing variable in the desired range. To move and resize field, drag it like the caption.

Unit field is a short string, similar to caption. Unit field is represented with green rectangle, and it is commonly used for displaying engineering units.

Bargraph is a semi-graphic horizontal progress bar. Few different styles are available. To use bargraph, both low and high limits should be defined.



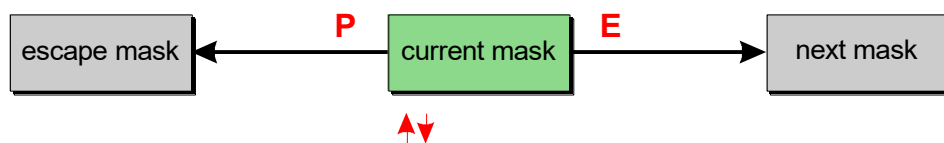
Lo limit and **Hi limit** define allowed range.

Step defines a value for which the variable will be changed for a single key press.

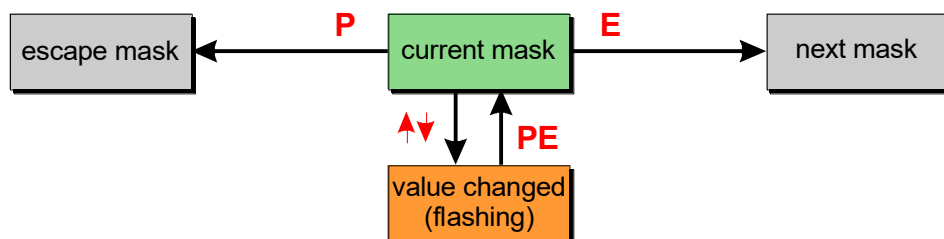
Decimal places may be used for real as well as for integer and long variables. In the former case, only the display is fractional (e.g. for decimal places=1, value 254 is shown as 25.4).

Enter required and **Jump on first press** define method to operate with navigation keys (P, E). Three combinations are available:

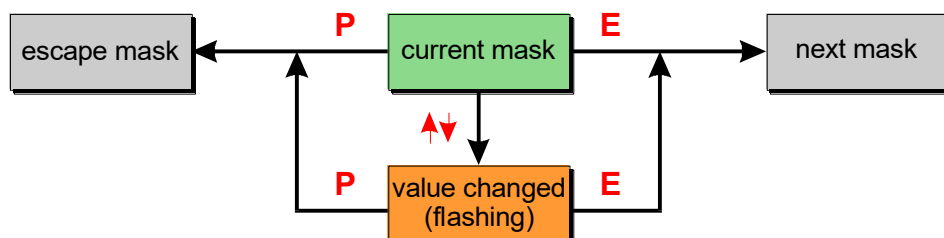
Enter required: no



Enter required: yes
Jump on first press: no

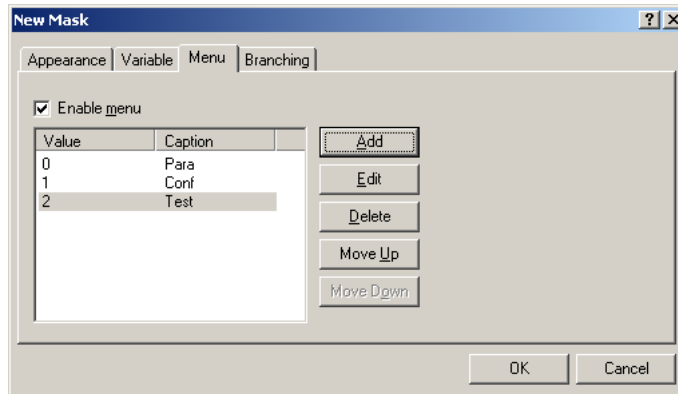


Enter required: yes
Jump on first press: yes



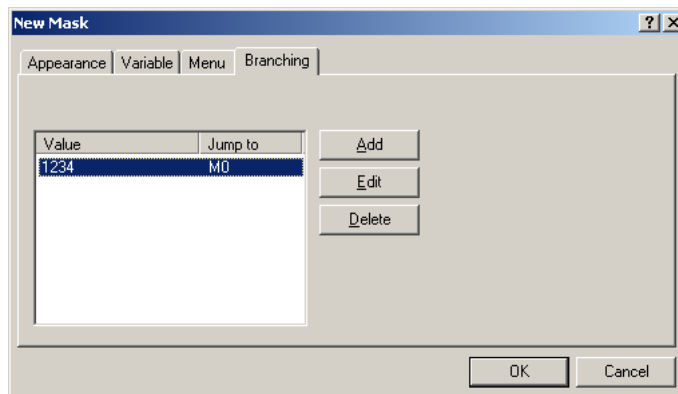
If enter required is false, changed value will be sent to Cybro immediately after **up** or **dn** key is pressed. If enter required is true, changed value will be sent to Cybro only when **E** key is pressed. To indicate that change is not confirmed, changed value will flash.

Variable may be entered as menu rather than as numerical value. To define menu entries, run [Mask Editor](#), click [Menu](#) tab and [Add](#) as many items as needed.



When executing Cybro program, the display will show items by name, and variable `product_type` will take value 0, 1 or 2.

Branching tab provides branching onto different masks according to the entered value. This can be used to organize parameters into various parameter sets, but also for a password protected parameters.



Active mask takes control of all panel keys except the **F** key, so it is not possible to use them from Cybro program at the same time. Mask fields are displayed “over” the user display. After exiting mask, display content is restored.

If mask is too large to fit into operator panel it will not be activated, and it will operate like an empty mask. Mask size is displayed in [Mask List](#) dialog box. Available operator panel mask memory is displayed in the [Hardware Setup](#) dialog box. To decrease mask size reduce number of menu entries or reduce edit field width. Reducing caption and unit field width may also save few bytes.

Only one mask can be active at the time.

Program interface

Cybro program can get currently active mask number by reading variable `current_mask`. When `current_mask` is zero, no mask is active.

Program may force execution of a certain mask by writing to variable `next_mask`. After the mask is sent, `next_mask` is set to -1, and `current_mask` changes accordingly.

The following example shows mask handling process:

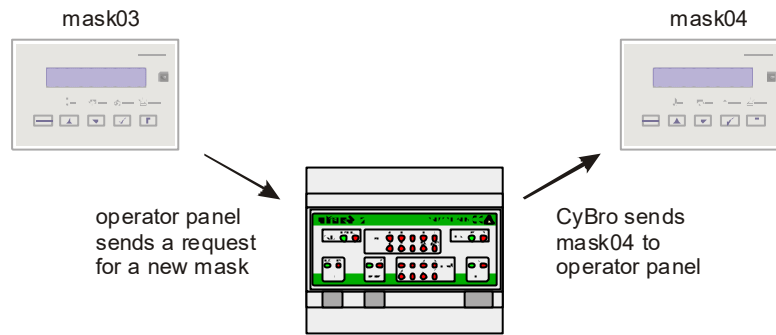


Table shows approximate timings and values for the transition:

		1 ↓	2 ↓	3 ↓		4 ↓
mask03 variable	20	20	→ 25	25	25	25
current_mask	3	3	3	0	→ 4	4
next_mask	-1	-1	-1	→ 4	4	-1
		2-3ms		2-3ms		50-100ms

Events are marked by black arrows:

1. Enter is pressed
2. Value is sent to Cybro
3. Request for new mask is sent to Cybro
4. New mask sent to operator panel and activated

Red arrows mark value change.

The same transition may be initiated with the following plc program:

```
if <condition> then
  op00_next_mask:=4;
end_if;
```

Short gap in `current_mask` value comes from the network response time. To check if there is an active mask, program should also check the value of `next_mask`, like the following example:

```
if op00_current_mask=0 and op00_next_mask=-1 then
  op00_next_mask:=10;
end_if;
```

Both mask control variables may also be accessed remotely, using the A-bus.

Serial interface

Features

Cybro controller features multiple communication ports. All of them can be used simultaneously. Port parameters are set at the compile time, it's not possible to change them within the program.

No	Port	description	A-bus slave	A-bus socket	Modbus master	Modbus slave	free pgm
1	COM1	RS232 serial port	yes	-	PLC program	yes	yes
2	COM2	RS232 serial port	yes	-	PLC program	yes	yes
3	COM3 ENO	RS485 serial port EnOcean interface	-	-	PLC program	-	yes
4	RFM	free-programmable radio interface	-	-	PLC program	-	yes
5	ETH	Ethernet interface, TCP/IP protocol	yes	yes	PLC program	yes	yes
6	CAN	SMS interface on GSM-2 module	yes	-	-	-	yes

A-bus is native protocol used to send program (A-bus slave), read/write variables (A-bus slave) and exchange data between controllers (A-bus socket). For more details, check Networking section.

Modbus protocol is developed for industrial applications. It is relatively easy to deploy and maintain compared to other standards, and places few restrictions on the format of the data. Modbus has become de facto standard and is now commonly available in various electronic devices.

Free-programmable means PLC program can send and receive messages, which opens up potential to implement various protocols.

COM3 port is serial port available on Cybro-3H and Cybro-3W as RS485 interface. On ENO models it is used for EnOcean transceiver. For more details, check hardware manual.

RFM wireless interface uses 868MHz ISM band to send and receive messages. It is used to control WD-1 (DALI bridge), WM-1 (Modbus bridge), WR-1 (Modbus relay) and WR-5 (Modbus relay). It can also be used for Cybro-to-Cybro communication. For more details, check device data sheet.



ETH interface enables plc program to send and receive TCP and UDP messages. Both server and client operation is supported.

CAN is virtual serial port on IEX bus. It can be used to send and receive SMS messages using GSM-2 module.

Free-programmable mode

With this feature, a wide range of devices can be controlled: various sensors, scales, printers, radio modems, camera and other. Protocol is implemented using the PLC program.



COM1, COM2 and ETH communication ports are full duplex, COM3, ENO and RFM are half duplex. Both master and slave operation is possible.

Serial ports have separate transmit and receive buffer. Each buffer is 1042 bytes in size. That allows for 1024 bytes payload and a few bytes for eventual descriptor and redundancy check.

Rx buffer



Tx buffer



Select port

```
com_select(port: int);
```

Select must be executed first, before other communication commands. Available ports are:

- 1 - COM1, RS232 serial port
- 2 - COM2, RS232 serial port
- 3 - COM3, RS485 serial port or EnOcean interface
- 4 - RFM, free-programmable radio
- 5 - ETH, free-programmable TCP/IP
- 6 - CAN, free-programmable virtual port

The best place for `com_select()` is at the beginning of function which implements the protocol. It may be executed in each scan, that has no effect on current receive and transmit operation.

Create message

Binary messages are created by writing byte by byte to the transmit buffer.

```
tx_bufwr(pos:int, data:int);
```

Write data byte to transmit buffer. Position is 0 to 1041, value is 0 to 255.

```
tx_bufrd(pos:int):int;
```

Read data byte from transmit buffer. Position is 0 to 1041, value is 0 to 255.

ASCII messages may be created with display print commands. Slot number is zero, x coordinate is buffer position, y coordinate is ignored. Output goes to the selected transmit buffer.


```
dc1r(0);
```

Fill both receive and transmit buffer with zeros.

```
dprnc(0, x:int, 0, c:char);
```

Write a single character on position x (same as `tx_bufwr()`).

```
dprns(0, x:int, 0, str:string);
```

Write a string enclosed in single quotes ('abcd'). Special characters are entered as two or three-character combinations:

combination	ASCII code	hex code
\n	CR LF	0D 0A
\r	CR	0D
\t	TAB	09
\\	\	5C
\xx	any	xx

The last option is used to enter any hexadecimal code 00 to FF, e.g. '\41' is the letter 'A'.

```
dprnb(0, x:int, 0, c0:char, c1:char, value:bit);
```

Write a single character, `c0` or `c1`, depending on the bit value.

```
dprni(0, x:int, 0, w:int, zb:bit, value:int);
```

Write 16-bit signed integer as ASCII decimal number. Parameter `w` is width, `zb` is zero blanking.

```
dprnl(0, x:int, 0, w:int, zb:bit, value:long);
```

Write 32-bit signed integer as ASCII decimal number. Parameter `w` is width, `zb` is zero blanking.

```
dprnr(0, x:int, 0, w:int, dec:int, value:real);
```

Write floating point value as ASCII number with decimals. Parameter `w` is total width, including decimal point and decimals. Parameter `dec` is number of decimals. Zero blanking is always on.

Send message

```
tx_start(size:int);
```

Send the prepared message. Parameter `size` is the number of characters to transmit.

```
tx_active():bit;
```

Check whether the transmitter is active: 0-no, 1-yes.

```
tx_count():int;
```

Number of characters left to send. When `tx_count()` is zero and `tx_active()` is true, the last character is transmitting.

```
tx_stop();
```

Stop transmitter. Current character will be finished, then `tx_active()` goes to zero.

Start receiver

```
rx_start(beg_ch:char, end_ch:char, len:int, beg_tout:int, end_tout:int);
```

Start receiving and define condition to stop.

beg_ch - first character of received message. When receiving is started, all characters are ignored, until the specified character is received. The character is written in the zero position of the receive buffer. To receive message with no specific start character, set to zero.

end_ch - last character of received message. When specified character is received, receiver is stopped (status 2). Character is written as the last byte of the received message. To receive message with no specific end character, set to zero.

len - expected length of received message. After the specified number of bytes is received, receiver is stopped (status 3). To receive a message of variable size, set to zero.

beg_tout - maximum waiting time for the first character, in milliseconds. When timeout is reached, receiver is stopped (status 4). To receive with no time limit, set to zero.

end_tout - maximum time between consecutive characters, in milliseconds. When timeout is reached, receiver is stopped (status 4). To receive with no time limit, set to zero.

For example, with 1200 bps, 8 bits and no parity; transmission of a single character takes about 8ms (start bit + 8 data bits + stop bit = 10bits, 10bits/1200bps = 8.3ms). In such case, end time is typically set to about 25ms.

Examples:

```
rx_start(0,0,0,0,0); // receive continuously
rx_start(0,0,0,0,50); // receive continuously, stop 50ms after the last character
rx_start(':', '\r', 0, 0, 0); // receive message starting with ':' and ending with CR
```

Maximum message length is 1042 bytes. When one character more is received, receiver is restarted and the number of received characters starts from 1 again. The buffer is not cleared.

Receiver and transmitter are fully independent.

```
rx_stop();
```

Stop receiving immediately (status 1).

```
rx_count():int;
```

Returns number of received characters. Function `rx_start()` reset number of characters to zero.

```
rx_active():bit;
```

Check whether the receiver is active: 0-no, 1-yes.

```
rx_status():int;
```

Returns receiver status:

- 0 - receiver active
- 1 - stopped by stop command
- 2 - end character detected
- 3 - requested size received
- 4 - timeout expired

Parse received message

```
rx_bufrd(pos:int):int;
```

Read data byte from receive buffer. Position is 0 to 1041, value is 0 to 255.

```
rx_bufwr(pos:int, data:int);
```

Write data byte to receive buffer. Position is 0 to 1041, value is 0 to 255.

```
rx_strcmp(pos:int, str:string):bit;
```

Compare receive buffer with a specified string. True when string matches, false otherwise.

```
rx_strpos(pos:int, str:string):int;
```

Search for the specified string. Search starts from the given position. If string is found, function returns position of the first matching character, otherwise it returns -1.

```
rx_strtoi(pos:int):int;
```

Read ASCII decimal number at the given position. If character at the specified position is space, it is skipped until a digit is found. Conversion continues until the first non-digit character.

```
rx_strtol(pos:int):long;
```

Read ASCII decimal number at the given position. If character at the specified position is space, it is skipped until a digit is found. Conversion continues until the first non-digit character.

```
rx_strtor(pos:int):real;
```

Read ASCII decimal number at the given position. If character at the specified position is space, it is skipped until a digit is found. Conversion goes until the first non-numeric character.

Example:

Received message may contain keywords OPEN, CLOSE, AUTO and SET=<value>. Keywords are sent in no particular order and separated by one or more spaces.

```
SET=225 OPEN AUTO
```

Program that parses message according to given specifications:

```
if rx_strpos(0, 'OPEN') <> -1 then
  main_valve=1;
end_if;

if rx_strpos(0, 'CLOSE') <> -1 then
  main_valve=0;
end_if;

if rx_strpos(0, 'AUTO') <> -1 then
  automatic_mode=1;
end_if;

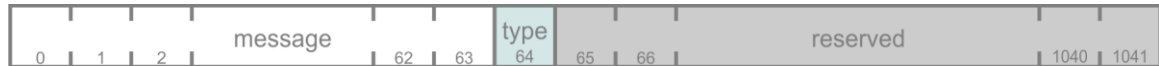
position=rx_strpos(0, 'SET=');
if position <> -1 then
  setpoint=rx_strtoi(position+4);
end_if;
```

Free-programmable radio

Initialize socket

The first command must be `com_select(4)`, it directs consecutive commands to the radio driver.

Rx/Tx buffer



The following command is `rx_start()`, providing parameters to initialize the radio interface:

```
rx_start(dummy:char, dummy:char, group_hi:int, group_lo:int, timeout:int);
```

`group` 32-bit group address, zero means factory default

`timeout`..... the time after which the reception stops [ms], zero to disable

Wireless devices use factory default address 10 seconds from power on, then switch to configured address, if one exists. That allows sending new group address to all devices at the same time. When 10s period runs out, address is locked to protect devices against intrusion. On Cybro, this process is under the control of PLC program, which allows receiving new address at any time.

Send and receive

RFM radio behaves very much like other serial ports. When message is received, receiver is stopped and need to be started again. To stop receiving at any time, use `rx_stop()`. Function `rx_active()` returns receiver state (0-off, 1-on), function `rx_status()` returns more details:

- 0 - receiver active
- 1 - stopped by stop command
- 2 - message received
- 4 - timeout expired

Command `tx_start()` begin transmitting prepared message, `tx_active()` returns transmitter state (0-off, 1-on). It is active immediately after the start command is executed.

Message type

The type byte (position 64) specifies the content of the message:

- 0 - DALI (WD-1)
- 1 - group address (all devices)
- 2 - Modbus or other serial protocol

Type must be set before the message is sent, and comes with the received message. Type 2 can be used for any serial communication.

Group address

By default, all devices share the same group address and listen to each other. To separate your devices, create a new secure group. Once group is created, no other device can listen or interfere with your data.

Group can be changed within 10 seconds of power up. After that, the group address is locked.

Note that groups share the same bandwidth. To avoid collisions, keep the traffic low or synchronize requests so that messages don't overlap.

For more details, check [RFM demo.cyp](#).

Free-programmable TCP/IP

Initialize socket

The first command must be `com_select(5)`, it directs all consecutive commands to TCP/IP driver. With ETH selected, first 10 bytes of buffer are reserved for IP header:

Tx buffer



Rx buffer



Receiver IP address and port must be written by plc program before the message is sent. Sender IP address and port are written by system when message is received.

The following command is `rx_start()`, providing parameters to initialize the TCP/IP socket:

```
rx_start(protocol:char, dummy:char, port:int, autostop:int, timeout:int);
```

`protocol`.... 0-none, 1-UDP, 2-TCP master, 3-TCP slave

`port` controller port through which messages are sent and received

`autostop`... when active, receiving reply message will close the connection

`timeout`.... when time runs out [ms], connection is closed; zero to disable

In UDP mode, the controller is ready to receive and transmit UDP messages right away.

In TCP mode, either master (client) or slave (server) operation is selected. When initialized as a master, Cybro uses receiver address and port to open the connection and send the first message. When initialized as a slave, Cybro enters listen mode, waiting for connection on the selected port.

To prepare the outgoing message, use `tx_bufwr()` or display print commands. To send the message, use `tx_start()`. Parameter `size` is the length of the message, without the header. Other transmit commands are not used.

To check if the message has been received, read the first byte of the buffer using `rx_bufrd()`. When result is not zero, message has arrived. The `rx_count()` returns received size, without the header. Parsing is the same as with the serial port. When finished, use `rx_bufwr()` to invalidate the message and prepare for the next one.

Command `rx_status()` returns state of the socket:

- 0 - closed
- 1 - UDP open
- 2 - TCP initialised
- 3 - TCP listen
- 4 - TCP connected

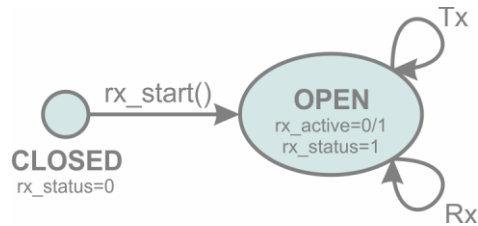
When message is transmitted or received, timeout is reloaded and `rx_active()` is set. When timeout expires, `rx_active()` goes to zero.

Command `rx_active()` returns 1 when connection is established (status 4). To close connection at any time, use `rx_stop()` command. To close the socket, use `rx_start()` with protocol set to zero.

Reserved local ports are 53 (DNS), 68 (DHCP), 8442 (A-bus LAN), 20000..29999 (A-bus WAN) and 502 (Modbus slave). Other port numbers are free to use.

UDP mode

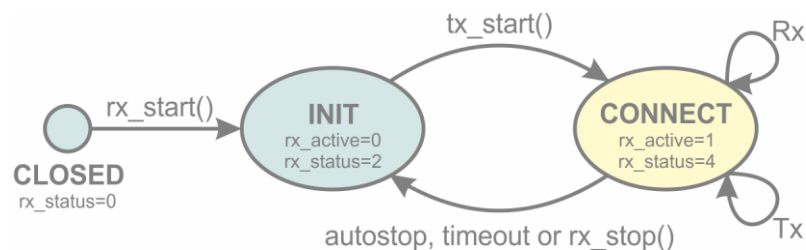
User Datagram Protocol (UDP) is a simple connectionless protocol that allow devices to send and receive messages. Sender destination port must be the same as the receiver local port. Message can be sent to multiple recipients using the subnet broadcast address.



Once socket is open, use `tx_start()` to send and `rx_bufrd()` to detect the received messages. Although the state is not changed, `autostop` and `timeout` can be used by reading `rx_active()`.

TCP master

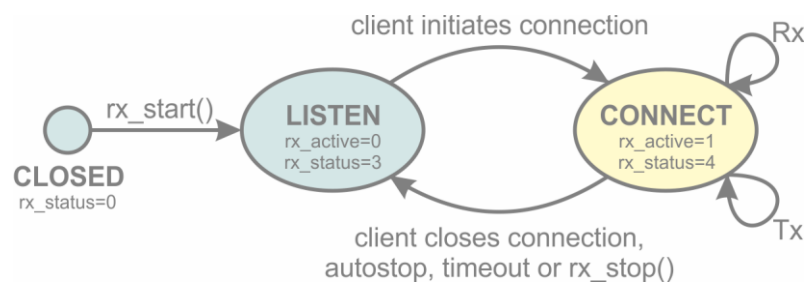
Transmission Control Protocol (TCP) is a connection-oriented protocol and requires handshaking to start communication. Once a connection is established, data can be sent. In master mode, connection is established when controller sends a message to the slave device.



If `autostop` is set, connection is closed when message is received. If `timeout` is set, connection closes when time runs out. Timer is reloaded with each received and transmitted message. Only one connection can be opened at a time.

TCP slave

In slave mode, controller is initialized and waiting for a connection.



The message can only be sent when the connection is established. When sending the message, receiver ip address and port are not used, since connection is already established.

If `autostop` is set, connection is closed when message is transmitted. If `timeout` is set, connection closes when time runs out. Slave timeout should be longer than or equal to the master timeout. Timer is reloaded with each received and transmitted message. Only one connection can be opened at a time.

Networking

Ethernet setup

Cybro may have a dynamic IP address given by DHCP server, or static IP address set in [Kernel Maintenance](#). To configure static address, turn on checkbox [Static IP address](#) and fill the fields. DNS server is required when push to domain name is used.

The screenshot shows the 'Kernel Maintenance' dialog box with two main sections: 'Current kernel' and 'New kernel'.

Current kernel:

- NAD: 10002
- Version: 3.0.7
- Transfer date: 2019-07-01 14:44:02
- Size: 48752 bytes
- Magic: 27183 (OK)
- CRC: 696Fh
- Hardware model: Cybro-3H
- IEX baud rate: 100kbps
- NAD alias:
- Static IP address
- IP address:
- Subnet mask:
- Gateway:
- DNS server:
- Push
- Period:
- IP or URL:
- Refresh button

New kernel:

- File: N:\Projekt\CyPro\Runtime\kernel.t
- Version: 3.0.7
- Build date: 2019-06-03 23:47:36
- Size: 48752 bytes
- Magic: 27183 (OK)
- CRC: 696Fh
- Hardware model: Cybro-3H
- IEX baud rate: 100kbps
- NAD alias:
- Static IP address 10M
- IP address: 192.168.1.100
- Subnet mask: 255.255.255.0
- Gateway: 192.168.1.1
- DNS server: 8.8.8.8
- Push
- Period:
- IP or URL (:port):
- Load button
- Send button
- Close button

Cybro with static IP is accessible right after power-on. Dynamic address may need a few seconds, and up to a minute if controller is connected in a new network. When DHCP server is not available, Cybro will have an invalid IP address (0.0.0.0).

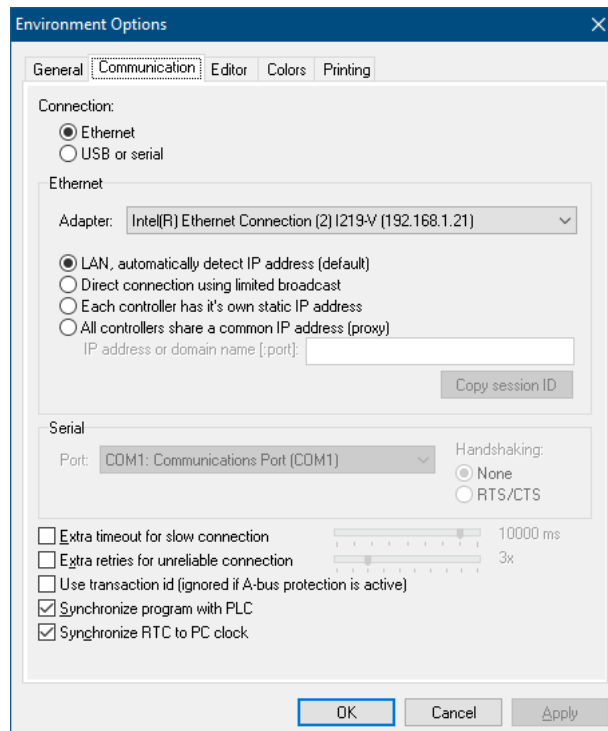
Checkbox [10M](#) is used to disable baud negotiation and force 10Mbps. It may be used when negotiation fails, for whatever reason.

Cybro has 6-byte MAC address in form 00-CB-00-xx-xx-xx, where last three bytes are serial number (NAD). For example, Cybro 20000 (0x4E20) has MAC address 00:CB:00:00:4E:20.

Connection options

There are several ways to connect programming environment and the controller:

- LAN, IP address is detected automatically
- Direct connection using limited broadcast
- Each controller has it's own static IP address
- All controllers share a common IP address (proxy)



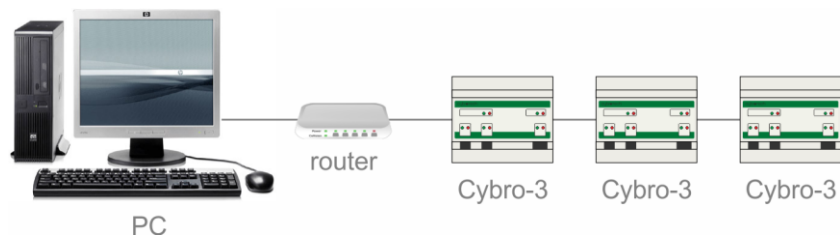
[Session id](#) is used when connection is going through server based on CybroWebScada.

[Extra timeout](#) and [Extra retries](#) may be used when communication channel is slow. [Transaction id](#) adds an unique id to each request/acknowledge pair, avoiding problems with delayed and lost messages. It can't be used if A-bus protection is active.

Recommended settings, depending on network speed:

	roundtrip	transaction id off		transaction id on	
		extra timeout	extra retries	extra timeout	extra retries
local network connection	0..5ms	-	-	-	-
wired internet	10..100ms	200ms	2x	100ms	3x
3G/4G/5G connection	10..200ms	500ms	5x	200ms	5x

[Synchronize program with PLC](#) means the Start button will also send program. [Synchronize RTC to PC clock](#) means the controller real-time clock will be updated when program is sent.

LAN connection

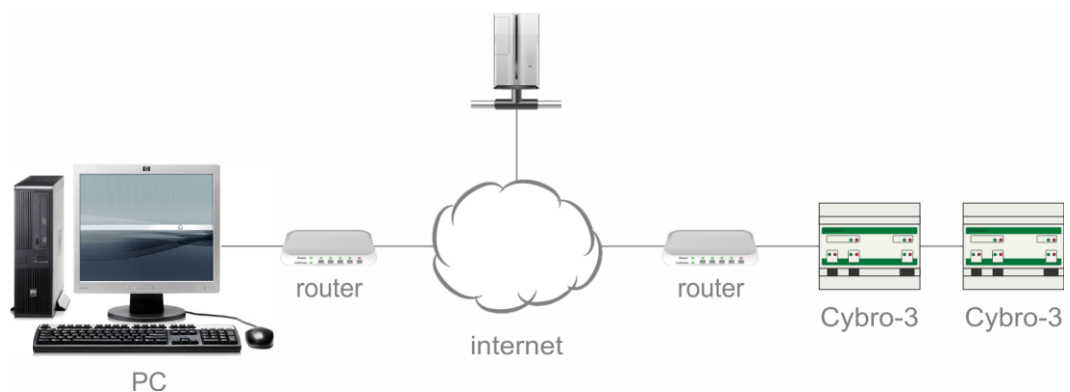
This is the most common setting, all devices are in the same subnet. IP address may be dynamic (DHCP) or static. CyPro uses subnet broadcast (192.168.0.255) to automatically detect IP address.

USB or serial connection

Connect micro USB cable, set environment options to [USB or serial](#), then select port "USB-SERIAL CH340". Connection can be used without power supply. USB provides power supply for CPU, inputs and outputs are inoperative.

Direct connection (no router)

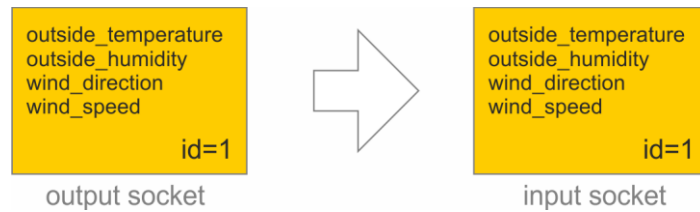
This connection is used in case of emergency, when no valid IP is available. Messages are transmitted as limited broadcast (255.255.255.255:65535). Don't open CyPro before autoconfiguration address (169.254.x.x) is assigned to PC.

Internet connection

Internet connection has to solve two problems: how to get ip address of the other party, and how to get through the router. For more information, check hardware manual, chapter internet. For more details how to set the connection, check the documentation of the tool used.

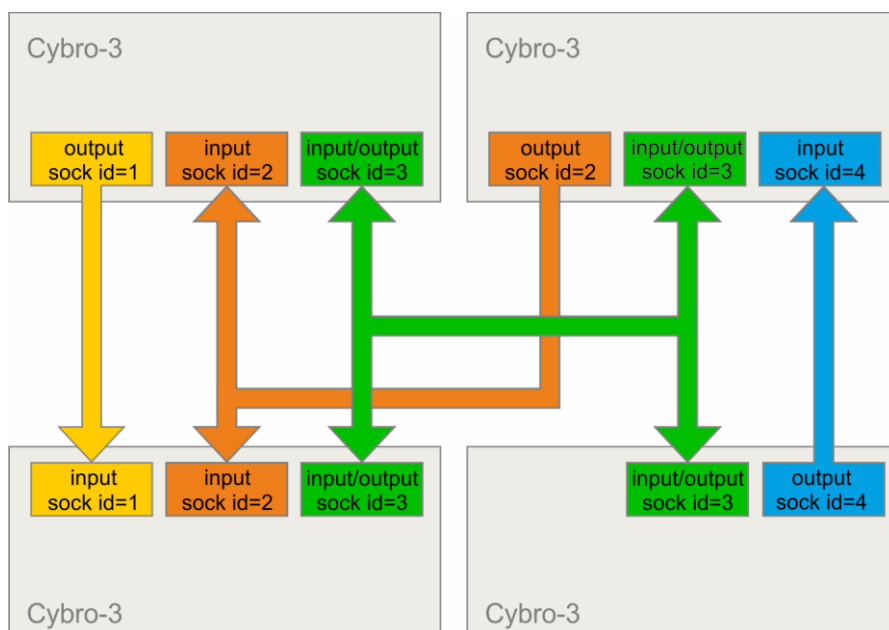
Socket interface

Socket interface is used for cybro-to-cybro communication. A socket is group of variables passed from one controller to another. User defines a matching pair of sockets. Sockets must have the same id and must use the same variables. Type and order matters, name is not important.



Socket id can be in the range 1 to 255.

Multiple sockets can be used at the same time:



Controller receives only sockets defined in its program, all others are ignored.

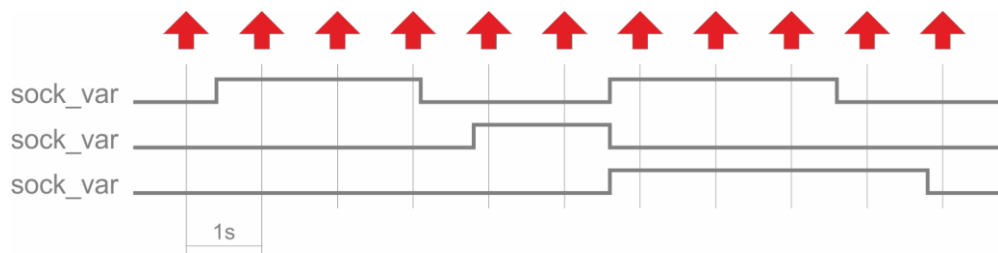
Sender does not know if the receiver actually received the socket. The acknowledge can be sent back through a second socket pair.

Receiver does not know who sent the message, but socket may include sender id as a variable within the socket.

Socket size is limited by the maximum size of A-bus message (1024 bytes).

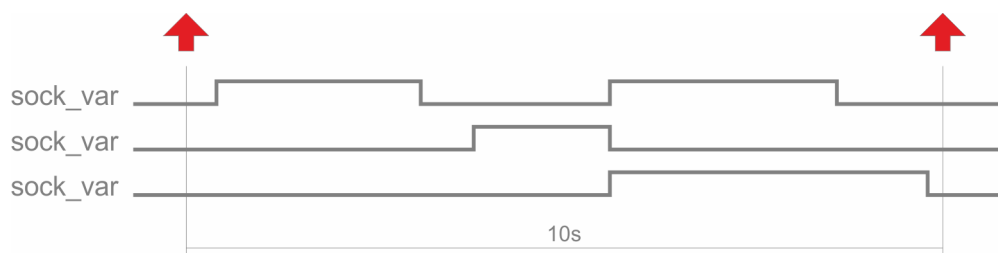
Sending is initiated in several ways:

1. Periodic 1s



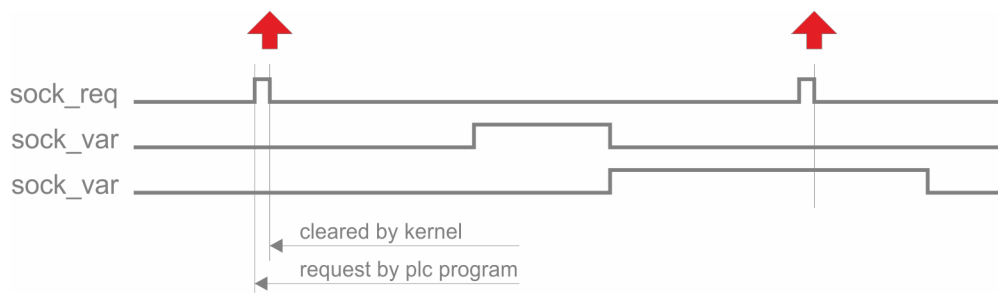
Socket is transmitted once a second.

2. Periodic 10s



Socket is transmitted once every ten seconds.

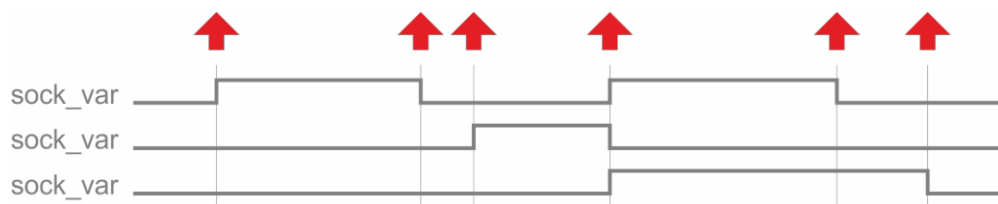
3. On-request



Socket is transmitted on request from plc program.

Transmission begins when request bit is set. Kernel responds by clearing the request and sending the socket. Request is the first bit variable in the socket. It is transmitted as 1, so it can be used by receiver to check if the socket has arrived.

4. On-change

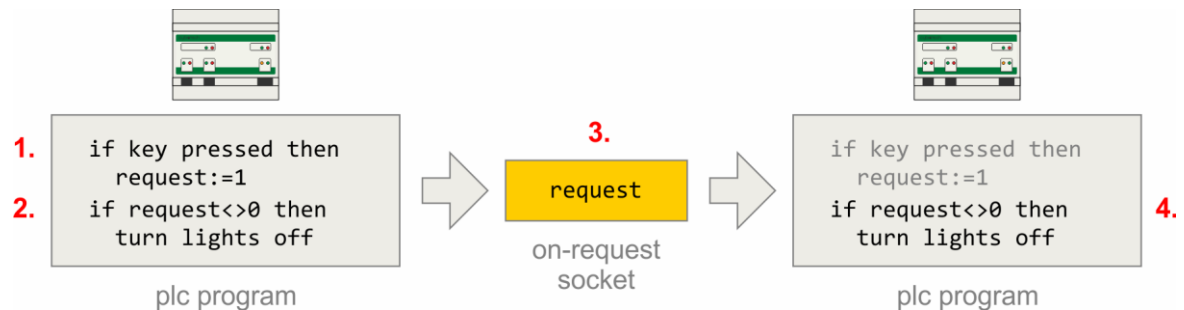


Socket is transmitted each time one of socket variables is changed. Controller must be running.

On-request example

On-request socket may be used to send event to multiple controllers. One controller sends the socket, all others will receive it. Number of controllers is not limited.

The example shows how to turn off lights controlled by two controllers.



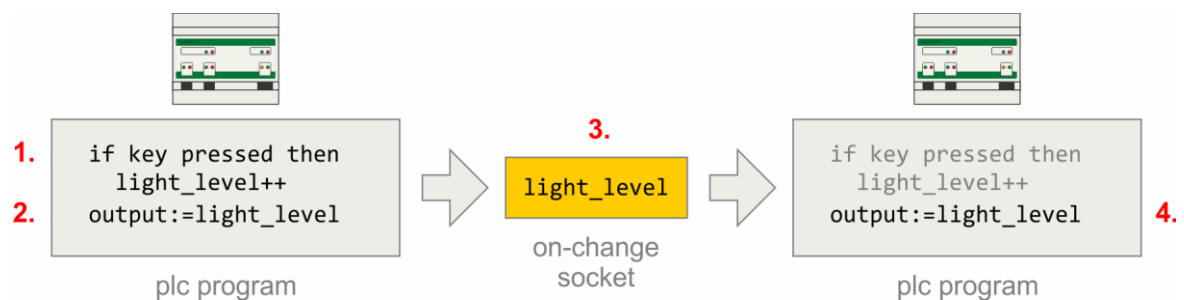
Program in both controllers is the same.

When receiver needs to know request source, 1 is local, 3 is remote (cast to bit when comparing). Request will be active for at least a single scan.

On-change example

On-change socket is used to synchronize a value between controllers. Each controller may modify the value, all others receive the new value. Number of controllers is not limited.

The example shows a light level setting (0-100%), synchronized between controllers.



Each controller has the same program, local i/o assignment may be different.

Features

Real-time clock

Real-time clock (RTC) consist of a hardware clock and calendar. When power is down, it runs from internal battery. For accuracy and data retention time, check hardware manual.

RTC is synchronized to PC when program is sent to the PLC. To enable or disable synchronization, use checkbox [Environment/Communication/Synchronize RTC to PC Clock](#). RTC is also synced with OPC server and HIQ Commander mobile application. It can be set also with PLC program.

To read and write time, use:

```
rtc_hour:int;
rtc_min:int;
rtc_sec:int;
```

hour	0..23
min	0..59
sec	0..59

To read and write date, use:

```
rtc_year:int;
rtc_month:int;
rtc_date:int;
```

year	2000..2099
month	1..12
date	1..31

To read and write day of the week, use:

```
rtc_weekday:int;
```

- 0 - Sunday
- 1 - Monday
- 2 - Tuesday
- 3 - Wednesday
- 4 - Thursday
- 5 - Friday
- 6 - Saturday

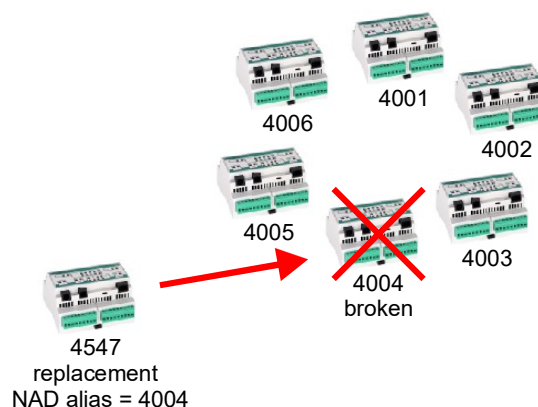
To set real-time clock, write new time/date to variables and set the request flag:

```
rtc_write_req:=1;
```

NAD alias

Each controller has unique serial number, used as communication address (NAD). Serial number is permanent and can not be changed.

NAD alias is a second, replacement address configurable by user. Alias functions same as the original NAD, controller may be addressed both ways.

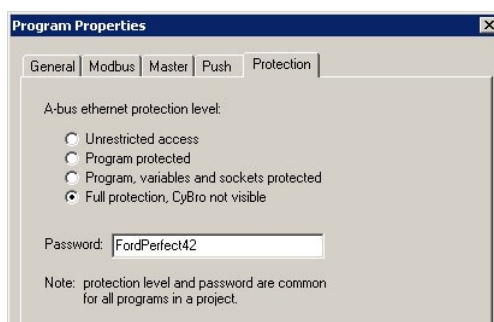


To set new NAD alias, open [Kernel Maintenance](#) dialog box, enter alias and send.

Because of security issues, **alias is used in local communication only**. When controller is connected to the internet, the original serial number is used exclusively.

Password protection

Cybro controller can restrict access to it's data with password. Depending on selected level, protection may cover only program, program and variables, or everything. For example, when protection level is [Program protected](#), anybody can read and write variables, but needs a password to send a new program.



Password protection affect only Ethernet interface. Serial ports are not restricted (including USB), even when full protection is used.

Password may contain any combination of letters and numbers of a reasonable length. It is case sensitive. Don't use spaces or national characters.

Password is common for all programs in project, it's not possible to define individual password for each controller. Password stored in project file is not secure, so keep your project safe.

When password is used, communication option [Transaction id](#) can not be used.

To send a new program to protected controller, use command [Erase protected program](#).

If you forget the password, unlock controller using the USB port.

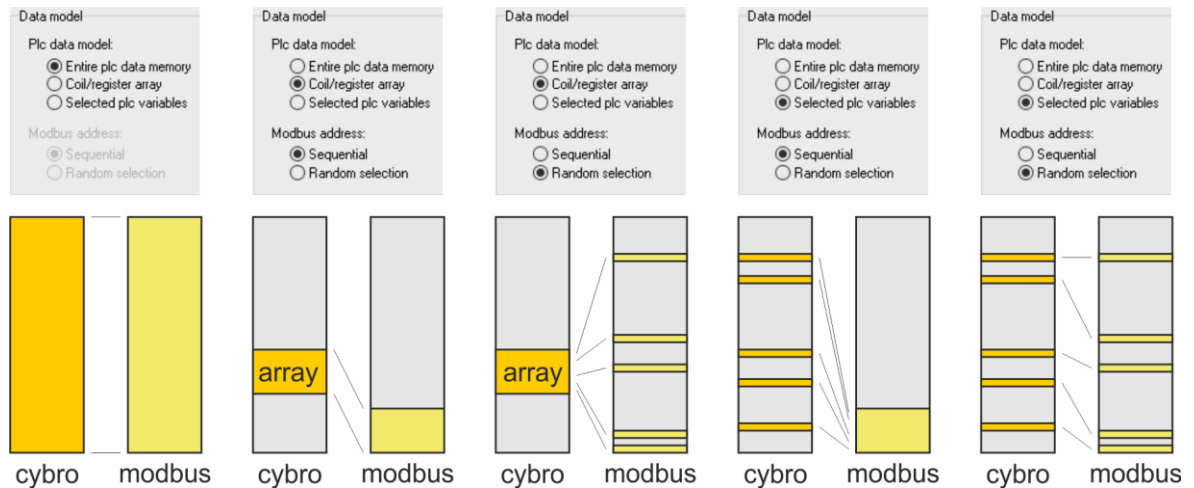
Modbus slave

Modbus communication protocol is published in 1979, for use with programmable logic controllers. It has since become de facto standard for connecting various devices.

Cybro supports:

- Modbus RTU slave (RS232/RS485)
- Modbus TCP slave (Ethernet)

Modbus data model describes how modbus coils and registers are translated to Cybro memory.



Modbus model include coils and holding registers. Discrete inputs and input registers are not supported.

When "Entire plc data memory" is selected, list of available coils/registers can be exported in csv format. List may be imported by modbus master, referring variables by name instead of a number.

Function codes:

code	hex	command
1	01h	READ_COILS
3	03h	READ_HOLDING_REGISTERS
5	05h	WRITE_SINGLE_COIL
6	06h	WRITE_SINGLE_REGISTER
15	0Fh	WRITE_MULTIPLE_COILS
16	10h	WRITE_MULTIPLE_REGISTERS

Other codes will be rejected as ILLEGAL_FUNCTION (exception code 01h).

Data types:

- bit (0 or 1) for coils
- int (16-bit integer) for registers

Other data types are not supported.

When Modbus RTU master is needed, use ModbusRtuMaster.cyp from Examples.

Mobile application

HIQ Commander is mobile app used to monitor, control and configure your plc program.

User should mark the variables, open app and start autodetect. The app displays a list of objects, each representing a single variable. Each object is used to display variable in one of the predefined modes. The mode can be configured with tags, which are entered in the variable description. Object can also be used to change the value, by using the [action](#) tag.

To setup variables, open allocation editor, variable properties, and do the following:

- tick checkbox "visible in smartphone scada"
- enter tags into the variable description

Tags may be placed anywhere within description, and separated by space or comma. Each tag has default setting. When default is alright, the tag doesn't need to be specified.

Make sure the variable is not hidden, and the allocation file is sent to the controller.

The number of objects is not limited.

When plc is configured, open [HIQ Commander](#) and start autodetect. Ensure the mobile is on Wi-Fi, the same network as the controller. If alright, the list of objects will appear.

To use the application remotely, over the internet, ownership of the controller must be confirmed. Ensure the mobile is on Wi-Fi, the same network as the controller, open Settings and press Enable.

For more details, open [HIQ Commander demo.cyp](#) from CyPro examples.

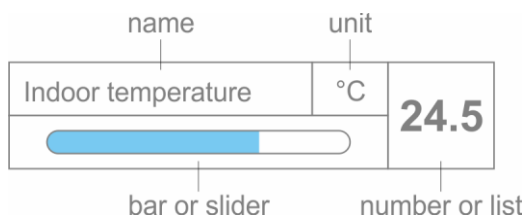
Binary object

name	unit	icon
Relay output	QX0	

Tags available for the binary object:

- type** object type, bit or int (default is bit for bit variables, int for others)
- name** object name (default is variable name)
- unit** short text displayed on the right (default is none)
- icon** icon number, check appendix (default is 0)
- action** 0-none, 1-write, 2-toggle (default is 0)
- value** value that will be written by write action (default is 1)

Integer object



Tags available for the integer object:

- type**..... object type, bit or int (default is bit for bit variables, int for others)
- name** object name (default is variable name)
- unit**..... short text displayed on the right (default is none)
- dec**..... number of digits after decimal point (default is 0)
- list** list of strings, separated by 'or' symbol (OFF|HEAT|COOL) (default is none)
- bar** 0-none, 1-show bargraph (default is 0)
- min**..... minimum for increment, slider, spin edit (default is 0) and keypad (default is none)
- max**..... maximum for increment, slider, spin edit (default is 100) and keypad (default is none)
- step**..... step size for increment, slider and spin edit (default is 1)
- action**..... 0-none, 1-write, 2-toggle, 3-increment, 4-slider, 5-string list, 6-spin edit, 7-keypad (def 0)
- value** value that will be written by write action (default is 1)

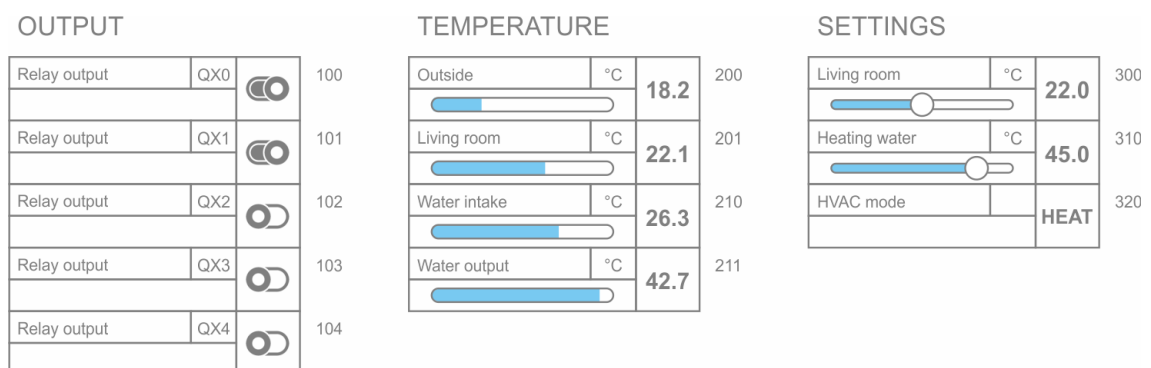
Tags are entered in the description field of the variable. General form is <tag>=<value>. The order and position doesn't matter. String with spaces must be enclosed in quotation marks. There should be no spaces before and after the '=' sign. Object name can be changed within application. Keypad action has no default limits, the limits are applied only when min and max are explicitly stated.

Tabs and order

The order of objects is determined by variable type and position in the allocation list. Bit variables are at the top, followed by integers, long integers and finally floats. To specify the order of objects manually, use the following tags:

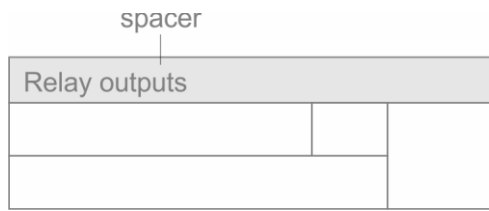
- pages**..... tab names (default is tabs not visible)
- pos**..... position and page number (default is as in the allocation list)

Tag **pages** defines the name for each tab, and consequently the total number of tabs. It should be specified only once, within the first variable. Tag **pos** defines position of object in the list. Numbers can be skipped, which may be useful when objects are added later. The hundreds digit has a special meaning, it defines the tab on which the object is displayed.



```
bit output_qx00: pos=100, action=2, pages=OUTPUT|TEMPERATURE|SETTINGS
bit output_qx01: pos=101, action=2
bit output_qx02: pos=102, action=2
...
int hvac mode: pos=320, action=5, list=OFF|HEAT|COOL
```

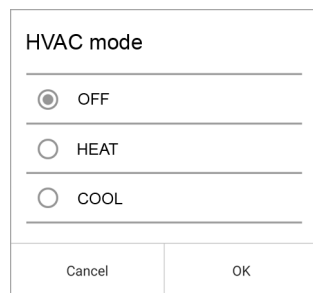
Spacer



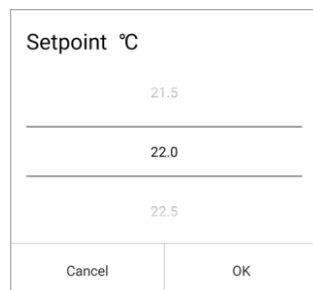
spacer.... visual separator between objects, with optional title (default is none)

Actions

- 0: none object is read only
- 1: write write a single value
- 2: toggle switch between 0 and 1
- 3: increment increment by **step**, positive or negative, loop back when min/max is reached
- 4: slider drag handle left and right to adjust the value
- 5: string list select a single choice from the list (0, 1 or 2)



6: spin edit enter value by turning the wheel



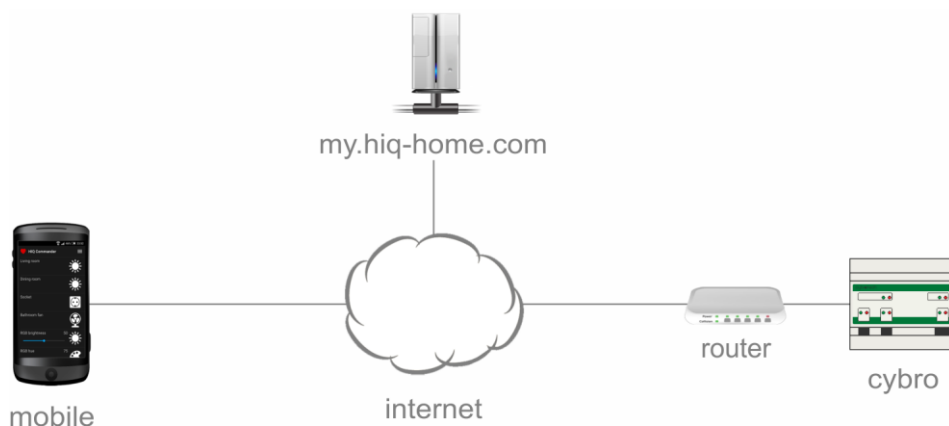
7: keypad enter value digit by digit



When EE variable is modified, EE magic and write request are applied automatically. When RTC variable is modified, RTC write request is applied automatically.

Internet

In a local network, application talks directly to the controller. When mobile is remote, the traffic is routed through the server.



There are two ways to register to the server:

- autodetect, turn on [enable internet access](#) switch
- settings, press internet access [enable](#) button

Mobile must be in the same network as the controller. Both procedures are fully automatic. Server creates a record for both mobile device and controller.

For a better security and access control, you may create the user account. Sign-in to <http://my.hiq-home.com>, then add your controller to the list. Account is not required for internet access.

To do this, you need CyPro online monitor. Set [authentication_req](#), copy [authentication_code](#) and type serial number and the 6-digit number into the online form. Description is optional.

HIQ Universe

damir.skrjanec@cybrotech.hr

Status Controllers Phones Settings Logout

Serial	Description	Created	Last push	Status	Properties	Ping	Enable	Delete
10020	Bohinjska	2022-11-02 12:26	-	●				
24002	Gredička	2016-09-06 03:32	2023-01-09 16:07	●				

Add new controller

robotina
copyright (c) 1998-2023, all rights reserved

[Contact](#)
[Privacy](#)
[Terms](#)

The mobile is automatically visible in the phones list.

The screenshot shows the HIQ Universe mobile application interface. At the top, the title 'HIQ Universe' is displayed in white on a blue background, with the email address 'damir.skrjanec@cybrotech.hr' on the right. Below the title is a navigation bar with tabs for 'Status', 'Controllers', 'Phones', and 'Settings', and a 'Logout' button on the right. The main content area displays a table of registered phones:

Phone	Maker	Model	Carrier	Registered	Last login	Enable	Delete
AGS2-W09	HUAWEI	AGS2-W09		2020-02-12 03:50	2022-11-05 11:22	✓	✗
NEO-U9-H	MINIX	NEO-U9-H		2018-09-29 19:53	2022-02-08 01:39	✓	✗
SM-G991B	Samsung	SM-G991B	A1 HR	2022-11-02 12:20	2023-01-09 16:00	✓	✗

Below the table, there is a checkbox labeled 'Disable new phones' which is checked. At the bottom of the screen, there is a footer with the 'robotina' logo, the text 'copyright (c) 1998-2023, all rights reserved', and links for 'Contact', 'Privacy', and 'Terms'.

To add new phone, run the same procedure again.

To secure the system, disable adding new phones. That will ensure maximum security, nobody will be able to gain control, even if they have access to your local network.

Examples

Relay output	QX0	


```
bit relay_output;
```

description: Click to turn the relay output on and off (name="Relay output", unit=QX0, action=2).

Output power	kW	2.1


```
int output_power;
```

description: Measured output power for all phases (name="Output power", unit=kW, dec=1).

House temperature	°C	24.5
		

```
int temperature;
```

description: Measured temperature (name="House temperature", unit=°C, dec=1, bar=1, min=100, max=300).

Setpoint	°C	22.0
		

int setpoint;
 description: Setpoint temperature (name="Setpoint", unit="°C", dec=1, min=100, max=300, step=5, action=4).

HVAC mode		HEAT

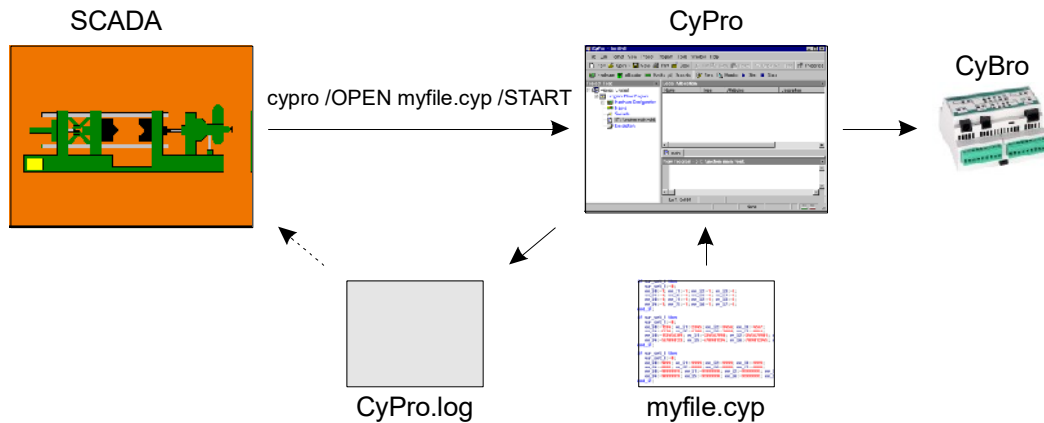
int hvac_mode;
 description: HVAC operating mode (name="HVAC mode", list=OFF|HEAT|COOL, action=5).

Heating and cooling		
HVAC mode		HEAT

int hvac_mode;
 description: HVAC operating mode (spacer="Heating and cooling", name="HVAC mode", list=OFF|HEAT|COOL, action=5).

Command line options

Command line options are specified upon starting CyPro. They are used to automatically perform some tasks, such as sending a program. Using command line options, CyPro may be used as external compiler for another application.



Command line options are:

<code>/NEW [filename.cyp]</code>	Create a new project. Filename is optional.
<code>/OPEN filename.cyp</code>	Open existing project with specified filename.
<code>/SAVE</code>	Save project.
<code>/SAVEAS filename.cyp</code>	Save project under specified name.
<code>/EXIT</code>	Exit CyPro.
<code>/NAD number</code>	Select program. If specified NAD exists, that program will be selected, otherwise NAD is appended to current program.
<code>/AUTODETECT</code>	Hardware autodetect.
<code>/START</code>	Compile, send (only if different) and run.
<code>/STARTALL</code>	Start all programs in project.
<code>/STOP</code>	Stop current program.
<code>/SEND</code>	Send current program.
<code>/HIDDEN</code>	Silent operation, do not show any window or dialog box.

Filename may be given as name or full path. When file name contain spaces, use double-quote ("my file.cyp"). If an operation requires user input to continue execution, default option is used automatically. For example, when autodetect asks for a network address, default address (zero) will be used automatically.

When started with command line options, CyPro creates log file with all commands and results. Log file is saved in CyPro directory (c:\Program Files (x86)\Cybrotech\CyPro-3\CyPro.log).

When `/HIDDEN` mode is used, CyPro will automatically exit after executing the last command.

When using command line options, it is advisable to turn on checkbox [Allow multiple instances](#) in [Environment Options](#). If only single instance is allowed and CyPro is already running, command line requests will be proceeded to the active copy.

Examples:

```
cypro.exe myfile.cyp
```

Start CyPro and open project myfile.cyp.

```
cypro.exe "c:\My Documents\myfile.cyp"
```

Start CyPro and open project myfile.cyp in specified directory. As path may contain spaces, quotas are required.

```
cypro.exe /HIDDEN /OPEN "myfile.cyp" /START /EXIT
```

Start CyPro, open an existing project (myfile.cyp), start PLC (compile, send & run) and exit. Operation is hidden, no window or dialog box will appear. Possible errors are saved in CyPro.log.

```
cypro.exe /HIDDEN /NEW /AUTODETECT /SAVEAS "myfile.cyp" /EXIT
```

Start CyPro, open a new project, start Autodetect, save as myfile.cyp and exit. Operation is hidden, no window or dialog box will appear. Possible errors are saved in CyPro.log.

```
cypro.exe /HIDDEN /NEW /NAD 4000 /AUTODETECT /SAVEAS "myfile.cyp" /EXIT
```

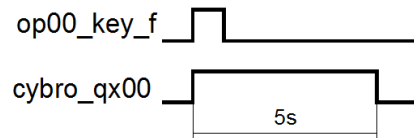
Start CyPro, open a new project, add new NAD, start Autodetect to detect connected IEX-2 modules, save as myfile.cyp and exit. Operation is invisible, no window or dialog box appears. Possible errors are saved in CyPro.log.

```
cypro.exe /HIDDEN /OPEN "myfile.cyp" /AUTODETECT /START /EXIT
```

Start CyPro, open an existing project (myfile.cyp), start Autodetect (assuming the project has no hardware setup and network address), start PLC (compile, send & run) and exit. Original file remain unchanged. Operation is silent, no window or dialog box will appear. Errors are saved in log file.

Getting started

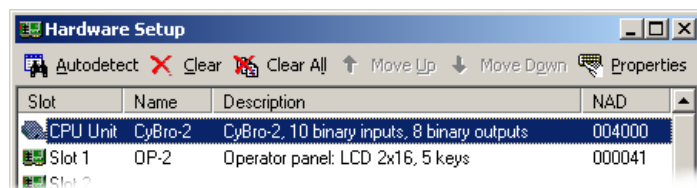
This example, a simple timer activated with a key, will show steps to get program running.



Step 1: hardware

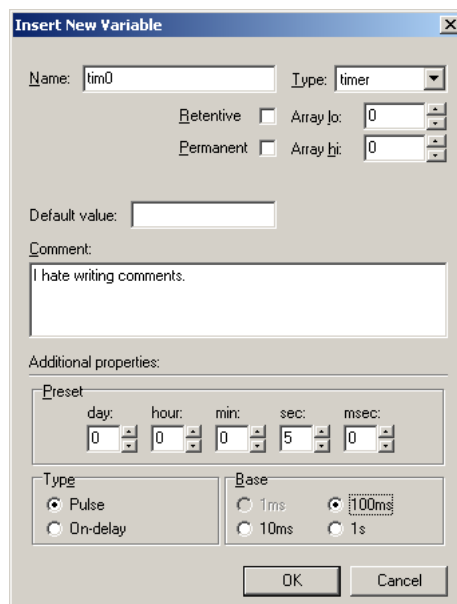
The example will use Cybro controller and OP-2 panel. Connect power supply, ethernet and panel according to hardware manual.

Open CyPro and select [File/New Project](#). Open [Hardware Setup](#) and run [Autodetect](#).



Step 2: variables

Project will use variable of timer type. Start [Allocation Editor](#), and press [Insert](#):



Enter name, select type, preset and time base.

Step 3: write code

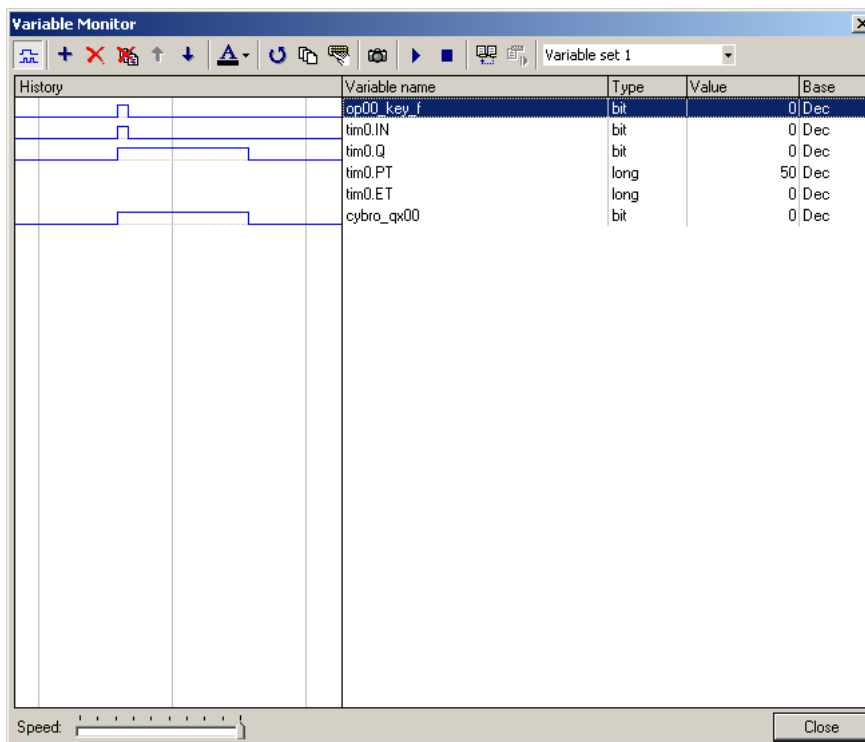
PLC code connects the OP key to the timer input, and the timer output to the output relay:

```
tim0.in:=op00_key_f;
cybro_qx00:=tim.q;
```

Step 4: run

To compile and send program, just press **Start** button. Status line shows the program is running.

To check operation, open **Variable Monitor**, add variables, and press **F** key.



Graph in the left pane shows the program is running as expected.

Appendix

Data type summary

Elementary

type	width	range
bit	1-bit	0..1 (*)
integer	16-bit signed	-32768..32767
long	32-bit signed	-2147483648..2147483647
real	32-bit single precision	-3.4x10 ³⁸ ..3.4x10 ³⁸

(*) each bit variable is stored as a byte, casting to bit allows 0..255 range

Input/Output

type	width	type	description
ix	1-bit	bit	digital input
qx	1-bit	bit	digital output
iw	16-bit	integer	analog input
qw	16-bit	integer	analog output

Timer

field	type	access	description
in	bit	read write	control input
q	bit	read write	timer output
pt	long	read write	preset time
et	long	read write	elapsed time

Constants

decimal

```
address := 12345; // 16 or 32-bit signed integer
```

binary

```
address := 2#10111; // 16-bit signed integer
```

hexadecimal

```
address := 16#FFFF; // 16-bit signed integer
```

Structured text summary

Operators

operator	alias	unary	binary	function	bit	int	long	real	result
+			•			•	•	•	same
-		•	•			•	•	•	same
*			•			•	•	•	same
/			•			•	•	•	same
mod	%		•			•	•		same
not	!	•		•	•	•	•		same
and	&		•		•	•	•		same
or			•		•	•	•		same
xor	^		•		•	•	•		same
shl, shr	<< >>		•			•	•		same
rol, ror			•			•	•		same
=	==		•		•	•	•	•	bit
<>	!=		•		•	•	•	•	bit
<, <=			•			•	•	•	bit
>, >=			•			•	•	•	bit
:=			•		•	•	•	•	same

Flow control

if...then...else

```
if <expression> then
  <statements>;
elseif <expression> then
  <statements>;
else
  <statements>;
end_if;
```

case...of

```
case <expression> of
  <value>: <statements>;
  <value>: <statements>;
else
  <statements>;
end_case;
```

for...do

```
for <var>:=<expression> to <expression> do
  <statements>;
end_for;
```

while...do

```
while <expression> do
  <statements>;
end_while;
```

Edge detect

positive edge detect (zero to one)

```
fp(b:bit):bit;
```

negative edge detect (one to zero)

```
fn(b:bit):bit;
```

Type conversion

evaluate expression and convert to desired data type

```
int(expression):int; // convert expression to integer
long(expression):long; // convert expression to long, respect sign
ulong(expression):long; // convert expression to long, assume unsigned
real(expression):real; // convert expression to float
blong(expression):long; // assume bit pattern as long, no conversion
breal(expression):real; // assume bit pattern as real, no conversion
```

Serial ports

port select

```
com_select(port:int); // 1-COM1, 2-COM2, 3-COM3, 4-RFM, 5-ETH, 6-CAN
```

transmit

```
tx_bufwr(pos:int, data:int); // write data byte to tx buffer
tx_bufrd(pos:int):int; // read data byte from tx buffer
tx_start(size:int); // send message
tx_stop(); // stop sending
tx_count():int; // number of characters sent
tx_active():bit; // 0-stopped, 1-transmitting
```

receive

```
rx_start(beg_ch:char, end_ch:char, len:int, beg_tout:int, end_tout:int); // COM
rx_start(dummy:char, dummy:char, group_hi:int, group_lo:int, timeout:int); // RFM
rx_start(protocol:char, dummy:char, port:int, autostop:int, timeout:int); // ETH
rx_start(dummy:char, dummy:char, dummy:int, dummy:int, dummy:int); // CAN
rx_stop(); // stop receiving
rx_count():int; // number of characters received
rx_active():bit; // 0-stopped, 1-receiving
rx_status():int; // 0-receiving, 1-stopped, 2-end char, 3-length, 4-timeout
```

parse received message

```
rx_bufrd(pos:int):int; // read data byte from rx buffer
rx_bufwr(pos:int, data:int); // write data byte to rx buffer
rx_strcmp(pos:int, str:string):bit; // compare rx buffer with string
rx_strpos(pos:int, str:string):int; // find string in rx buffer
rx_strtoi(pos:int):int; // read number from rx buffer
rx_strtol(pos:int):long; // read number from rx buffer
rx_strtor(pos:int):real; // read number with decimals from rx buffer
```

Display functions

```
dclr(slot:int); // clear display
dprnc(slot:int, x:int, y:int, c:char); // print character
dprns(slot:int, x:int, y:int, str:string); // print string
dprnb(slot:int, x:int, y:int, c0:char, c1:char, value:bit); // print c0 or c1
dprni(slot:int, x:int, y:int, width:int, zb:bit, value:int); // print integer number
dprnl(slot:int, x:int, y:int, width:int, zb:bit, value:long); // print long number
dprnr(slot:int, x:int, y:int, width:int, dec:int, value:real); // print decimal number
```

Legend:

slot..... slot number (0-write to selected serial buffer)
 x..... x position (0-left)
 y..... y position (0-top)
 width number of characters to print
 zb..... zero blanking (0-no, 1-yes)
 dec..... number of decimal places
 c..... single character
 str array of characters enclosed in single quotes
 value data to print

Network functions

```
get_nad():long; // read current A-bus address (alias or serial)
get_serial():long; // read controller serial number
get_ip():long; // read controller IP address
set_ip(ip_address:long, subnet:long, gateway:long, dns_server:long); // set IP address
```

Return value

return value from a function

```
result := a + b; // return sum of a and b
```

Program examples

Library

Ready-made application or set of functions, that can be invoked to carry out the particular task. Generally, library functions are used as they are, without modifying the code.

CybroDashboard	demonstration of controller features and quick test of main components
DaliConfigurator	assign short addresses, configure groups and set parameters
EnOceanGateway	gateway for EnOcean wireless devices, including configuration and usage
FunctionLibrary	collection of standard functions used to carry out common tasks

Template

Fully functional application that can be modified and included in the user program.

AccessControl	reception desk, manage access for hotel rooms and spaces
DaliControl	use cybro controller to control DALI ballasts
DaliControl DT8	control DALI DT8 RGB ballast, template for multiframe messages
HIQ Commander demo	use mobile phone to control cybro application over the internet
HTTP client	read variables from www.solar-cybro.com server
HTTP server	cybro controller as a simple web server, implementing HTTP protocol
ModbusRtuMaster	read power meter registers using serial communication
ModbusTcpMaster	fully functional application to read/write data from multiple slaves
RFM demo	configure cybro wireless devices and control WR-1 or WR-5 relay
TCP demo	send and receive custom TCP messages between two controllers
UDP demo	send and receive custom UDP messages between two controllers

Hardware demo

Fully functional application that shows how to use the particular hardware.

DmxController	control professional lighting using COM-DMX module
ModbusRtuMaster w COM-MB	read power meter registers using COM-MB module
PhilipsWizControl	control Philips WiZ light bulb dimming using free programmable UDP port
Serial port w COM-PGM	free programmable serial port using COM-PGM module

Demo program

Short demonstration how a particular task can be implemented.

DigitalFiltering	remove noise and create a smooth output response
PidController	simple implementation of PID (proportional integral derivative) controller
MaskDemo	shows how to enter parameters using the operator panel
MsTimerDemo	how to implement precise 1ms resolution timer
SetIpAddress	set controller IP address using PLC program
SocketDemo	connect two or more controllers using cybro sockets
SosBuzzer	send SOS message using Morse code
Sun position	calculate if sun is visible for given date, time and location on the globe

Function library

Function library is a collection of commonly used functions, written in structure text. It is a part of CyPro package, located in [\CyPro\Examples\FunctionLibrary.cyp](#). To use a function, copy and paste from library (right click project tree) to your program. For more details, check function source.

bit manipulation

```
int_to_long(lo,hi: int):long; // two 16-bit integers into a single long
long_to_real(x: long):real; // bit-to-bit copy, without conversion
real_to_long(x: real):long; // bit-to-bit copy, without conversion
byte_to_real(byte3, byte2, byte1, byte0: int):real; // four bytes into float
ip_to_long(ip3, ip2, ip1, ip0: int):long; // four byte ip address into a single long
datetime(year, month, date, hour, min, sec: int):long; // 32-bit ms-dos datetime
```

elementary functions

```
abs(x: int):int; // absolute value of integer
min(x, y: int):int; // smaller of two integers
max(x, y: int):int; // bigger of two integers
round(x: real):real; // round to the closest integer
frac(x: real):real; // return fractional part
sqrt(x: real):real; // square root
```

trigonometric functions

```
sin(x: real):real; // sine of x
cos(x: real):real; // cosine of x
atan(x: real):real; // arctangent of x
atan2(x, y: real):real; // arctangent of x/y
```

exponential and logarithmic

```
exp(x: real):real; // exponential of x
ln(x: real):real; // natural logarithm of x (base e)
log10(x: real):real; // logarithm of x with base 10
log(x, base: real):real; // logarithm of x with given base
```

cyclic redundancy check

```
crc8(len: int):int; // 8-bit cyclic redundancy check
crc16(len: int):int; // 16-bit cyclic redundancy check
crc32(len: int):long; // 32-bit cyclic redundancy check
```

pseudo-random generator

```
rnd(range: int):int; // simple pseudo-random generator
```

other functions

```
display bargraph(slot,x,y,width,min,max,val: int):void; // OP semi-graphic bargraph
```


Instruction list summary

Move

ld	move variable or constant to accumulator
ldn	move complement of variable to accumulator
st	move accumulator to variable
stn	move complement of accumulator to variable
set	set accumulator or variable
setc	if condition true set variable
res	clear accumulator or variable
resc	if condition true clear variable

Logic

cpl	complement accumulator or variable
and	logical and accumulator with variable or constant
andn	logical and accumulator with complement of variable or constant
or	logical or accumulator with variable or constant
orn	logical or accumulator with complement of variable or constant
xor	exclusive or accumulator with variable or constant
xorn	exclusive or accumulator with complement of variable or constant
shl	shift left accumulator, set LSB to zero
shr	shift right accumulator, set MSB to zero
rol	rotate left accumulator, copy MSB to LSB, 32-bit only
ror	rotate right accumulator, copy LSB to MSB, 32-bit only
fp	detect positive flank, accumulator only
fn	detect negative flank, accumulator only

Arithmetic

neg	change sign of accumulator
add	add variable or constant to accumulator
sub	subtract variable or constant from accumulator
mul	multiply accumulator with variable or constant
div	divide accumulator with variable or constant
mod	remains of dividing accumulator with variable or constant

Compare

eq	test if accumulator equal to value
ne	test if accumulator not equal to value
gt	test if accumulator greater than value
ge	test if accumulator greater or equal value
lt	test if accumulator lower than value
le	test if accumulator lower or equal value

Branch

jmp label	unconditional jump to position indicated by label
jmpc label	jump if condition true
jmpnc label	jump if condition not true
cal subroutine	call subroutine
calc subroutine	call subroutine if condition is true
calnc subroutine	call subroutine if condition is not true

Type combinations

	bit	int	long	real	acc	const	var
ld	+	+	+	+		+	+
ldn	+						+
st	+	+	+	+			+
stn	+						+
set	+				+		+
setc	+						+
res	+				+		+
resc	+						+
cpl	+				+		+
and	+	+	+			+	+
andn	+					+	+
or	+	+	+			+	+
orn	+					+	+
xor	+	+	+			+	+
xorn	+					+	+
shl		+	+		+		
shr		+	+		+		
rol			+		+		
ror			+		+		
fp	+				+		+
fn	+				+		+
neg		+	+	+	+		
add	+	+	+	+		+	+
sub	+	+	+	+		+	+
mul	+	+	+	+		+	+
div		+	+	+		+	+
mod		+	+			+	+
eq	+	+	+	+		+	+
ne	+	+	+	+		+	+
gt		+	+	+		+	+
ge		+	+	+		+	+
lt		+	+	+		+	+
le		+	+	+		+	+
jmp						+	
jmpc						+	
jmpnc						+	
cal						+	
calc						+	
calnc						+	
x-to-y	+	+	+	+	+		

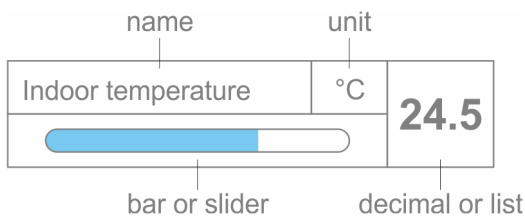
Mobile app tags

Binary object



type object type, bit or int (default is bit for bit variables, int for others)
name object name (default is variable name)
unit short text displayed on the right (default is none)
icon icon number, check appendix (default is 0)
action 0-none, 1-write, 2-toggle (default is 0)
value value that will be written by write action (default is 1)

Integer object

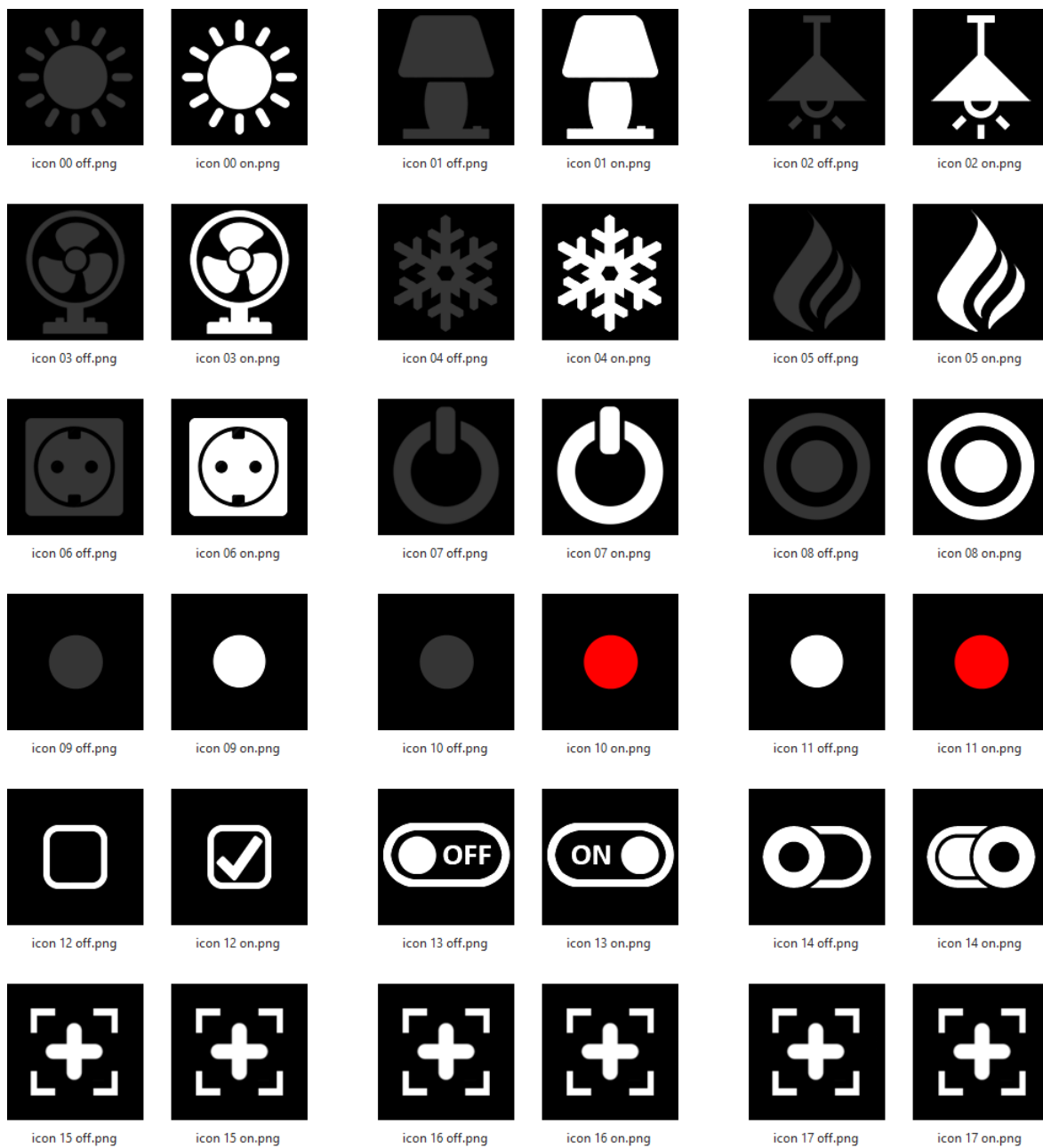


type object type, bit or int (default is bit for bit variables, int for others)
name object name (default is variable name)
unit short text displayed on the right (default is none)
dec number of digits after decimal point (default is 0)
list list of strings, separated by 'or' symbol (OFF|HEAT|COOL, default is none)
bar 0-none, 1-show bargraph (default is none)
min minimum for increment, slider, spin edit (default is 0) and keypad (default is none)
max maximum for increment, slider, spin edit (default is 100) and keypad (default is none)
step step size for slider and spin edit (default is 1)
action 0-none, 1-write, 2-toggle, 3-increment, 4-slider, 5-string list, 6-spin edit, 7-keypad (def 0)
value value that will be written by write action (default is 1)

Pages and order

spacer visual separator between objects in the list (default is none)
pages tab names (OUTPUT|TEMPERATURE|SETTINGS, default is no tabs)
pos position and page number (100, 101, 102, 200, 201, 300, 301...)

Mobile app icons



Operator panel characters

High Low	0	2	3	4	5	6	7	A	B	C	D	E	F
0	10	00	P`	P	-	9	3	0	p				
1	11	!	1	0	0	a	q	7	7	4	ä	q	
2	12	"	2	B	R	b	r	'	ı	ı	ı	ı	ı
3	13	#	3	C	S	c	s	ı	ı	ı	ı	ı	ı
4	14	\$	4	D	T	d	t	\	ı	ı	ı	ı	ı
5	15	5	E	U	e	u	.	ı	ı	ı	ı	ı	ı
6	16	&	6	F	U	f	v	ı	ı	ı	ı	ı	ı
7	17	'	7	G	W	g	w	ı	ı	ı	ı	ı	ı
8		(8	H	X	h	x	ı	ı	ı	ı	ı	ı
9)	9	I	Y	i	y	ı	ı	ı	ı	ı	ı
A		*	:	J	Z	j	z	ı	ı	ı	ı	ı	ı
B		+	:	K	L	k	l	ı	ı	ı	ı	ı	ı
C		,	<	L	ı	ı	ı	ı	ı	ı	ı	ı	ı
D		-	=	M	ı	ı	ı	ı	ı	ı	ı	ı	ı
E		.	>	N	ı	ı	ı	ı	ı	ı	ı	ı	ı
F		/	?	0	_	ı	ı	ı	ı	ı	ı	ı	ı

To enter character code, press **Alt**, type decimal character code preceded by 0, then release **Alt**. Numeric keypad should be used, Num Lock should be on.

Example:

According to table, symbol ° (degrees centigrade) hexadecimal code is DF, which is 223 decimal.

To enter the symbol:

- make sure num lock is on
- press **Alt**
- press consecutively **0223**
- release **Alt**

Because of the character set, monitor displays "ß" character, but the LCD will show correctly.

```
dprns(1,0,0,'T=xx.xBC');
dprnr(1,2,0,4,1,cybro_temperature*0.1);
```

The image shows a blue LCD display with white text. The text reads "T = 24.7 °C". The characters are displayed in a monospaced font. The background of the display area is a grid of blue squares.

Codes 0..7 are reserved for bar-graph and national characters.

Keyboard shortcuts

General

F1		Help
F2		Syntax check
F4		Program settings
Shift-F4		Environment settings
F5		Hardware setup
F6		Allocation editor
F7		Mask editor
F8		Socket editor
F9		Send program to controller
Ctrl-F9		Send without initializing variables
F10		Open online monitor
F11		Start PLC program
F12		Stop PLC program
Ctrl-F12		Pause PLC program
Ctrl-O		Open project
Ctrl-S		Save project
Ctrl-Shift-S		Save As
Ctrl-D		Connect/disconnect communication
Ctrl-L		Select NAD
Ins		Context sensitive insert
Delete		Context sensitive delete
Ctrl-Up		Move item up
Ctrl-Dn		Move item down
Ctrl-Tab		Next window
Ctrl-Shift-Tab		Previous window
Ctrl-F4		Close window
Alt-F4		Exit program

Text editor

Ctrl-space		Insert variable or function
Ctrl-Z	Alt-Backspace	Undo
Shift-Ctrl-Z		Redo
Ctrl-X	Shift-Del	Cut
Ctrl-C	Ctrl-Insert	Copy
Ctrl-V	Shift-Insert	Paste
Ctrl-A		Select all
Ctrl-F		Find
F3		Find next
Ctrl-R		Replace
Ctrl-G		Go to line
Ctrl-Shift-I		Indent selection
Ctrl-Shift-U		Unindent selection
Ctrl-Shift-C		Comment/uncomment selection