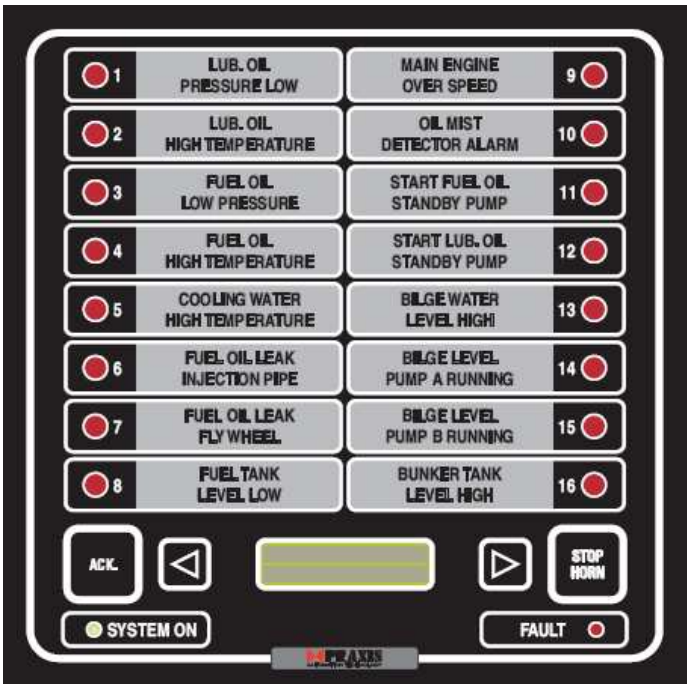
MINI-GUARD
SHIP ALARM & CONTROL SYSTEM



CPDev

PROGRAMMING INSTRUCTION

Controllers:

- Alarm Panel
- Nav Light Control
- Wiper Control
- Fire Alarm
- Battery Monitoring
- PMS Control
- Tank Monitoring

CONTENTS

September 2009

Rzeszow University of Technology, Poland
CPDev development has been supported by MNiSW R02 058 03 grant.

CPDev INSTALLATION

Operating system

Windows (32 bit) Vista/XP/2000/98 SE

Microsoft prerequisites

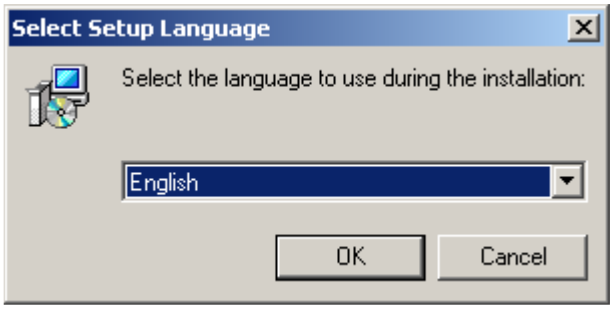
Microsoft .NET Framework 2.0
Microsoft Visual C++ 2005 (SP1) Redistributable

> *Remark.* .NET Framework SP1 is not available for Windows 98. Under Windows 98 CPDev functionality is restricted a little.

Installation program

cpdev-praxis-1.0.1.13.exe, possibly with updated number.

Language selection



The language applies for the setup and application folders. Interface language is initially chosen according to the following table. It can be changed in *Global settings* (*Tools* menu).

Installation	Interface
English	English
Polish	Polish
Dutch	English

Installation steps

Typical for Windows programs.

Installation options

Reset configuration … restores default configuration (*Environment options* in *Tools*). Reset is necessary if the following items have been changed:
1) Installation language (directory names changed)
2) Recipient group (Praxis, Lumel, Univ)

> *Remark.* Screen windows presented here correspond to 96 dpi (normal font). They may be slightly different for other sizes.

Uninstall

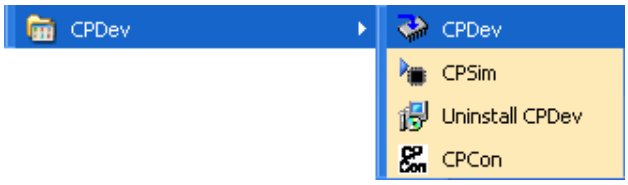
Available from Start menu.

Uninstallation does not remove user configuration files from *Local/ApplicationData* folder (see ev. CSIDL Values in Microsoft documentation).

STARTUP. MENU. TOOLS

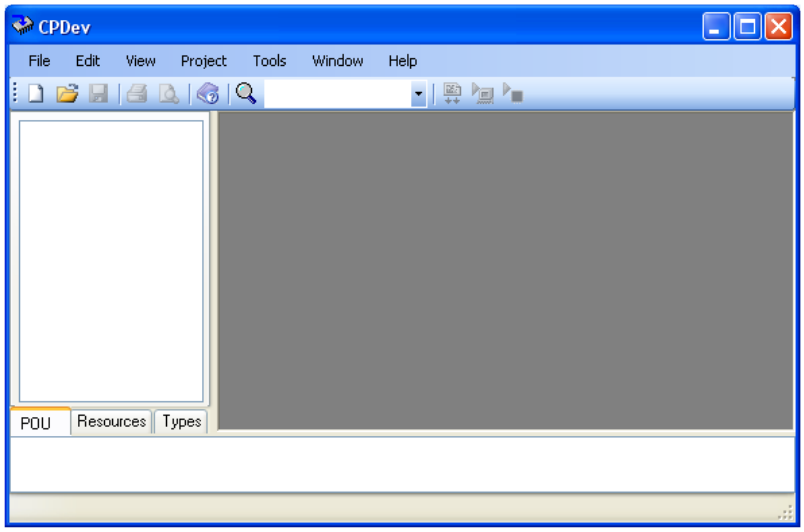
Startup

CPDev starts automatically if *Launch CPDev* is selected during installation. Start menu or desktop icon trigger standard startups.

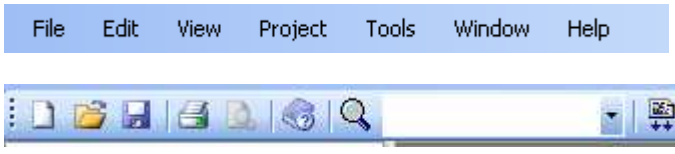


Remark. Nonstandard startups with additional parameters can be executed from directory in which CPDev is installed. Otherwise error of loading external modules appears.

The startup displays CPDev interface window whose left part will present project tree, middle one program code, and bottom part compiler messages.



Menu and toolbar



 New

 Open

 Save

 Print

 Print preview

 Help

 Find text

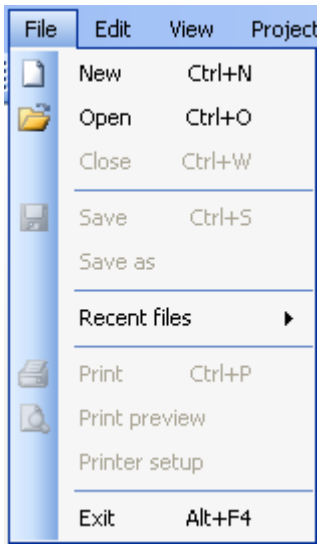
 Build

 Start simulator

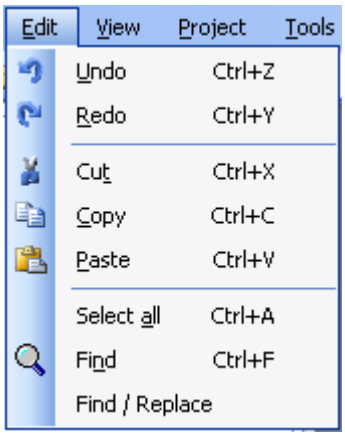
 Start configurer

Manu and toolbar functionality is typical for Windows programs.

File

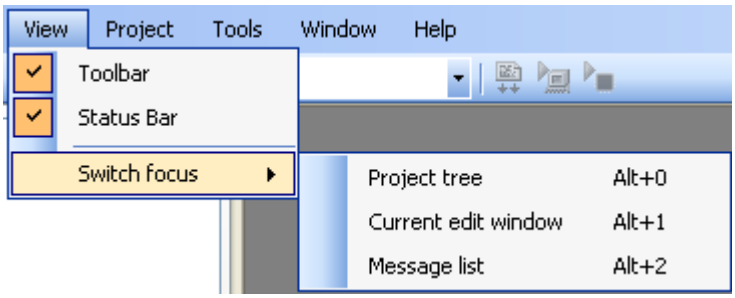


Edit



Some of the items remain inactive until a project is open. *Print* prints project report and source codes (print preview has not been implemented yet.) *Copy* and *Paste*, besides standard text operations, handle items from project tree (POU units, global variables, etc.). *Find* looks for text written in the toolbar cell.

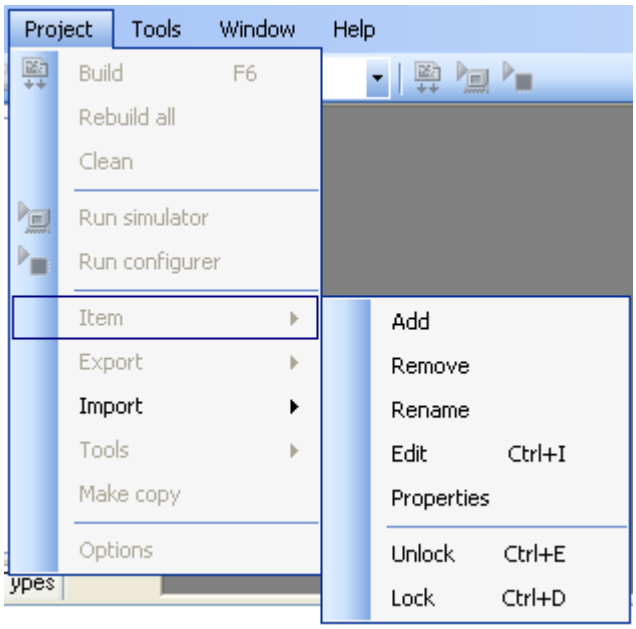
View



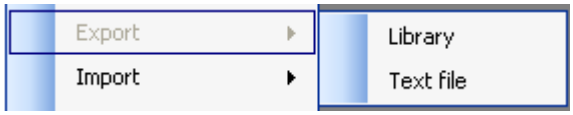
Press Alt+0 to get quickly to project tree, Alt+1 to program window, and Alt+2 to message list.

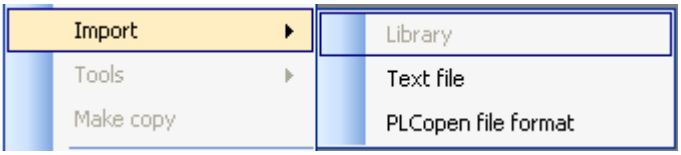
Project

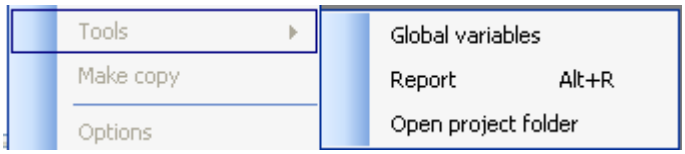
The option handles final stages of the project. *Build* compiles open project or its element. *Clear* removes intermediate files created automatically during compilation, leaving only two necessary (*.xml*, *.xmc*; see *Supplements*). Simulator and hardware configurer can be run after compilation. *Item* adds, removes, etc. project elements.



Export, *Import* deal with libraries (*.lcp*) or external files with ST programs (*.cst*). *Tools* edit list of global variables, present compilation report, and open project folder in Windows explorer.

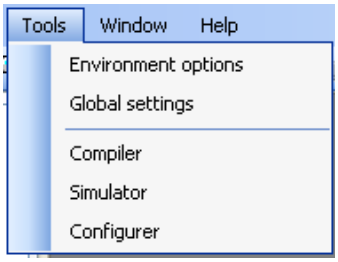




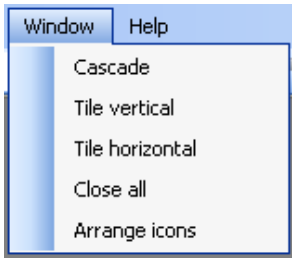


Contents of the last three options look as follows:

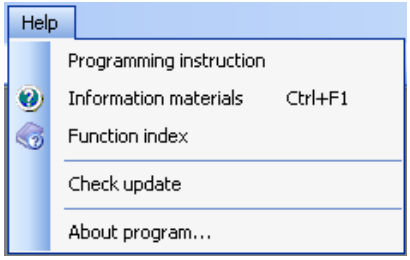
Tools



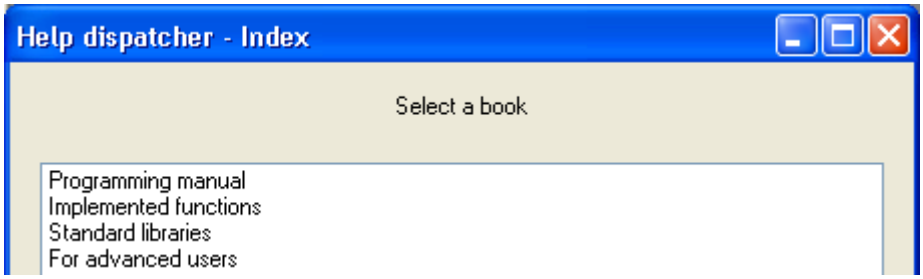
Window



Help

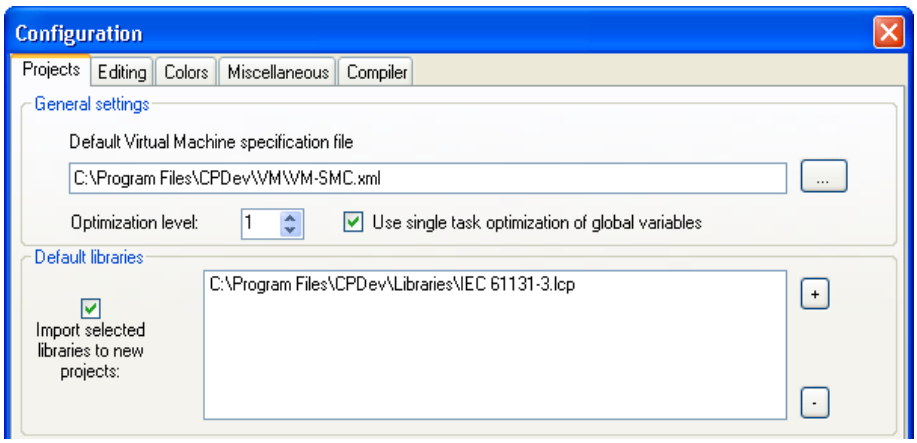


Tools configure environment, determine global settings, and run compiler, simulator and configurer standalone for working with external files (*.cst*, *.dcp*, *.dcp* or *.xmc*). *Window* arranges interface. *Help* accesses programming instruction, information materials with function, function blocks, and notes *For advanced users*. It also indicates whether CPDev has been updated.



Environment options

Configuration window with a few tabs is displayed.



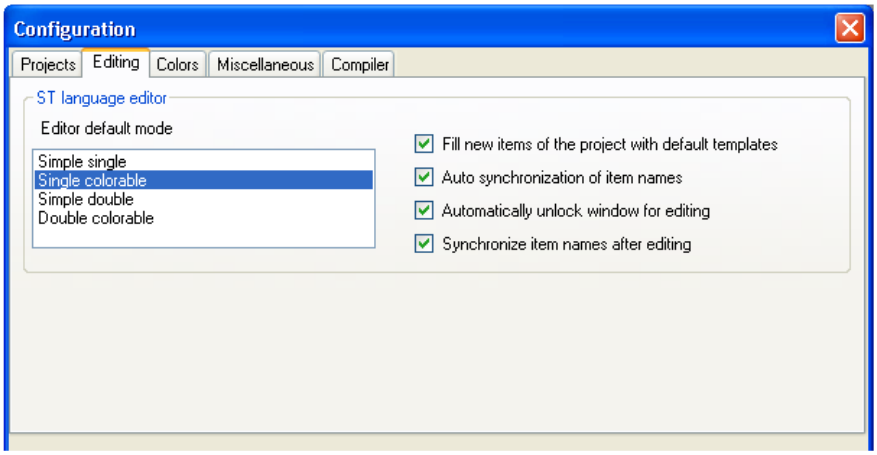
Projects

Path to a file with Virtual Machine specification (*runtime*) is provided. *Use*… option must remain selected (default) for single task VM. Optimization level 1 is normal (ev. see *For advanced users*).

The tab also indicates which libraries should be automatically imported into new projects. Button + adds library from *Libraries* folder. - removes selected library.

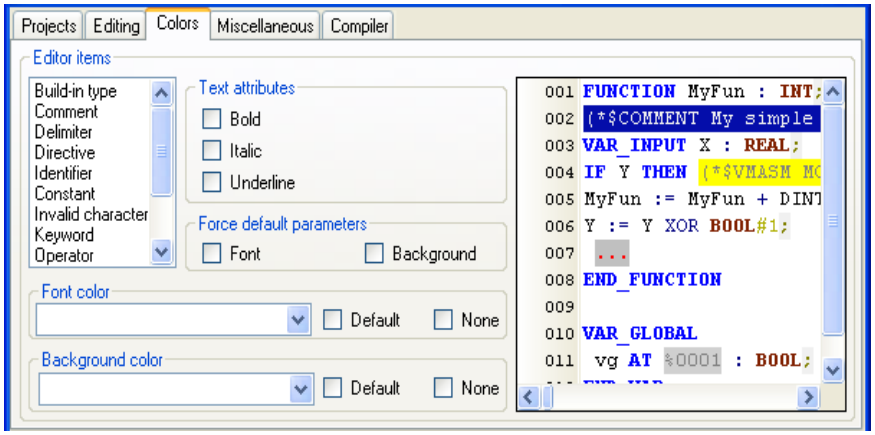
Editing

Single and *Double* colorable modes show keywords in different colors. *Single* (default) provides additional *autocomplete* help to finish names of variables, functions, etc. (*Supplements*). *Auto synchronization*… unifies names of the same elements in different parts of the project.



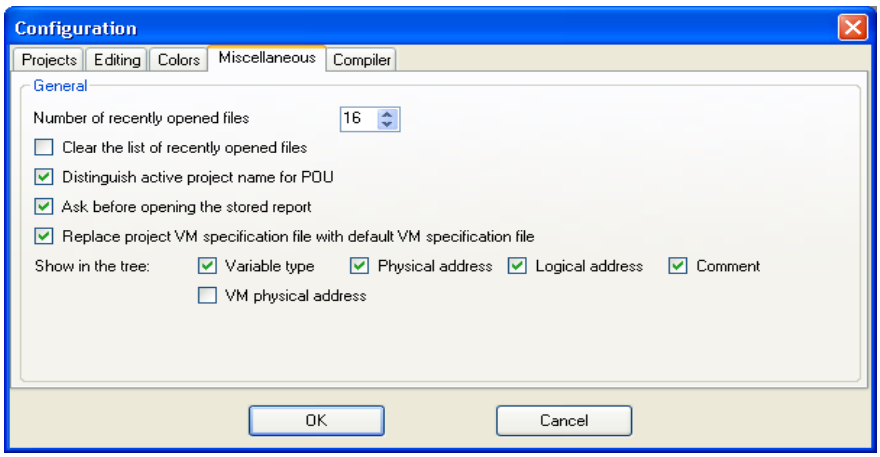
Colors

Scheme of editor colors, text attributes, etc., together with example of colored code, is shown below.



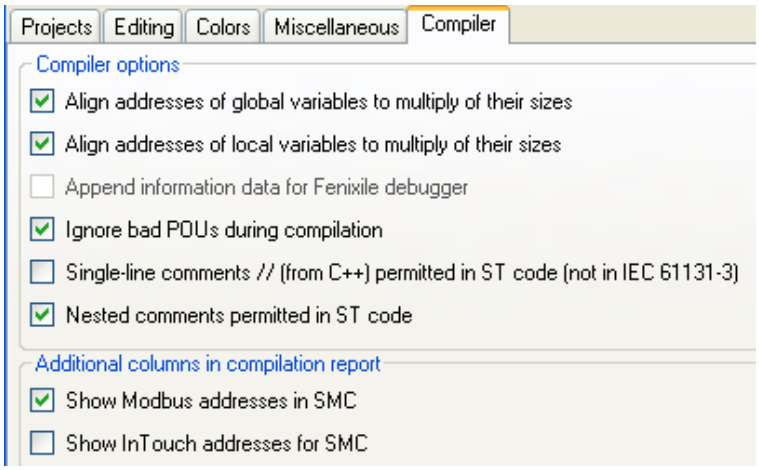
Miscellaneous

Size of *Recent files* list is determined. Bold characters distinguish active project for selected POU. *Ask*, or not, *before opening the stored report* in default browser. *Replace* Virtual Machine specification file by default (from *Projects* tab). For a global variable, the project tree may show type, three addresses and comment.



Compiler

Align addresses avoids overlapping of variables. C++ and nested comments may be accepted. Configuration of visualization package, e.g. InTouch, requires Modbus addresses (for SMC controller).

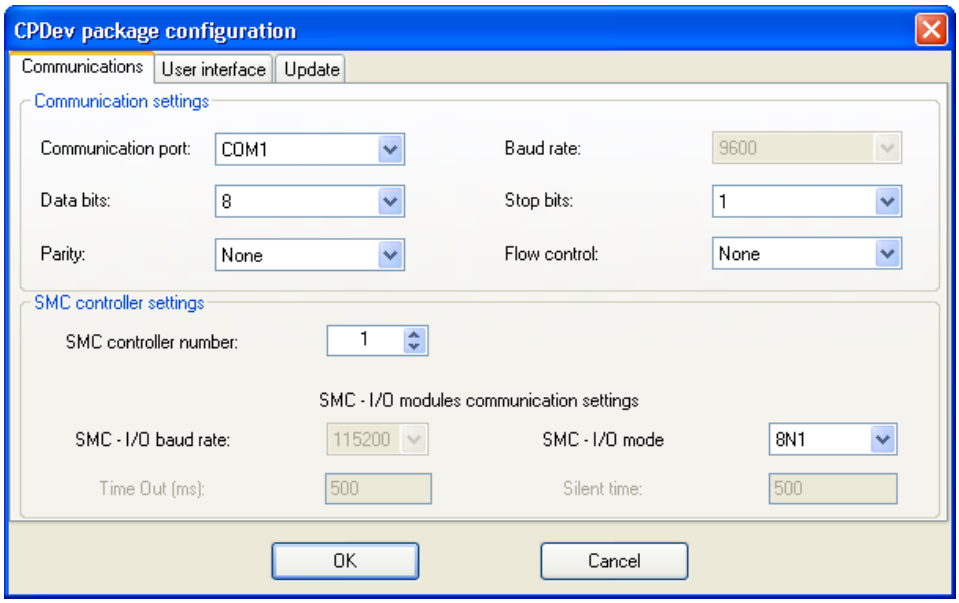


Global settings

They affect three CPDev programs, i.e. compiler, simulator and configurer. Selection of *Global settings* (in *Tools*) opens *CPDev package configuration* window with three tabs.

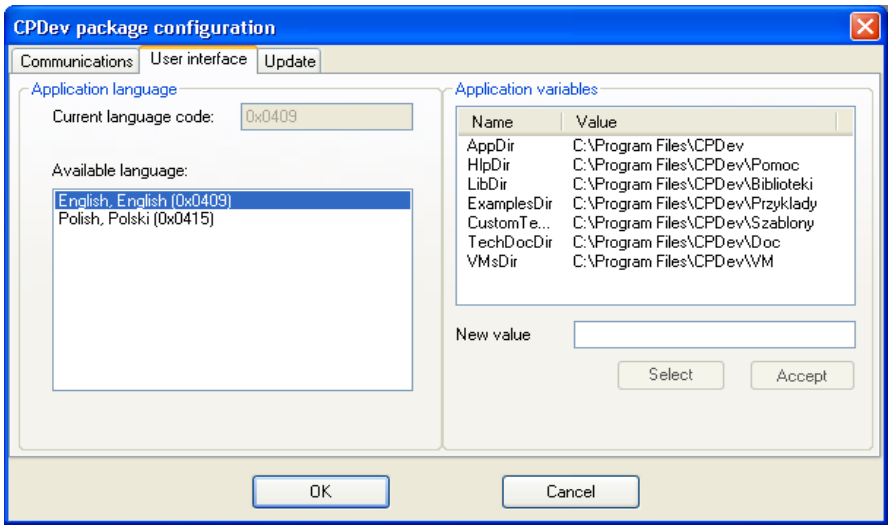
Communications

PC port for communication with the controller is configured according to *Communication settings*. If the controller is connected via USB, Windows *Device manager* determines port number. *SMC controller settings* define controller number for PC and parameters for communication with distributed I/O modules or other field devices. The 8N1 mode denotes 8 data bits, odd parity (N) and 1 stop bit.



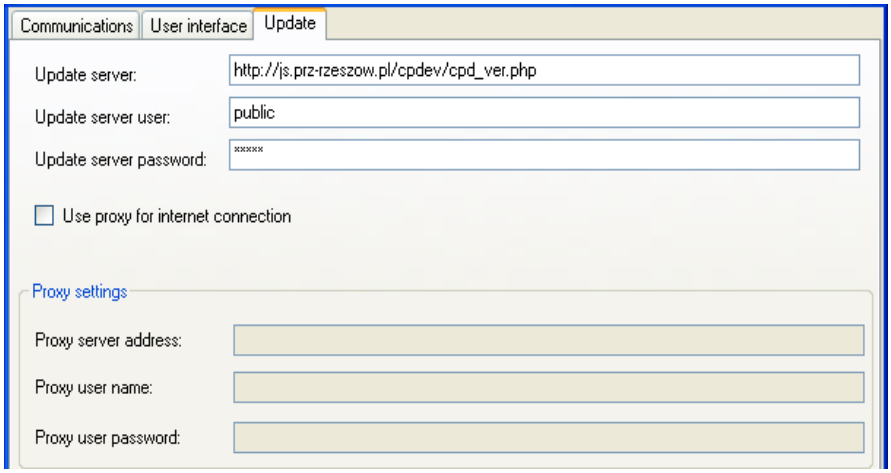
User interface

Interface language of CPDev package is chosen.



Update

The tab determines configuration to check whether new version of CPDev has appeared on the update server.



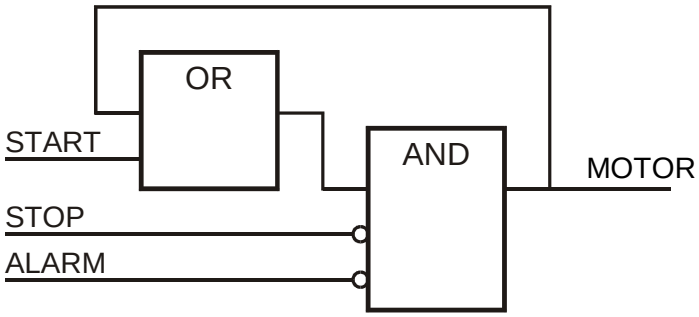
Remark. Passwords of the update and proxy users are not encoded, so should be erased after checking the update.

NEW PROJECT – START_STOP

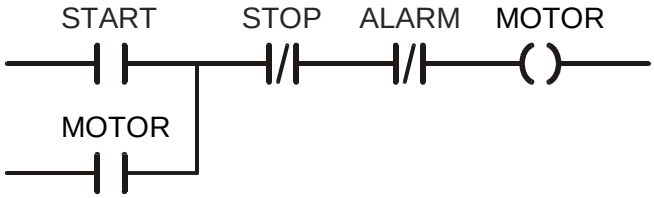
START_STOP system

The objective is to turn a motor on and off. Sample control diagrams are shown below.

Functions block diagram



Ladder diagram



START, STOP and ALARM inputs are acquired by the controller from binary input module. MOTOR output is sent from the controller to binary output module. The following addresses are assigned to variables.

START	0000
STOP	0001
ALARM	0002

MOTOR	0008

The adjacent three addresses indicate that START, STOP and ALARM will be read in one command or message. All signals correspond directly to hardware, so they will be declared as global variables.

Remark. The START_STOP system can also be implemented by means of RS flip–flop, with START connected to S input and STOP plus ALARM to R.

Create a project

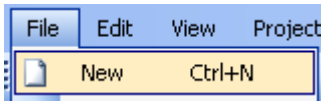
First open a new folder, e.g. START_STOP, for all files of the project. Steps executed by CPDev are then as follows:

1. Create a new file
2. Give name to the project
3. Declare global variables
4. Enter the program
5. Declare task
6. Compile the program
7. Save the source code in XML file
8. Close the project

Entering the program may precede declaration of variables. Closing the project saves all files in the project folder including binary code (*.xcp*) and data file (*.dcp*) for simulator and configurer.

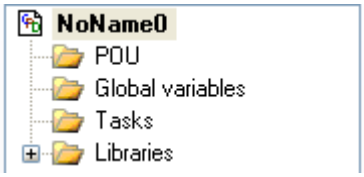
New file

- *File* > *New* (Ctrl+N)



or  in toolbar.

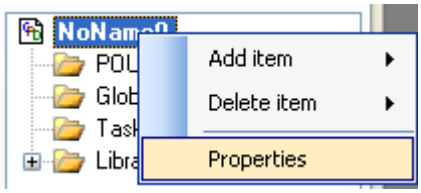
Empty *NoName* project appears in the project tree.



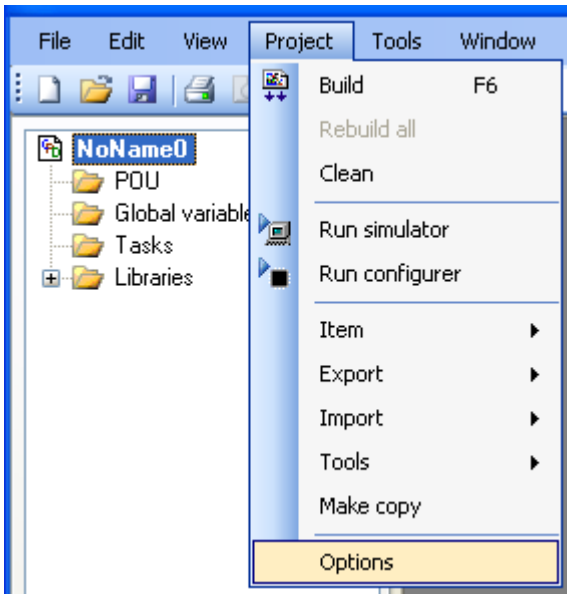
Project name

The project is given the name START_STOP entered in *Project properties* window.

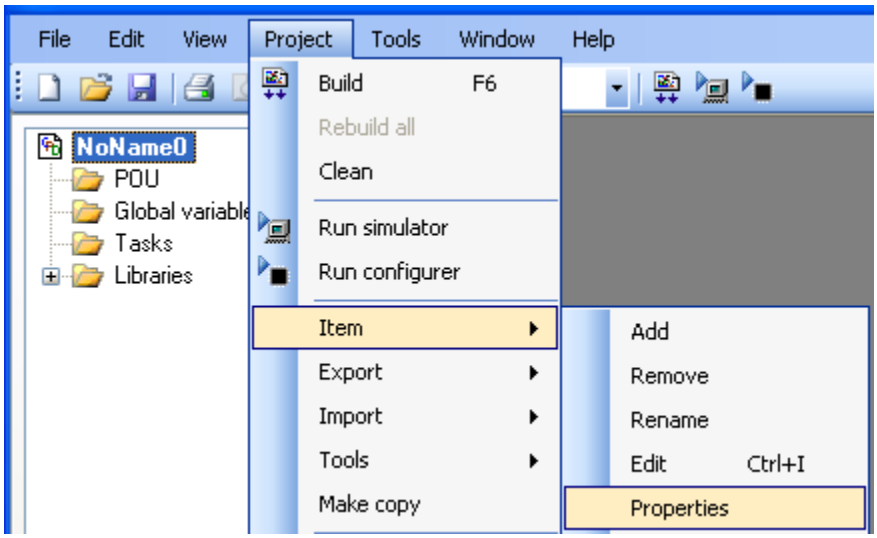
- Project tree > Select the project (*NoName*)
  Project properties can be opened in four ways:
  1) Context menu > *Properties*



  2) *Project* > *Options*

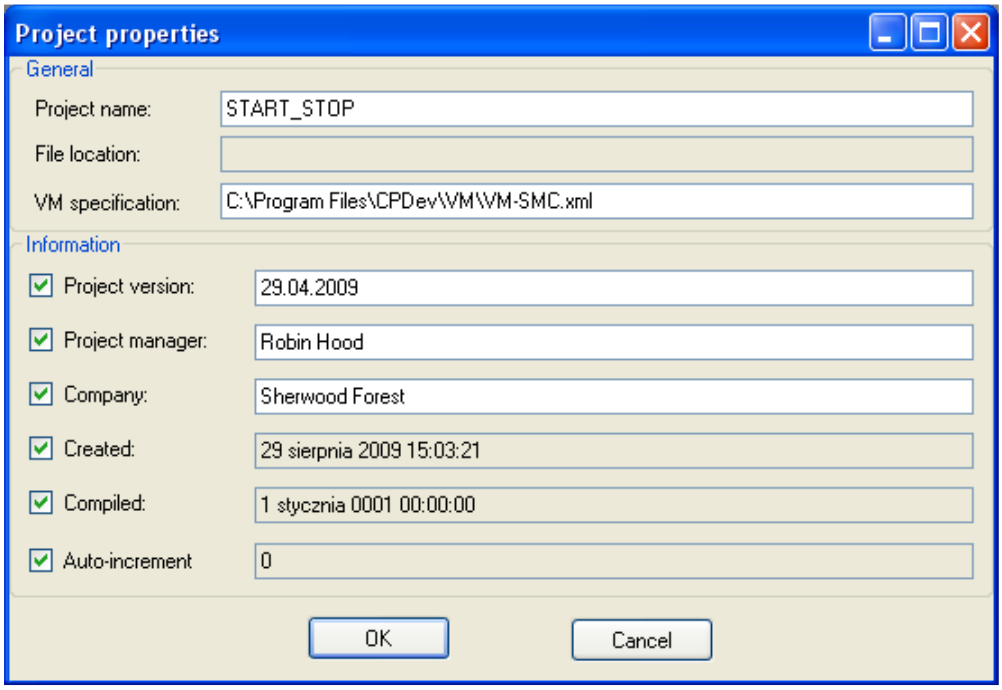


3) *Project > Item > Properties*



4) *Alt + Enter*

- Enter the name and eventually fill other information cells of *Project properties* (*created* and *compiled* are filled automatically). The name must be correct identifier in ST, so without spaces inside or digits at the beginning (see *ST language overview*).



After OK the new name appears in the project tree.

The contents of Version, Manager and Company cells will be downloaded to the controller together with the program. By reading it back you can always find out what program is executed.

Global variables

Global variables can be used in all programs of the project. Three ways of declaration are available:

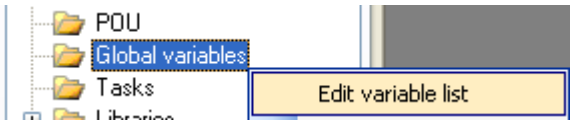
1) Global variable list
2) Individual declaration of each variable
3) VAR_GLOBAL declaration before the program.

The first way is most common. Individual declarations are described in the next section. VAR_GLOBAL before program, requires changes of a few options (see *For advanced users*).

Global variable list

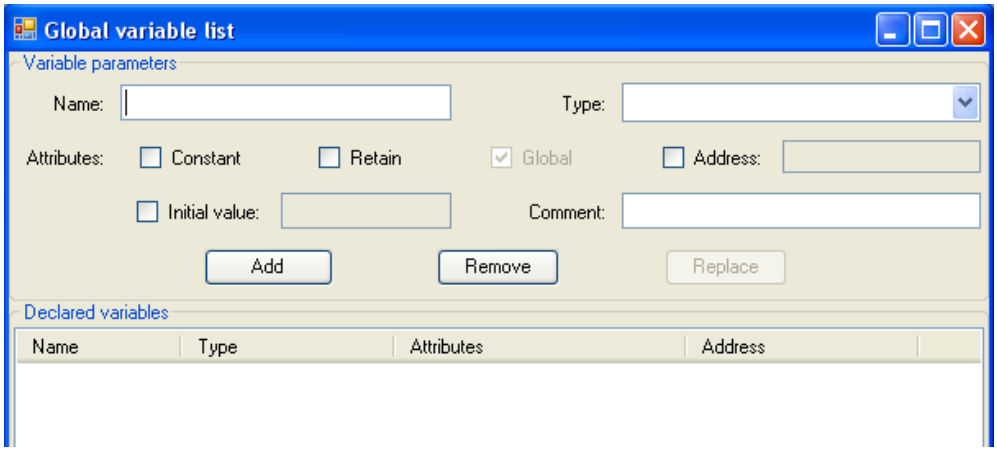
- Open the list in one of two ways:

  1) Select *Global variables* (project tree) > Context menu > *Edit variable list*

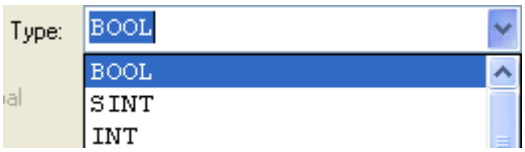


2) *Project > Tools > Global variables*

Empty list is displayed.



- *Group of variables, type*
  The group consists of variables of the same type with adjacent addresses, so START, STOP and ALARM here. Names are entered in Name cell, Type selected from drop–down list or typed in (type first characters, press the arrow ↓ and the editor will match the rest).



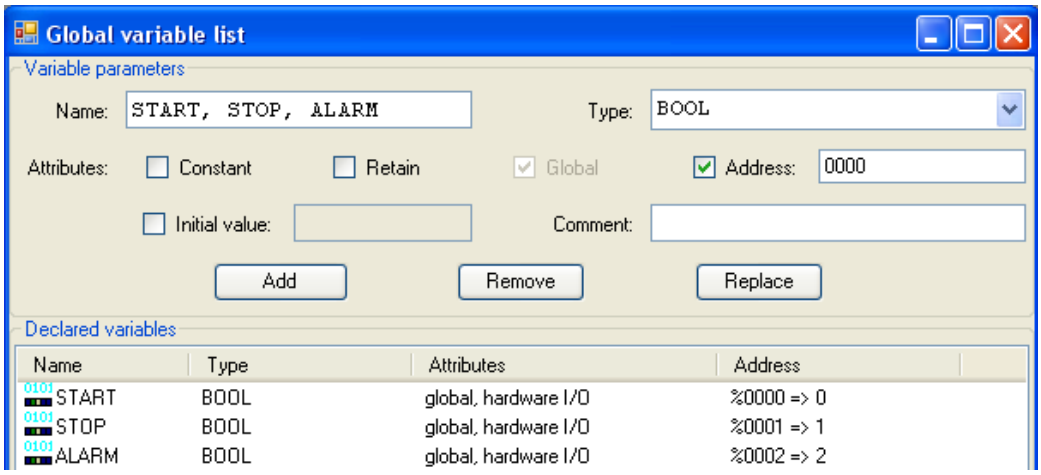
  Remark. STRING, USINT, UINT, UDINT and ULINT types are not implemented yet.

- *Address*
  Selection of *Address* option automatically fills the cell with first unoccupied address, so 0000 here. For types other than BOOL, the address begins with the sign % and size prefix (*ST language overview*). If *Address* is not selected, the variables are located automatically.

- *Constant, retain*
  Attribute CONSTANT declares a variable which does not change during program execution, and RETAIN a variable whose last value is kept in memory despite power failure.

- *Initial value*
  If the option is not selected, the variable is set initially to default value (usually zero). For RETAIN variable the initial value applies for cold start only (i.e. after downloading the program). In case of warm restart (power resumed), the last value kept in memory is used. Non RETAIN variables are set to initial values both during cold start and warm restarts.

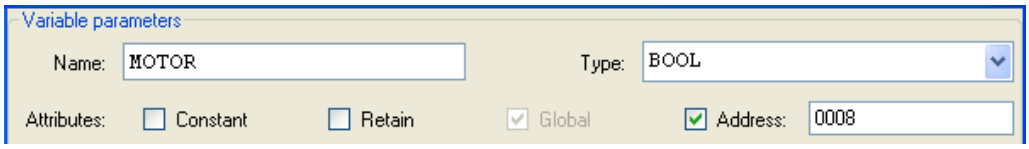
- *Comment*
  Text from the cell is displayed in the project tree and in autocomplete hints (Ctrl+space).
- *Add*
  Pressing the button fills the list with declared variables. If the Address option is not selected, text *auto* appears in the last column.



- OK closes the window. START, STOP and ALARM appear in *Global variables* section of the project tree.

The variables involve type, physical and logical addresses (or *auto*), and ev. comment.

- MOTOR variable
  It could not be declared in the previous group since its address is not adjacent (0008). Select *Address* and enter 0008 instead of initial 0003.



  After *Add* and OK, MOTOR shows up in the project tree.

Replace, Remove

Selecting a variable in the list recreates its name, type and attributes in the upper cells. To make corrections, enter new data and press *Replace*. *Remove* deletes selected variable. Selection of a few variables (Shift or Ctrl) recreates only those parameters which are the same. New entry and *Replace* makes change in all selected variables.

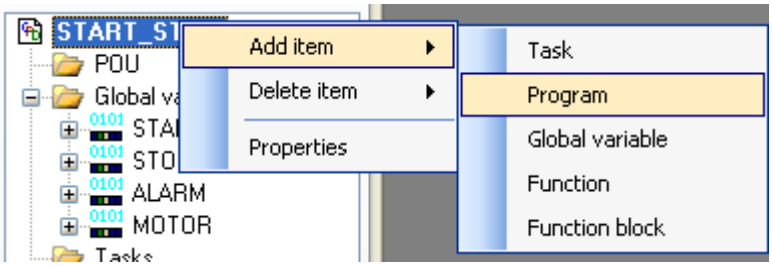
Remark. The CPDev package provides first free address for the group being declared, but does not check whether the whole group fits into the area before variables placed further down (if any). In case of collision the overlapping variables are shown in red.

Program

Name of the program is entered in *Program properties* window.

Program name and preview

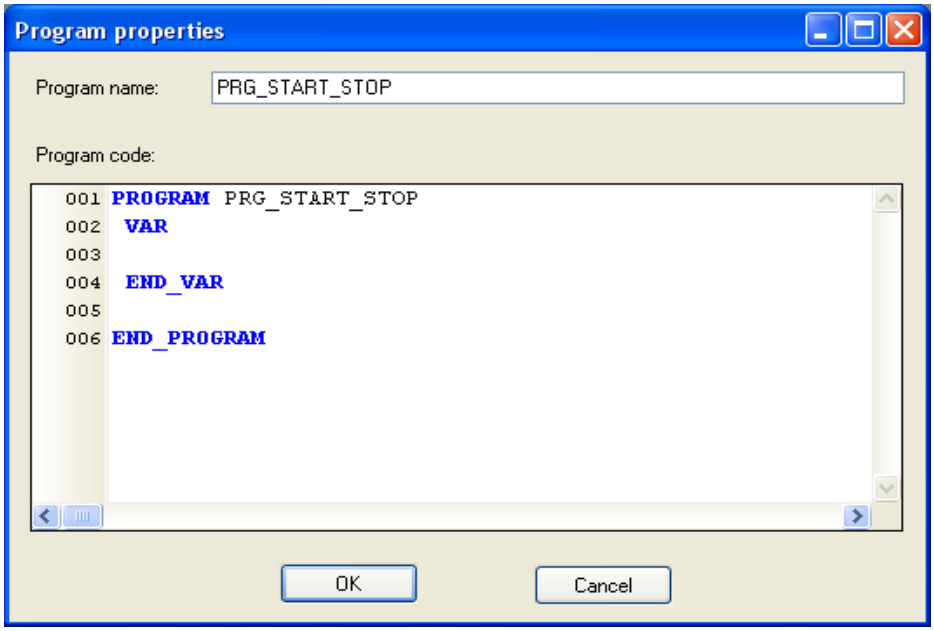
- Select the project > Context menu > *Add item* > *Program*



The window can also be opened by:

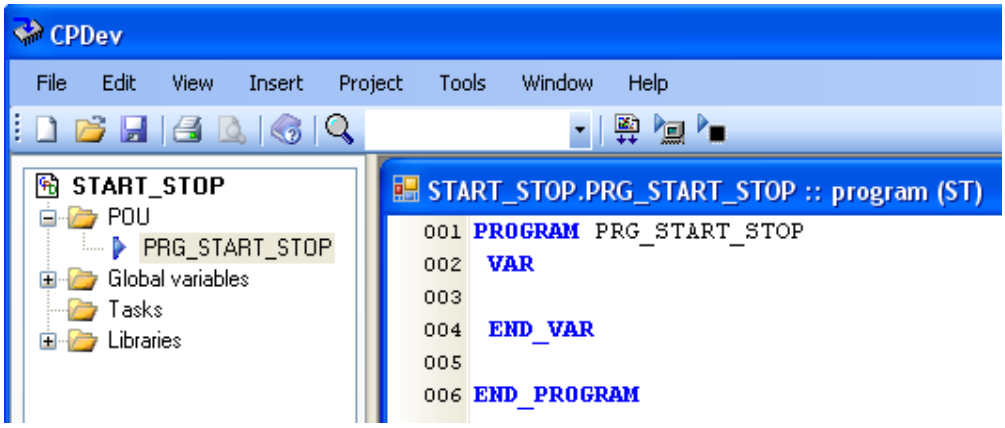
Select POU > Context menu > *Add* > *Program*

- Enter program name, here PRG_START_STOP (initial PRG is left to distinguish program from the project). Due to *Auto synchronization of project names* (*Environment options*) the name appears simultaneously in the line 001 of the code field.



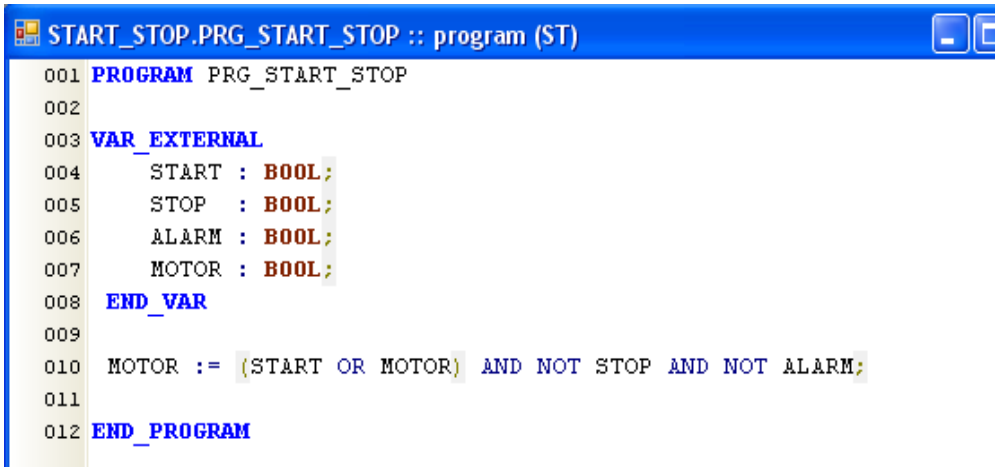
OK. The project tree involves PRG_START_STOP in POU section.

- Double click PRG_START_STOP.
  The program window in edit mode is displayed (*Automatically unlock window for editing*).



Enter the code

- Code of PRG_START_STOP is shown below. VAR_EXTERNAL declarations indicate that the global variables START, STOP, ALARM and MOTOR are used in the program. Body involves single assignment statement with expression corresponding to control diagrams at the beginning.



While entering the code, functionally different elements are shown in different colors and ev. bold. The editor is equipped with a number of useful shortcuts (*Supplements*).

Remark. The code can also be entered in *Program properties* window.

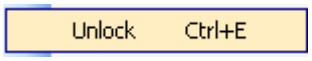
Preview vs. editing

Program and other elements of the project may be inspected in preview mode, protected against modifications. Preview is activated by:

- Select the program > *Project* > *Item* > *Lock* (Ctrl+D)

Return to edit mode is similar.

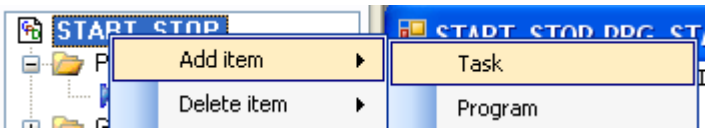
- *Project* > *Item* > *Unlock* (Ctrl+E)



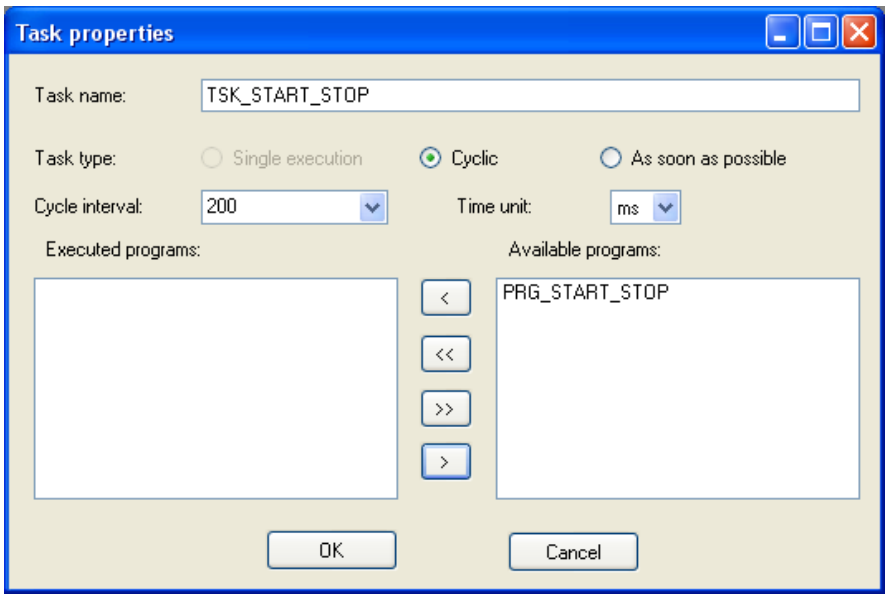
Task

Single task is available in the current version of CPDev. Name of the task and programs are declared in *Task properties* window.

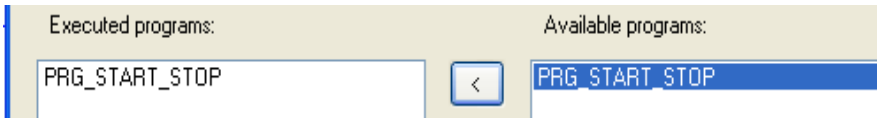
- Select the project > Context menu > *Add item* > *Task*



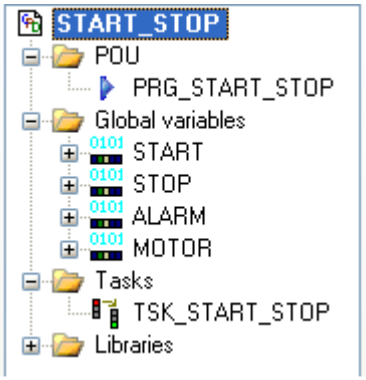
- Task name and type. Cycle time
  Fill appropriate cells, i.e. with TSK_START_STOP, *Cyclic* and 200 ms here. *As soon as possible* means that immediately after completing one execution, another begins (so–called PLC mode).



- Select PRG_START_STOP from *Available programs* and with upper buttons transfer it to *Executed programs*.



OK
TSK_START_STOP appears in *Tasks* section of the project tree.



Remarks. Programs stored in linked libraries (if any) appear in *Available programs*. A program repeated in *Executed programs* is executed more often.

Save project in XML file

New project must be saved in XML file before compilation. Recall that the START_STOP folder has been opened at the beginning for all files of the project. Current code is saved in *Start–Stop.xml* file in that folder.

- *File* > *Save* (Ctrl+S) or 

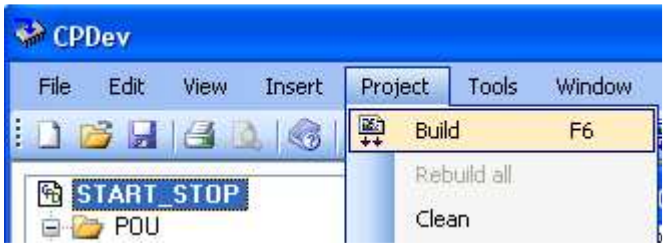
[Windows XP]

.xml extension is provided automatically.

Compilation

The program is compiled to universal executable code in binary format for virtual machine (*runtime*).

- Select the project (or any element of it) > *Project* > *Build* (F6)

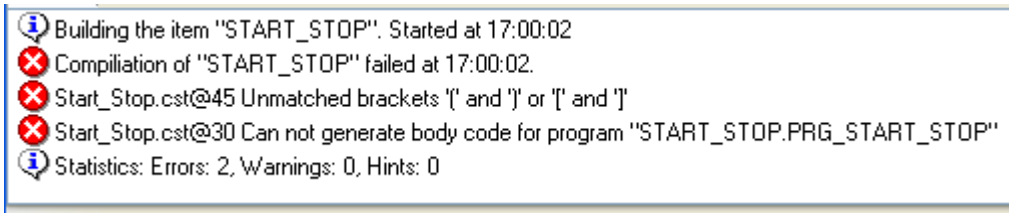


Message window shows compilation results.

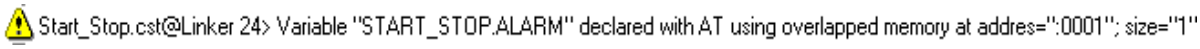
Global variables declared without addresses obtain physical addresses seen in the project tree, in parentheses. Logical addresses are still denoted by *auto*.

Error and warnings

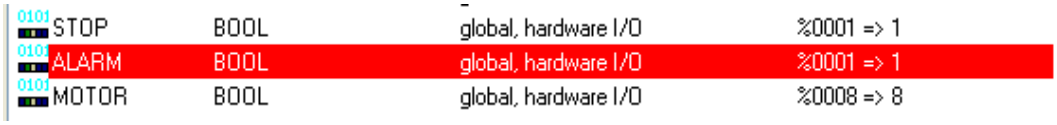
Error is indicated by red cross with corresponding description. Double click the description and program code is displayed with cursor in the line with the error (most probably). Errors caused by other reasons than violation of ST syntax are indicated at the beginning (line 0 or -1).



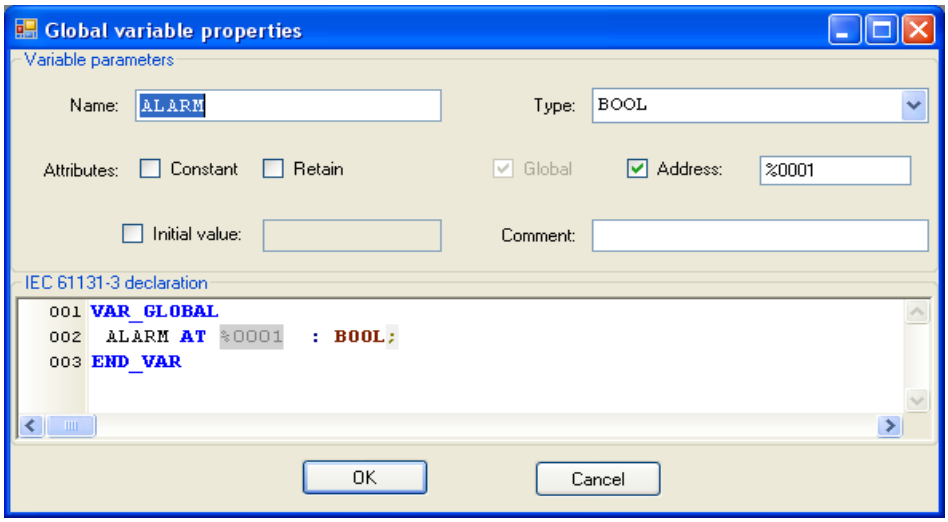
Yellow ”road” sign indicates warnings. If, for instance, ALARM were assigned the address 0001 (as STOP), the following warning would appear.



Double click the warning to open *Global variable properties* individual window for ALARM.



The address must be replaced and accepted.

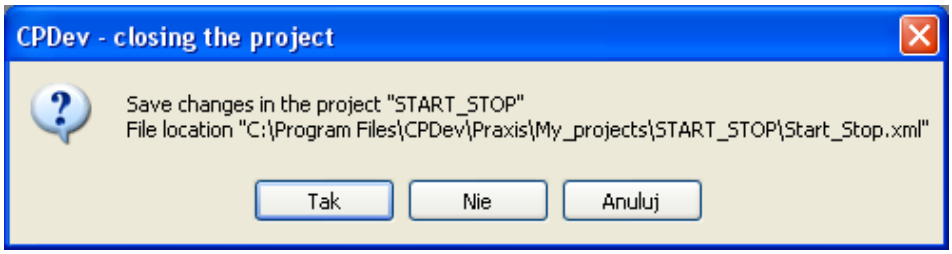


Group correction of global variable list is also possible (*Supplements*).

Save and close the project

The project is saved both in binary format (*.xcp*) and semi–compiled form (*.dcp*) for simulation and hardware configuration. Some intermediate files are also saved.

- Select the project > *File* > *Save* (Ctrl+S)
- *File* > *Close*
  CPDev – closing the project window is displayed with *Save changes* question and information on file location.



The question is asked even if no changes have been made (see *For advanced users* to remove it).

Remark. The START_STOP project will be extended in the next section, so it is closed here solely for demonstration.

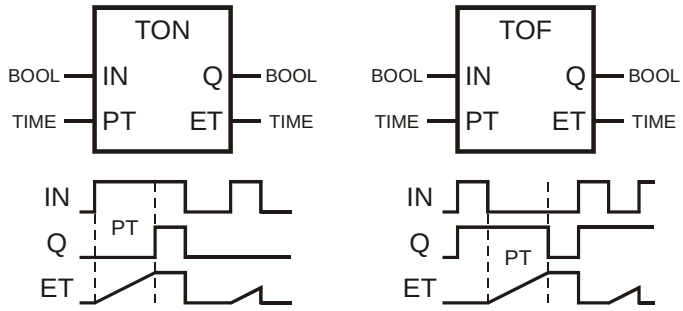
LIBRARY TIMERS

Delayed switchings

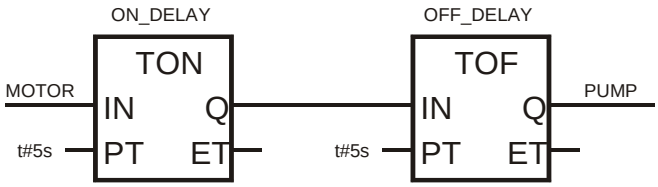
The START_STOP system will be extended by turning a pump on and off 5 seconds after the motor. The IEC 61131–3 standard defines a set of function blocks including three timers. Two of them will be used here:

TON – on–delay
TOF – off–delay.

Input/output symbols, types and time diagrams are shown below.



Let the instances of TON and TOF be declared as ON_DELAY and OFF_DELAY. The former program will be extended by statements implementing cascade connection of the following blocks.

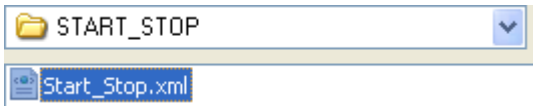


The PUMP signal will be sent to the same binary output module as MOTOR, so its logic address is 0009.

Open existing project

- *File* > *Open* (Ctrl+O) or 

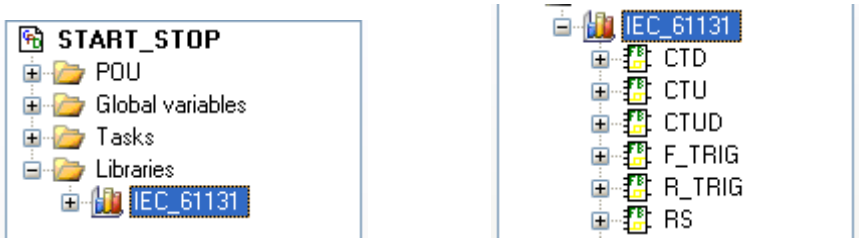
  Find START_STOP folder and open *Start_Stop.xml* file.



  The project tree appears in interface window.

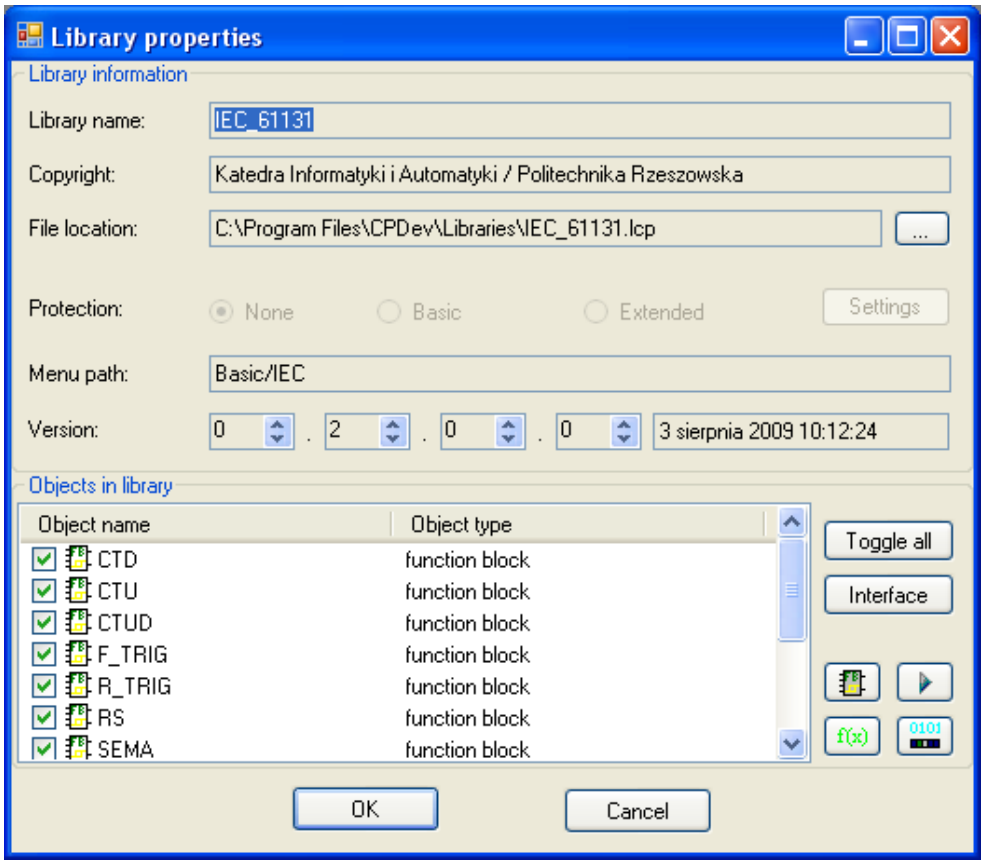
IEC_61131 standard library

The timers TON, TOF are stored in CPDev IEC_61131 library (linked to the project by *Environment options* > *Projects*).



Library content is displayed by unfolding the tree (above) or opening *Library properties* window.

- Select IEC_61131 library > Context menu > *Properties*



Remark. Time of the last compilation of the library is given in *Version*.

- *Buttons*

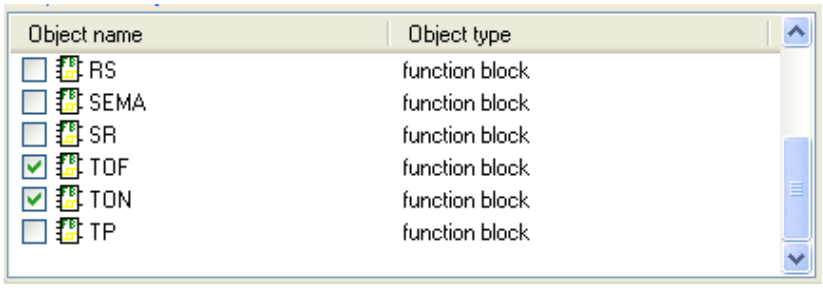
  – selects a library (transfers to *Libraries* folder)
  – reverses selections of all objects
  – shows declaration of selected object
  – reverses selections of function blocks
  – as above, for programs, functions and global variables, respectively.

The button 

is active only while exporting or importing the library (*Project* > *Export*/*Import* > *Library*).

- *Timers TON, TOF*

  Remove selections of other blocks than TON, TOF.



  Compiler links only those objects which are selected.

Extension of START_STOP project

The PRG_START_STOP program will be extended and variable PUMP declared.

Program

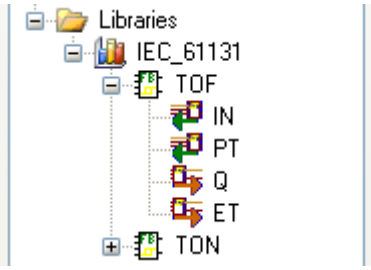
- Double click the program PRG_START_STOP in the project tree.

  Supplement the code with:
  – declarations of the instances ON_DELAY, OFF_DELAY
  – declaration of the use of global variable PUMP
  – statements corresponding to the cascade connection of the blocks and assignment to PUMP.

START_STOP.PRG_START_STOP :: program (ST)

```
001 PROGRAM PRG_START_STOP
002
003 VAR
004     ON_DELAY : TON;                    (* TON block instance *)
005     OFF_DELAY: TOF;                    (* TOF block instance *)
006 END_VAR
007
008 VAR_EXTERNAL
009     START : BOOL (*$READ*);
010     STOP  : BOOL (*$READ*);
011     ALARM : BOOL (*$READ*);
012     MOTOR : BOOL;
013     PUMP  : BOOL (*$WRITE*);
014 END_VAR
015
016 MOTOR := (START OR MOTOR) AND NOT STOP AND NOT ALARM;   (* MOTOR output *)
017
018 ON_DELAY(IN:=MOTOR, PT:=t#5s);          (* TON block - delayed turn on   *)
019 OFF_DELAY(IN:= ON_DELAY.Q, PT:=t#5s); (* TOF block - delayed turn off  *)
020 PUMP := OFF_DELAY.Q;                                  (* PUMP output   *)
021
022 END_PROGRAM
```

Optional directives (*$READ*), (*$WRITE*) assure ”read–only” and ”write–only” properties of declared variables. Input/output structure of function block can be recalled as tip in the project tree, or in the main window by selecting the block and clicking Enter.




```
001 FUNCTION_BLOCK TON
002 VAR_INPUT
003 IN : BOOL;
004 PT : TIME;
005 END_VAR
006 VAR_OUTPUT
007 Q : BOOL;
008 ET : TIME;
009 END_VAR
010  ...
011 END_FUNCTION_BLOCK
```

Remark. The two lines 19, 20 in the program code can be replaced by single one by using internal assignment *Q=>PUMP*.

```
018 ON_DELAY(IN:=MOTOR, PT:=t#5s);
019 OFF_DELAY(IN:= ON_DELAY.Q, PT:=t#5s, Q=>PUMP);
020
021 END_PROGRAM
```

Autocomplete

Name of type, function, variable, etc. may be automatically completed after writing at least one character, but only if the project at current stage has been compiled to acquire the names (*Build*). Pressing Ctrl + space generates list of names with the same beginning.

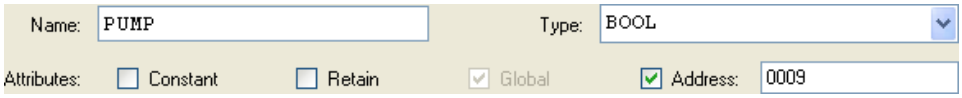
```
019 OFF_DELAY(IN:= ON_DEL
020
```

Local variable **ON_DELAY** : TON

New global variable

- Select *Global variables* > Context menu > *Edit variable list*

  Fill in upper cells and press Add.



Compilation

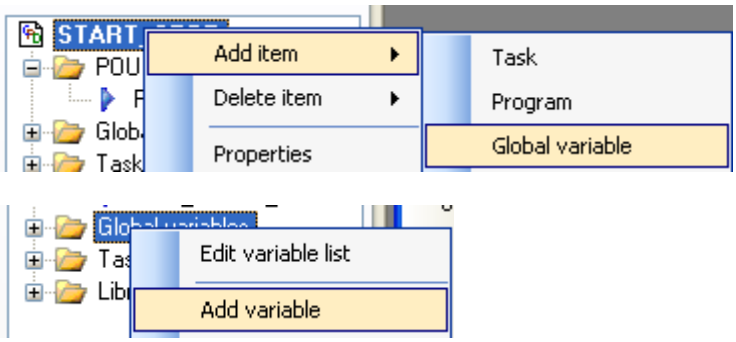
- Select START_STOP project
- *Project > Build*

Individual declaration of global variable

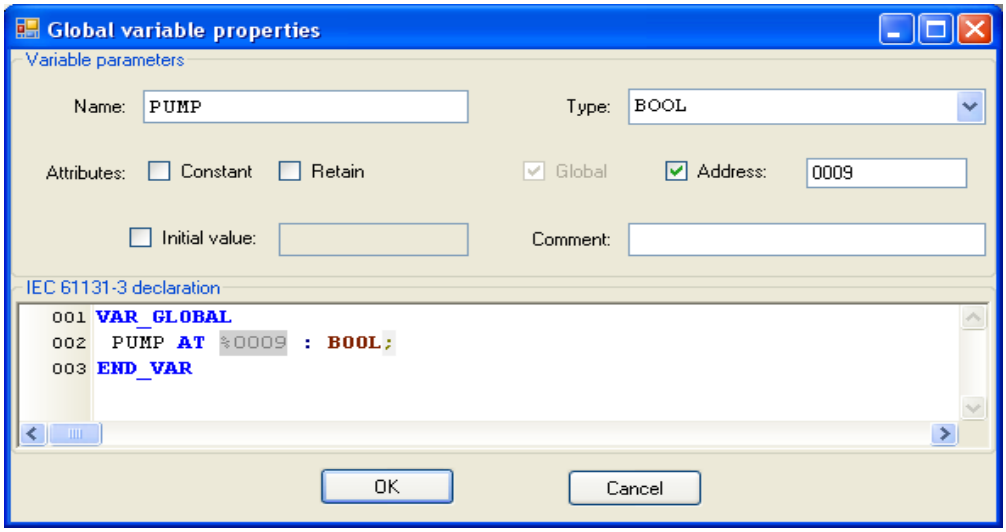
The variable PUMP can be also declared individually, what may be more convenient sometimes.

- Two ways are available:

  1) Select START_STOP project > Context menu > *Add item* > *Global variable*
  2) Select *Global variables* > *Add variable*

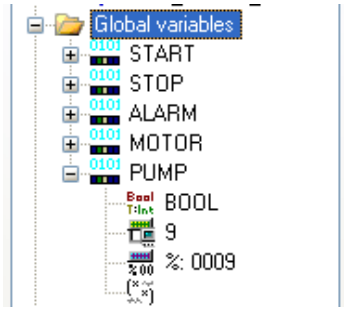


- Upper part of *Global variable properties* window should be filled in as before, lower part is updated automatically.



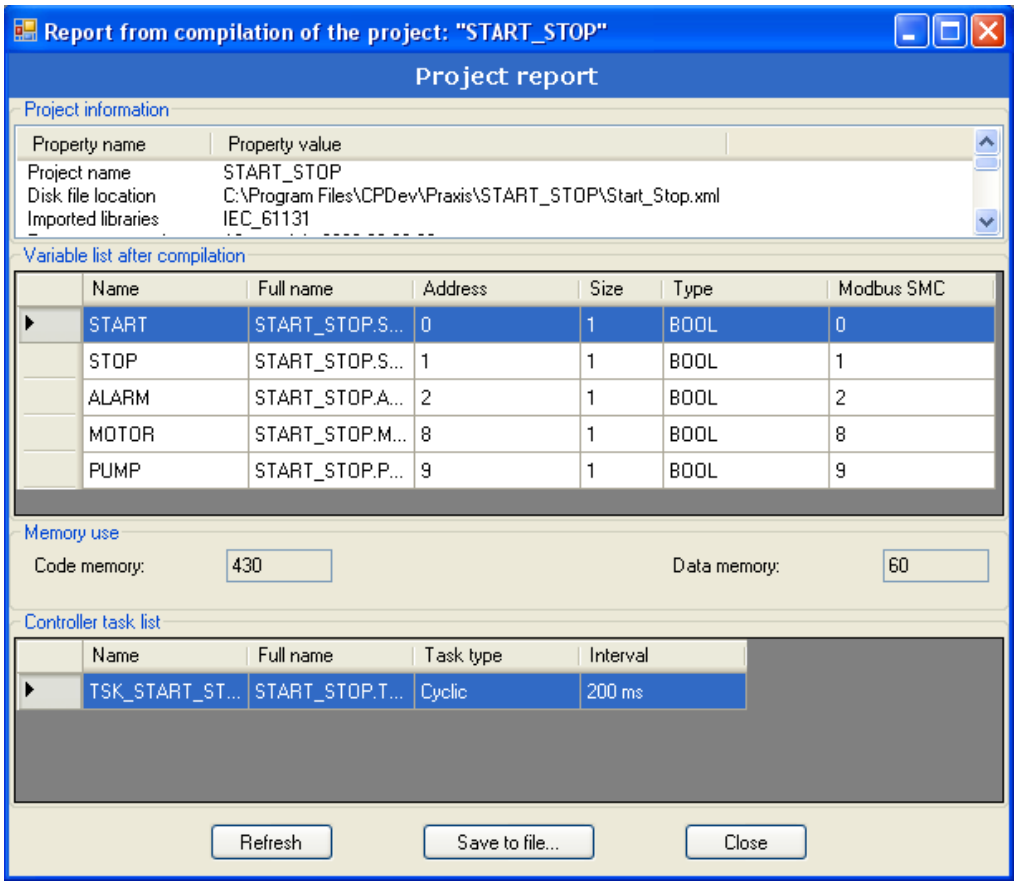
Recall that this window is also used to correct overlapping addresses.

- After OK the project tree is supplemented with PUMP.



Project report

- *Project > Tools > Report*



Full name column involves variable names preceded by project name (also in case of tasks).

Sorting

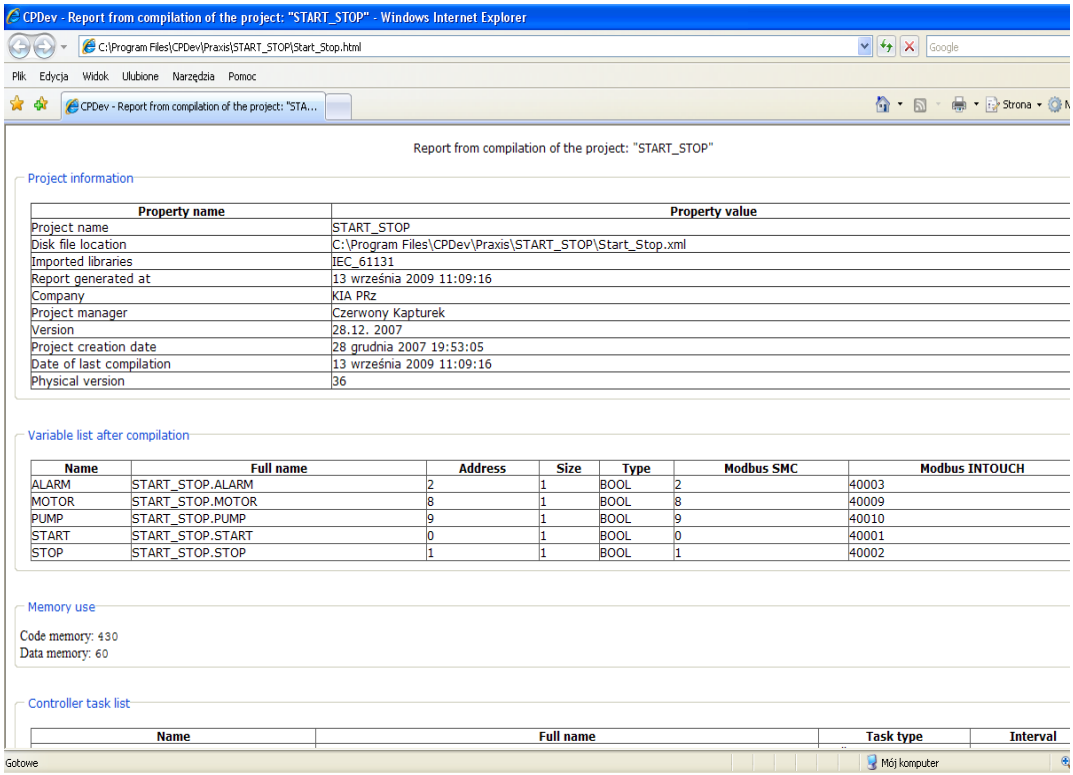
Initial order of variables in the report corresponds to declarations. This may be changed by clicking header of a column what shows the sign of increasing or decreasing sorting. Depending on the column, the sorting may be either alphabetic or numeric. The first one is shown below.

Variable list after compilation

Name	Full name	Address	Size	Type	Modbus SMC	Modbus INTOUCH
ALARM	START_STOP.A...	2	1	BOOL	2	40003
MOTOR	START_STOP.M...	8	1	BOOL	8	40009
PUMP	START_STOP.P...	9	1	BOOL	9	40010
START	START_STOP.S...	0	1	BOOL	0	40001
STOP	START_STOP.S...	1	1	BOOL	1	40002

HTML report file

Click *Save to file* in the previous window to save the report in HTML format.



Project save

- *File* > *Save* (Ctrl+S)

  Remark. The window indicating the path is not called up now since location of the file has been determined already (previous *Save*).

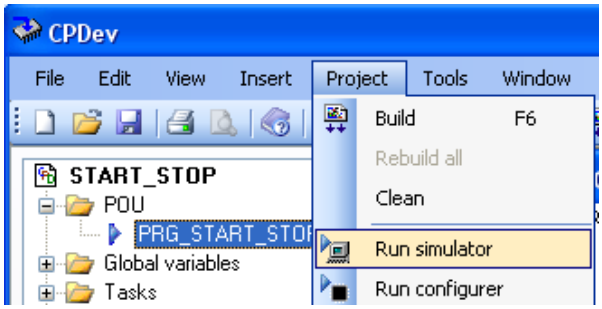
PROJECT SIMULATION

The purpose is to check operation of the project before final implementation. Both *off–line* and *on–line* tests can be carried out.

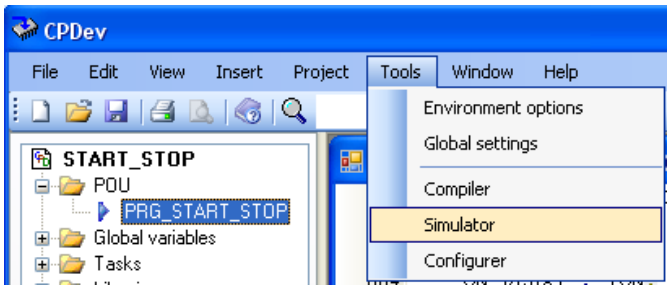
Run CPSim simulator

Three ways are available:

1) CPDev menu: *Project* > *Run simulator*



2) CPDev menu: *Tools* > *Simulator*



3) Start menu: *CPDev* > *CPSim*

The first way is used directly after compilation (*Project* > *Build*), what creates *.dcp* file read automatically by CPSim. The next two ways require opening the *.dcp* file from CPSim window.

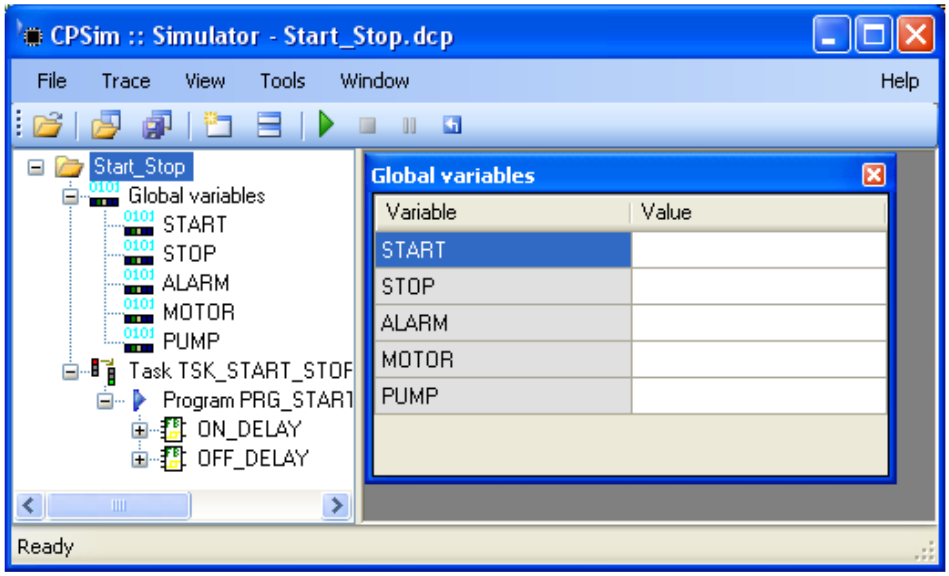
Open file for simulation

- *File > Open DCP file* or  (CPSim menu or toolbar, see below)

  Remark. If the project has been simulated already and session data saved, the question *Do you want to open saved session as well ?* is displayed.

Simulator window

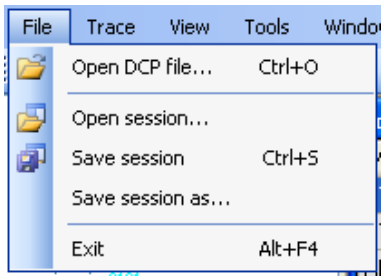
The window consists of two parts:
– variable tree
– view area



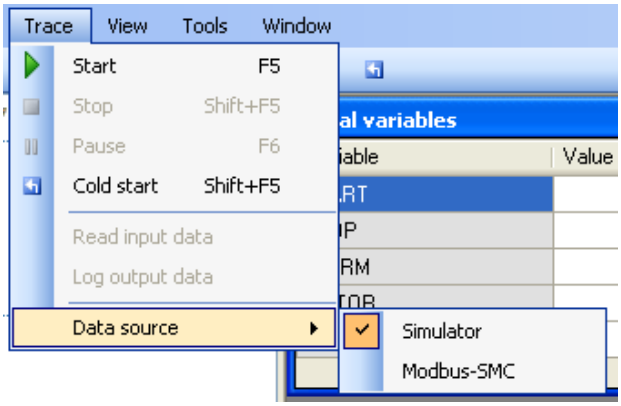
The variable tree differs a little from the project tree before. The view area presents initially the list of global variables or collection of individual windows for such variables (also called variable views). Panels for groups of variables or additional lists can also be placed in the view area. Scroll bars provide access to components outside (if any).

CPSim menu

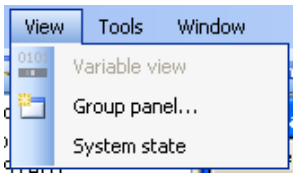
File



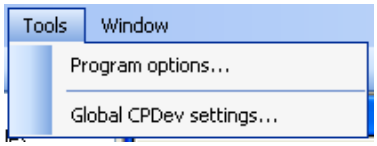
Trace



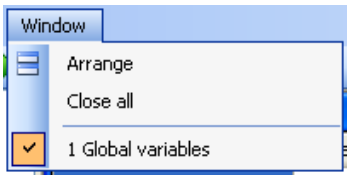
View



Tools



Window



Simulation session data can be saved in a file to repeat it later. *Trace* controls CPSim operation, so starts or stops it reads (*Supplements*) or logs variables, and selects data source, i.e. either Simulator (*off–line*) or Modbus–SMC (*on–line*). *Window* > *Arrange* places individual windows side–by–side.

Toolbar

Open DCP file | Open session | Save current session | Group panel

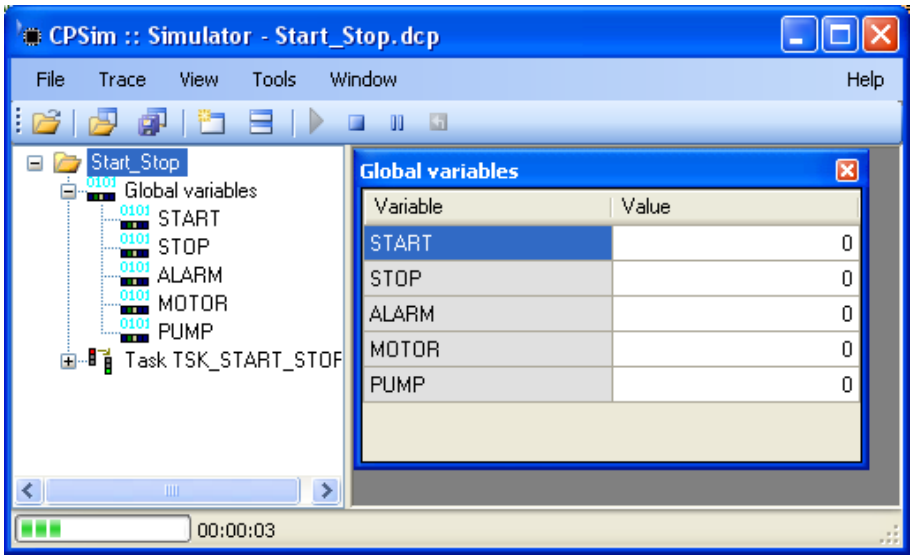
Arrange windows | Start trace | Stop trace | Pause or resume trace

Start program & reset RETAIN variables

Start, stop and pause

- *Trace* > *Start* or

  Simulation begins from initial values of variables (as first start after downloading the program into the controller). View area shows the results.



Bottom bar indicates simulation progress.

- *Trace > Stop* or 

  This corresponds to power brake in real controller, so last values of RETAIN variables are saved.

- Another *Trace > Start* or 

  Warm restart after power brake is simulated, so RETAIN variables are set to last values and non–RETAINs to initial.

-  Pause or resume trace

  Simulation stops and resumes without any change of variable values.

- *Trace > Cold start* or 

  This represents *cold start*, so simulation begins from initial values of all variables (as first start after downloading).

Variable list

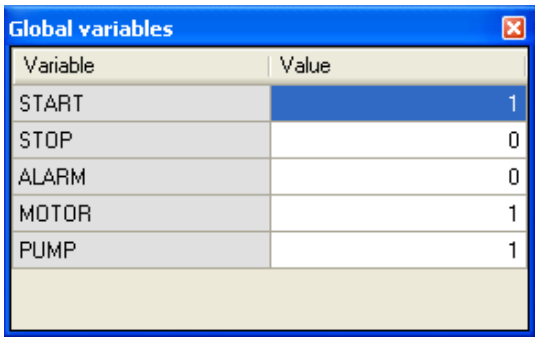
- *Enter value or variable*

  – Select corresponding cell 

  – Click for editing 

  – Enter new value, press Enter 

  Values after 5 seconds since 1 has been entered for START are shown below. MOTOR and PUMP are turned on.

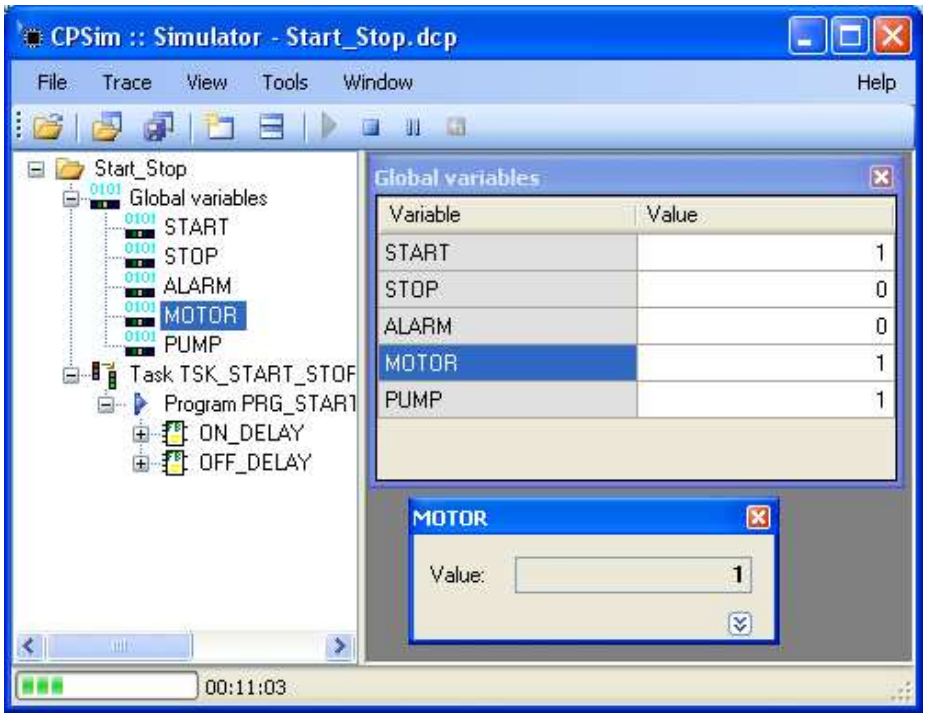


- *Add variable*
  Select variable in the tree, drag it to the list and drop (keeping pressed left key of the mouse).
- *Remove variable*
  Select line > Context menu > *Remove*

Variable views (individual windows)

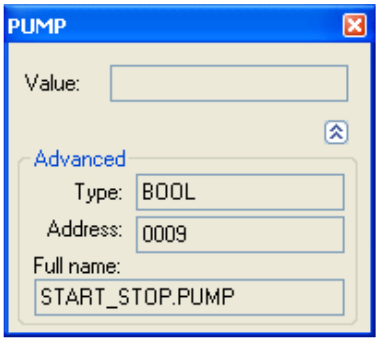
- *Add view*
  - Select variable in the tree.
  - View of the variable can be opened in three ways:
    1) Drag–and–drop the variable in view area.
    2) Menu: *View* > *Variable view.*
    3) Context menu: *Variable view.*

  Variable view for MOTOR is shown below.



New values are entered in the same way as in the list.

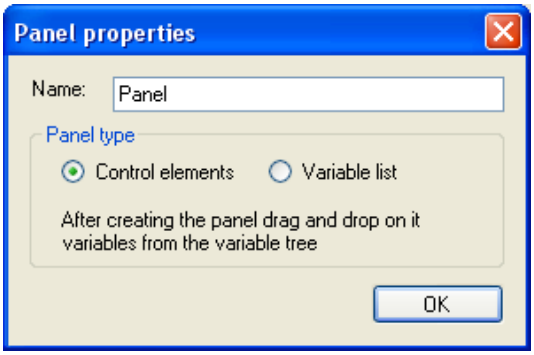
- *Close view*
  Click 
- *Additional information on variable*
  Click  to show lower part of the variable view, with type, address and full name.



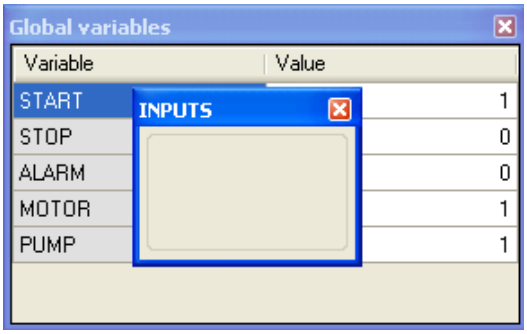
Group panels

Two kinds of group panels are available:
– control panels
– variable lists.
Variable lists look the same as the list of global variables before. Panels with control elements are created as follows:

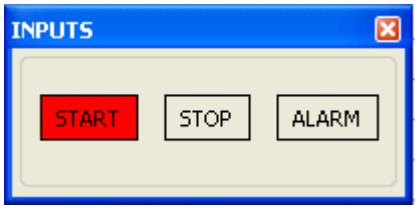
- *View* > *Group panel* or 
  Panel properties window is displayed.

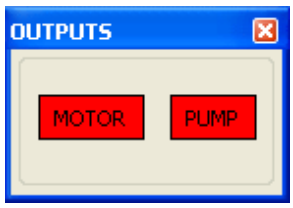


- Enter name, select *Control elements*, press OK.
  Empty panel with the name (INPUTS) appears in the view area.



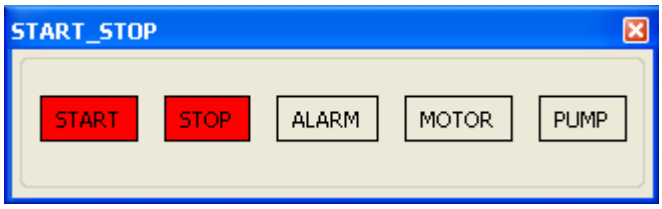
- Fill in the panel with appropriate variables by drag–and–drop from the tree. Panel grows automatically. Boolean variables are represented by rectangles, variables of other types by text cells.





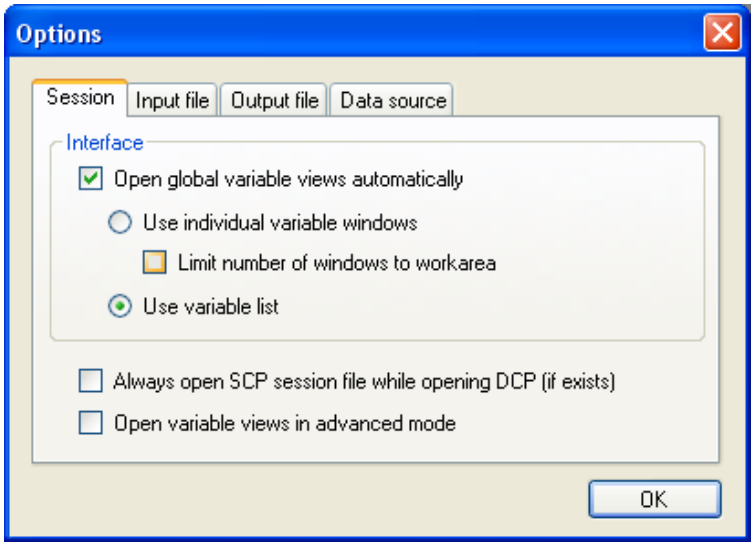
Panel in trace mode

Colors of rectangles depend on values. Click the rectangle to reverse value.



Program options

- Selecting *Tools* > *Program options* opens the window with four tabs.

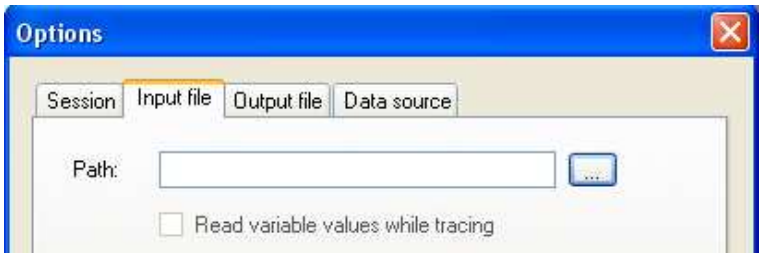


- *Session*

  The option *Open global variable views automatically* opens either the list (default) or collection of individual windows. The number of such windows may be limited for large projects. The question *Do you want to open saved session as well ?* asked at the beginning is dropped if the option *Always open SCP session file …* is selected. Open variable views in advanced mode opens lower parts of individual windows.

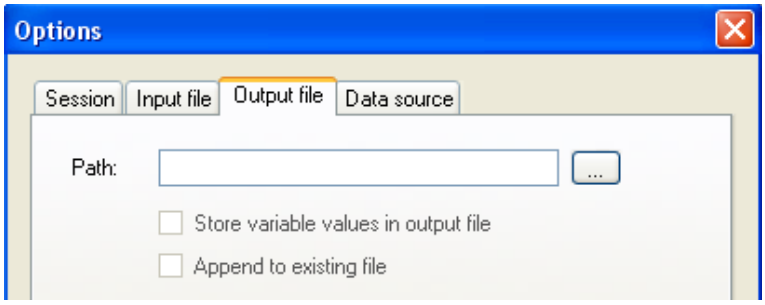
- *Input file*

  The tab defines *.inp* file for simulation controlled automatically *(Supplements). Path* to the file can be chosen by pressing ... or entered directly.



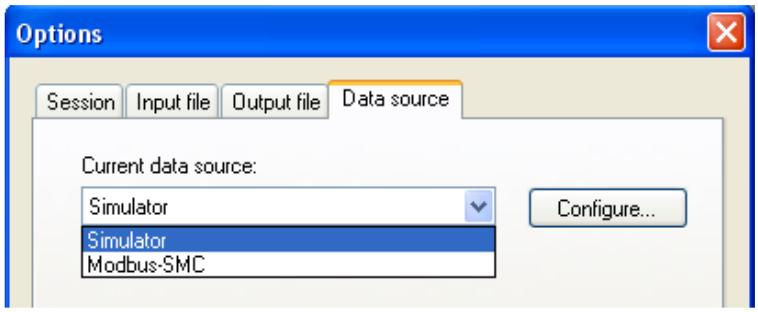
- *Output file*

  Simulation results may be recorded in *.out* file (default name as project file name). If the file exists already, its content may be overwritten or appended.



- *Data source*

  The tab is equivalent to *Tracking* > *Data source* in the menu, so it selects either *off–line* simulation or *on–line* commissioning (for SMC controller). Communication parameters can be checked by pressing *Configure*.

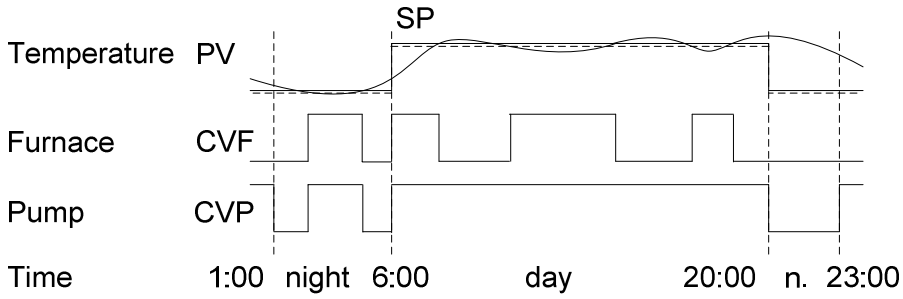


RTC CLOCK

Problem description

Temperature in an apartment must be kept at given level SP (Set Point), higher during the day, e.g. 22°C, lower at night, 18°C. Actual temperature PV (Process Variable) is measured by analog input. If SP>PV, heating furnace is turned on by Control Variable CVF (CV Furnace) from binary output, and if SP>PV the furnace is turned off. However, to avoid frequent switchings, the furnace can be turned on again only if the temperature PV drops below SP by at least 0.5°C (hysteresis). Circulation pump, controlled by the output CVP (CV Pump), is turned on all time during the day, and at night when the furnace is on and between the hours 23.00 and 1.00, no matter whether the furnace is on or off (the day is understood as the period between 6.00 and 20.00).

Sample diagrams

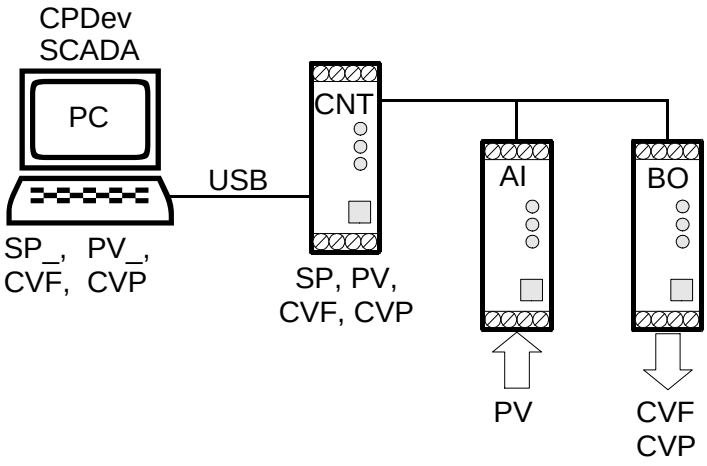


Control system

The controller CNT measures the temperature PV and controls the furnace and pump by the outputs CVF, CVP. It also communicates with PC computer, which:

- sets the set point SP,
- monitors the variables PV, CVF, CVP.

Temperatures at the controller side are denoted by SP, PV and at PC side by SP_, PV_ (different formats).



Analog input

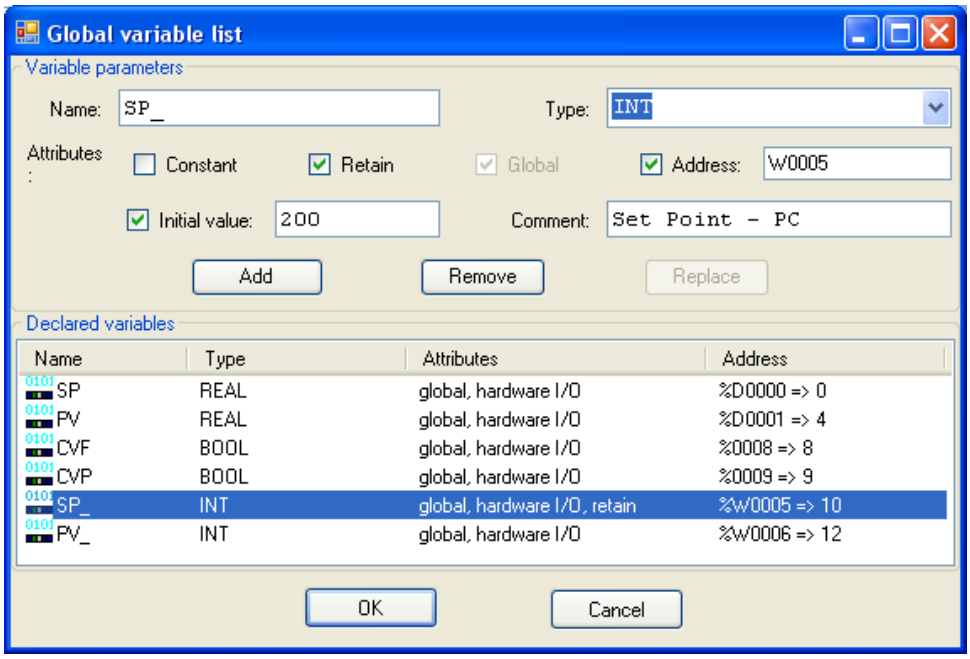
Temperature in the range 0...100°C is measured by a transmitter with voltage output 0...10V. A/D converter converts the voltage to REAL number PV in 0.0…10.0.

Communications

Assume that PC and the controller can exchange data of the types BOOL and INT only. So the temperatures SP_, PV_ at PC side are INT variables. Accuracy 0.1°C is required, so the range of SP_, PV_ corresponding to 0...100°C, is 0...1000 (SP=SP_/100, PV_=PV・100). For instance, the set point 20°C is represented by SP_=200 in PC and by SP=2.0 in the controller.

RTC project

Global variables

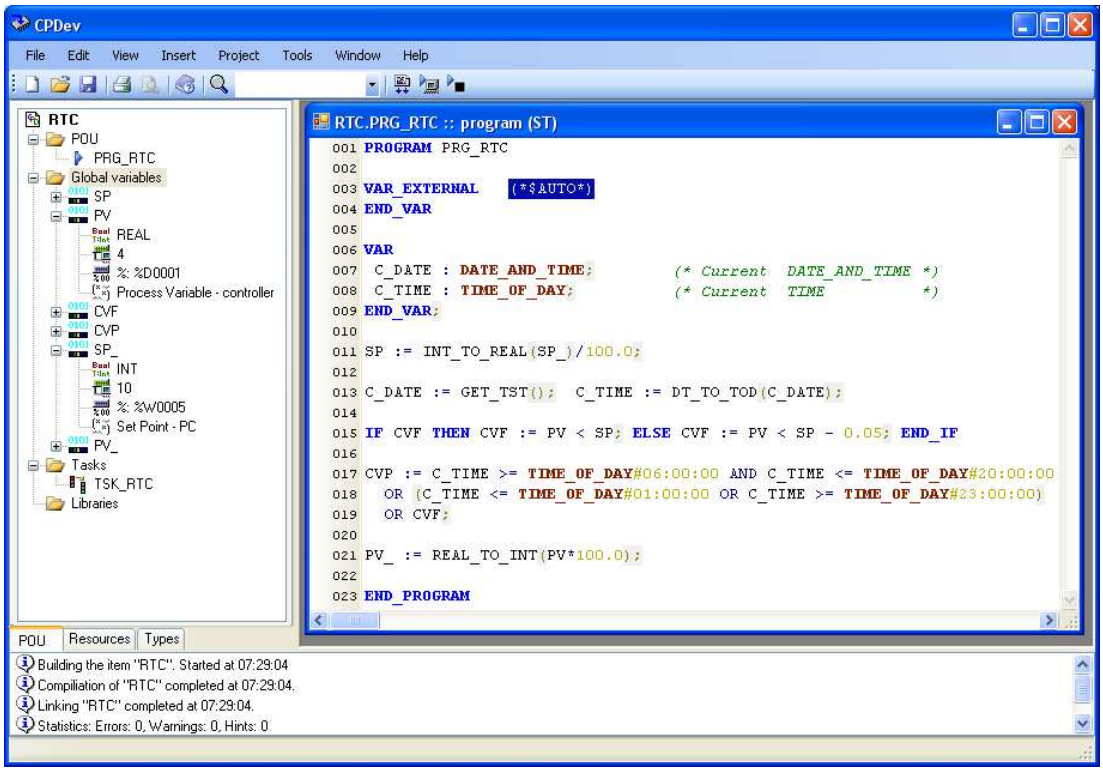


Note that corresponding pairs of variables can be declared as groups.

Set point temperature SP_ received from PC is declared as RETAIN, with initial value 200. So SP_ will be kept in memory despite power failure (warm restart) or communication brake. From SP_=200 (20ºC) the controller will begin operation after downloading the program (cold start).

Program

PRG_RTC program of RTC project is shown below. Comments seen in the project tree are entered during declaration of variables. The task TSK_RTC is executed every 200 ms.



The directive (*$AUTO*) after VAR_EXTERNAL automatically includes *Global variable list* into compiled program. Two local variables, C_DATE and C_TIME, are declared.

Statements in the lines

11: conversion of INT value received from PC into REAL, followed by adjustment of the range.

13: setting current date–and–time C_DATE to value returned by system function GET_TST() which reads the controller's RTC clock when the task begins (*Get Task Time*). Separation of current time C_TIME from C_DATE by DT_TO_TOD() conversion (*Day_and_Time To Time_of_Day*).

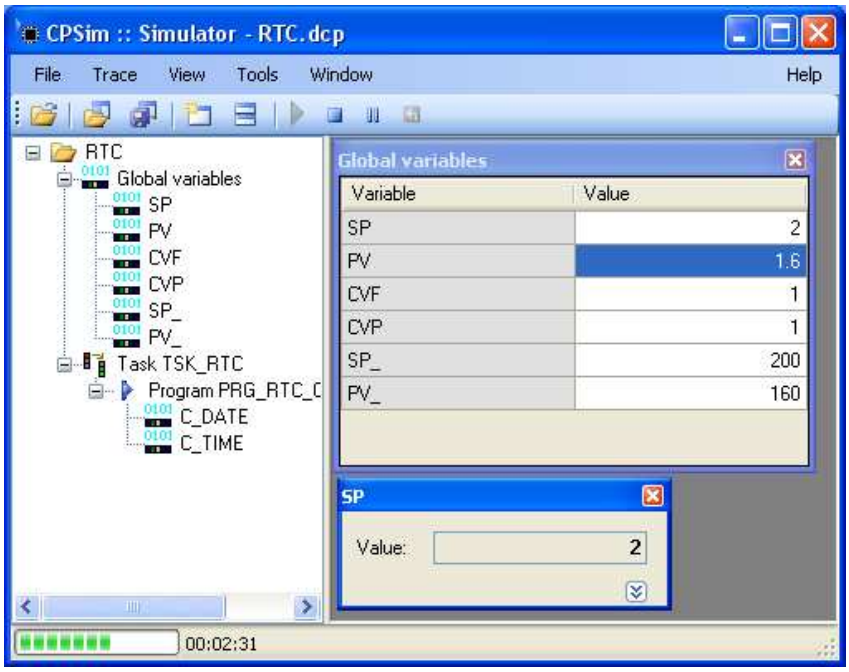
15: determination of the furnace control CVF by comparison of measurement PV and set point SP temperatures, taking into account 0.5°C drop after turning the furnace off.

17: determination of the pump control CVP, switched on all time during the day, at night between 23.00 and 1.00 and when the furnace is on.

21: conversion of REAL to INT after adjustment of the range, to be read by PC.

Simulation

The window shown below corresponds to 9 a.m. The measured temperature 16°C is lower than the set point 20°C, so the furnace is turned on. Pump is also on (daytime). Individual window for the set point SP (controller side) is shown under the list.


CPSim :: Simulator - RTC.dcp
File
Trace
View
Tools
Window
Help
RTC
Global variables
SP
PV
CVF
CVP
SP_
PV_
Task TSK_RTC
Program PRG_RTC_C
C_DATE
C_TIME
Global variables
Variable
Value
SP
2
PV
1.6
CVF
1
CVP
1
SP_
200
PV_
160
SP
Value:
2
00:02:31

USER–DEFINED LIBRARY

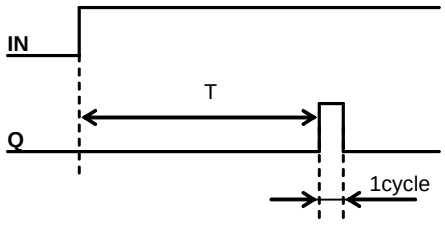
Library as a project

A library with two function blocks will be created:

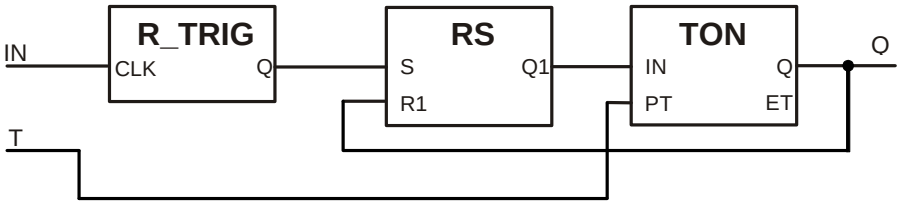
- *FB_AVERAGE* – average of three inputs

$$OUT = \frac{IN1 + IN2 + IN3}{3.0}$$

- *FB_PULSE* – single pulse after time T since rising edge appeared at the input



Pulse may be generated by the following block diagram:



User library is created as a new project with programs, function blocks, functions and global variables (or only some of them).

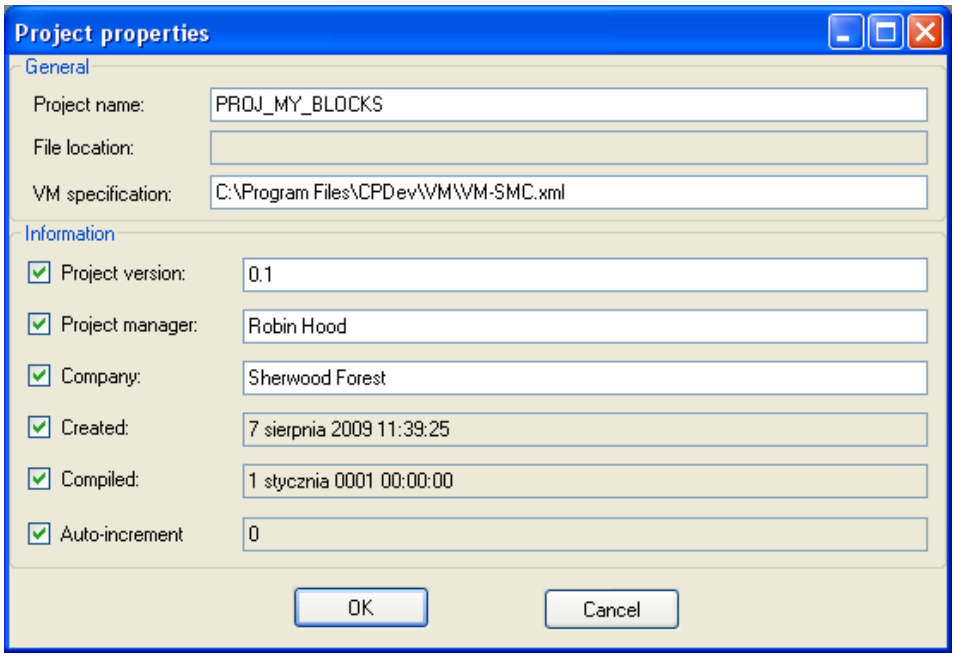
New project

- *File* > *New*

  NoName appears in the project tree.

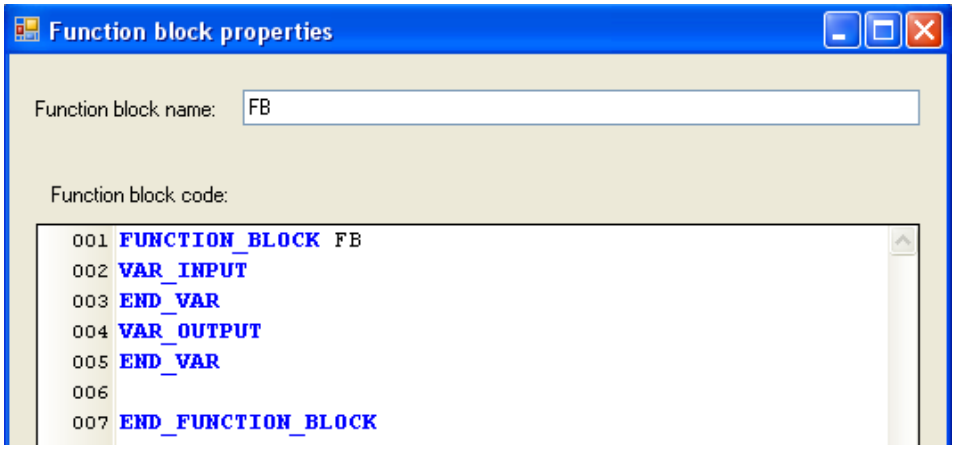
- *NoName* > Context menu > *Properties*

  Enter name in *Project properties*, for instance PROJ_MY_BLOCKS.



New function block

- POU > Context menu > *Add* > *Function block*



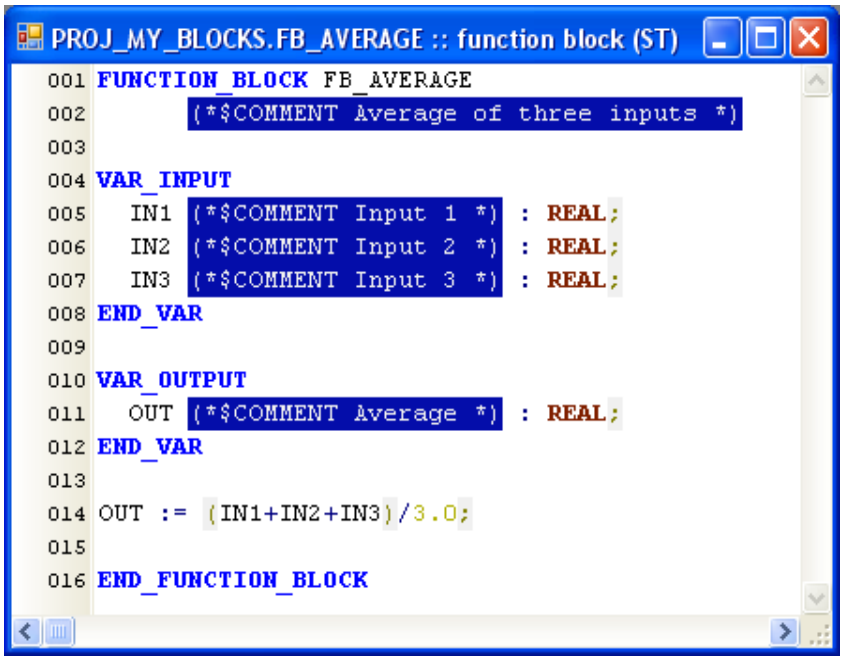
FB_AVERAGE

- *Name*

  Enter FB_AVERAGE. OK inserts the block into project tree.

- *Code*

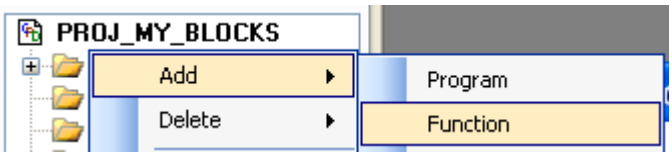
  Double click FB_AVERAGE to open editor window. Directive (*$COMMENT*) is particularly useful for user libraries.



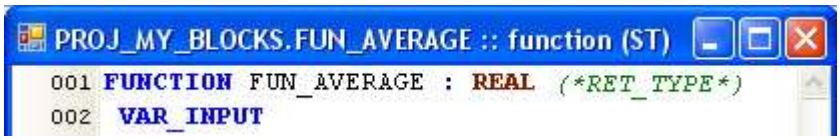
- *Compilation*
  Project > Build
  Correct errors, if any.

Function instead of a block

Since FB_AVERAGE does not store internal state, it may be replaced by a function.



Remaining steps are the same.



FB_PULSE

Blocks from IEC_61131 library will be used to implement the diagram shown at the beginning.

- *Code – part I*
  Local declarations define block instances.

PROJ_MY_BLOCKS.FB_PULSE :: function block (ST)

```
001 FUNCTION_BLOCK FB_PULSE
002       (*$COMMENT Pulse after time T *)
003
004 VAR_INPUT
005    IN (*$COMMENT Rising edge input *) : BOOL;
006    T  (*$COMMENT Time T *) : TIME;
007 END_VAR
008
009 VAR_OUTPUT
010   Q    (*$COMMENT Output *) : BOOL;
011 END_VAR
012
013 VAR
014   TRIG_B: R_TRIG;    RS_B: RS;    TON_B: TON;
015 END_VAR
016
```

- *Input/output names*

  Sometimes you may need to recall declarations of library blocks for input/output names. This can be done in two ways:

  1) Select block in the library folder in project tree. Tip with input/output declarations is briefly presented.

  2) Select the block and press Ctrl+I to get permanent window with the declarations.

IEC_61131.TON :: function block (ST)

```
001 FUNCTION_BLOCK TON
002 VAR_INPUT
003 IN : BOOL;
004 PT : TIME;
005 END_VAR
006 VAR_OUTPUT
007 Q : BOOL;
008 ET : TIME;
009 END_VAR
010  ...
011 END_FUNCTION_BLOCK
```

- *Code – part II*

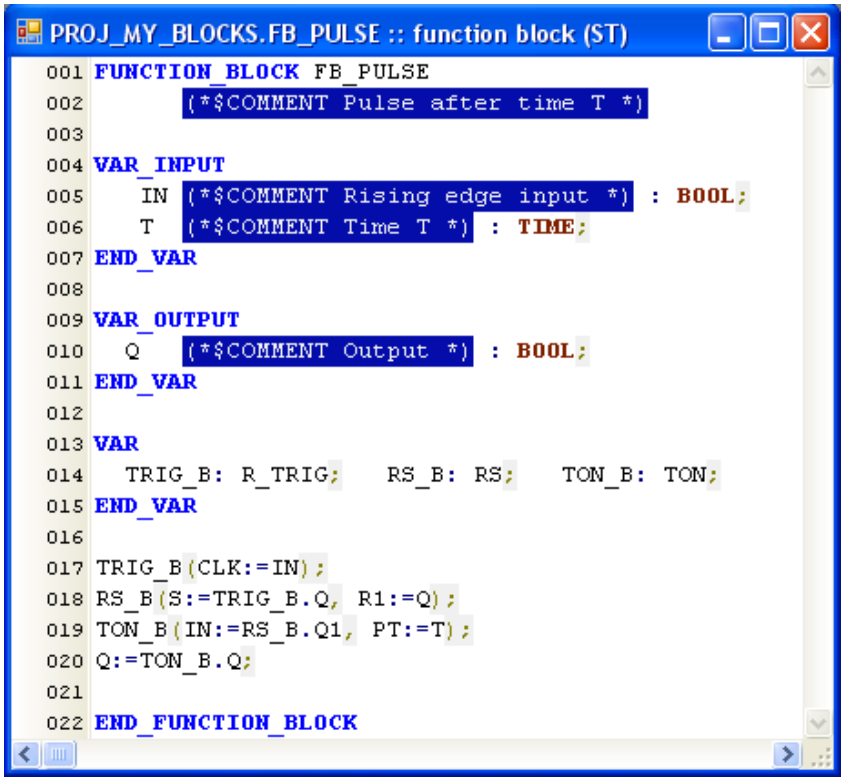
  While entering the code, *autocomplete* option of CPDev editor is available. Ctrl + space opens autocomplete list.

```
017 TRIG_B(CLK:=IN);
018 RS_B(S:=TR
019
020
021
```

Local variable TRIG_B : R_TRIG
Build-in function TRUNC(REAL):DINT;
Build-in function TRUNC(LREAL):LINT;

Compilation of the project after declarations is needed to build up the list (see *Supplements*). *Enter* inserts selected word and closes the list; you may also click the word or click outside. *Esc* closes the list as well.

Final code of FB_PULSE is shown below.



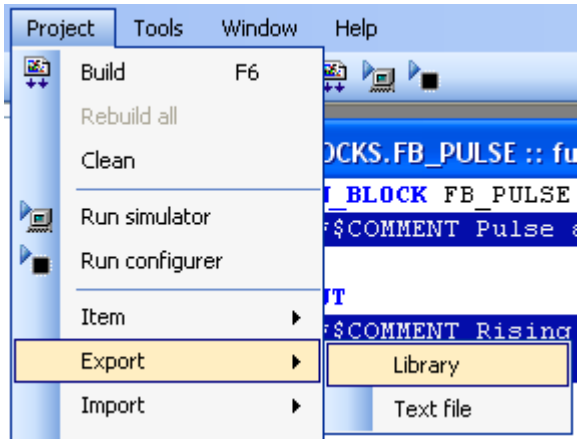
- *Compilation*

  Remark. You could now write a test program as additional POU unit and run it using simulator. However, it will be more natural from user viewpoint if we first export the project as a library, and test it later in another project.

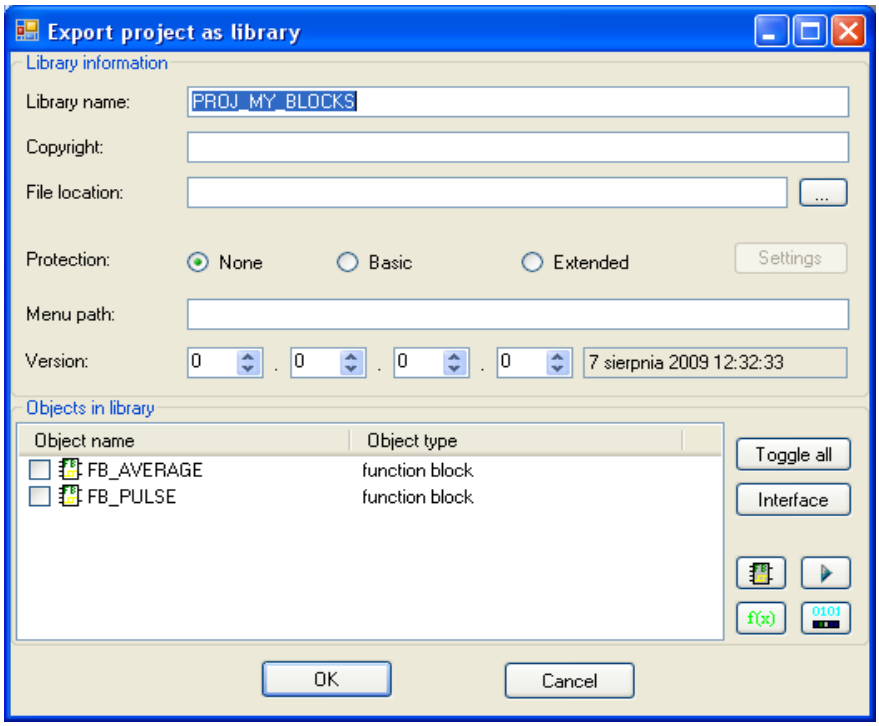
Library export

The project will be exported as semi–compiled library.

- *Project > Export > Library*

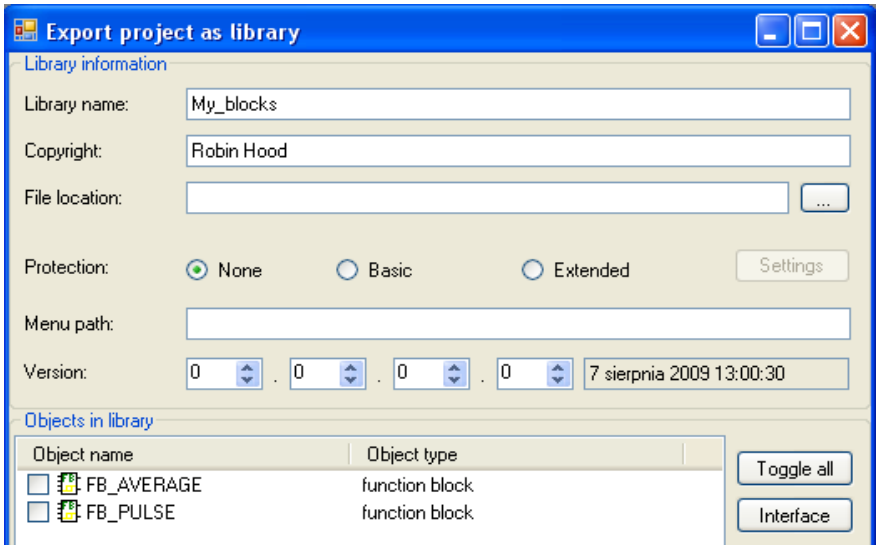


Project name is temporarily used as library name.

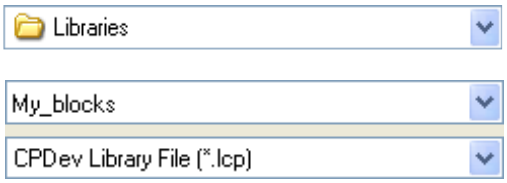


- *Library name*

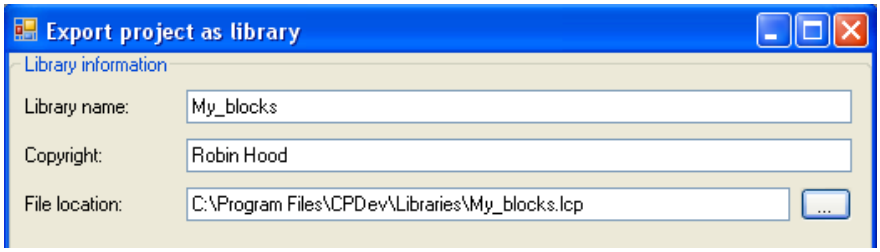
  Enter proper name, here My_blocks, version number and eventually fill in other cells (menu path is reserved for future use in FBD diagrams).



- *Library file location*
  - Click 
  - Select target folder, usually *Libraries*, enter name of library file with *.lcp* extension, here *My_blocks.lcp*, and save.

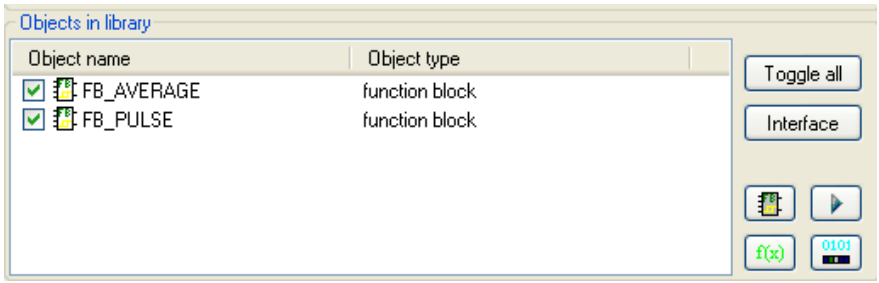


Filename may be the same as library name (but does not have to).



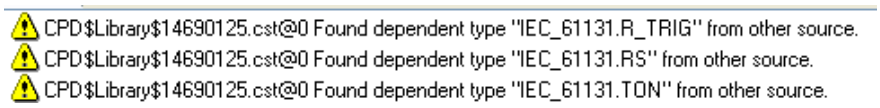
- *Objects for export*

  Options on the left side select exported objects (both here). Button *Toggle all* toggles selected/non–selected, *Interface* recalls input/output declarations, four buttons below select function blocks, programs, functions, and global variables.

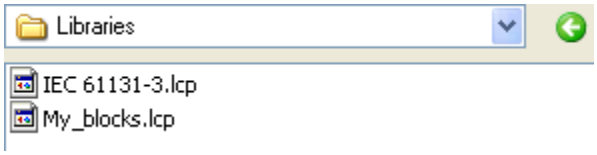


- *Semi–compilation*

  OK compiles selected objects into semi–compiled from (*.lcp* extension; *Project* > *Build* produces binary code). Warnings on non–imported dependencies are not relevant.



If no error occurs, *My_blocks.lcp* is saved in *Libraries* folder.



- *File > Save*

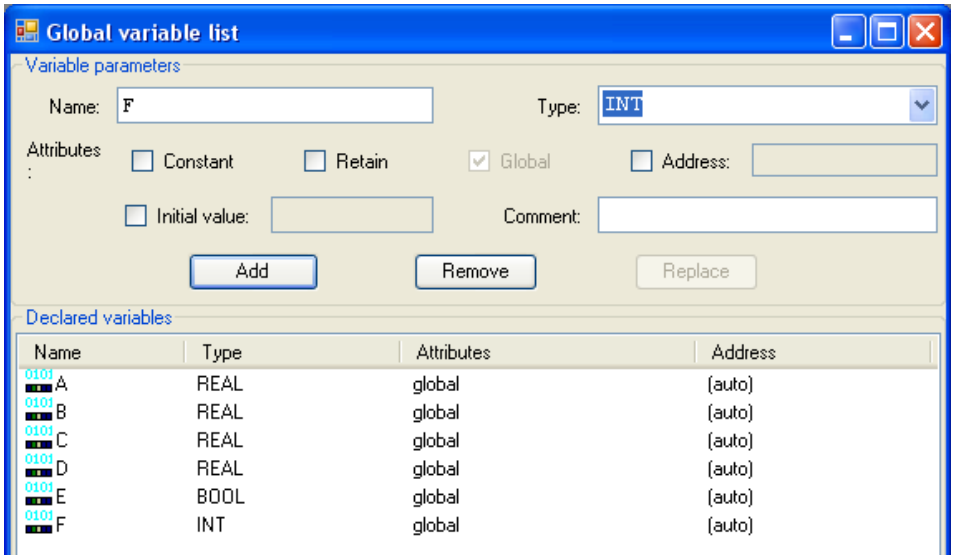
  The original project PROJ_MY_BLOCKS is saved in *.xml* file, for instance in *Proj_My_blocks.xml* here.

Library source code as XML file with original project should be saved for future use.

Testing

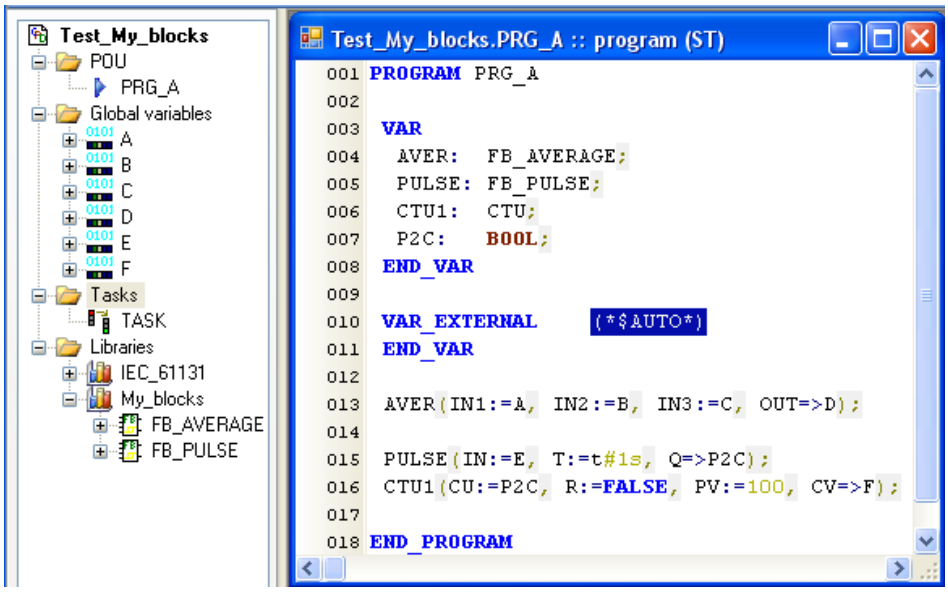
Separate project, here Test_My_blocks, is created. The block FB_AVERAGE will be tested by sample input data and FB_PULSE by counting number of pulses with CTU standard counter.

- *Global variable list*



A, B, C are inputs and D output of FB_AVERAGE, E input to FB_PULSE, and F output of CTU.

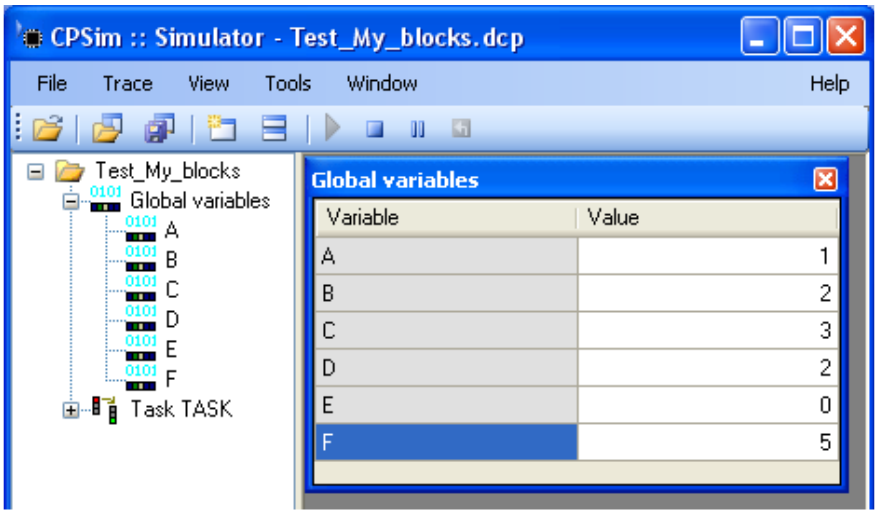
- *Test program*



The project Test_My_blocks uses two libraries, IEC_61131 and My_blocks. The first one is required by the second as dependent library. FB_PULSE and CTU are connected by local variable P2C.

- *Simulation*

  Compile Test_My_blocks, run CPSim, enter 1, 2, 3 for A, B, C, and set E five times alternately to 0, 1. The variable list of the simulator looks then as follows:



Library extension

It is done by supplementing the library source code (*Proj_My_Blocks.xml*) with new components. Export of the extended library is repeated by *Project* > *Export* > *Libraries*. Previous content of semi–compiled file (*My_blocks.lcp)* is replaced by the new one in *Libraries*.

ST LANGUAGE OVERVIEW

This overview is for the readers with some experience in high level language programming (C, Pascal, scripts). More on ST can be found in John K. H. and Tiegelkamp M.: *IEC 61131–3: Programming Industrial Automation Systems*, Springer, 2001, or elsewhere.

IEC 61131-3 standard

Programming languages

The IEC 61131-3 standard (IEC below) defines five languages for controller programming:

- structured text ST
- instruction list IL
- ladder diagram LD
- function block diagram FBD
- sequential function chart SFC

ST, a high level language similar to Pascal, is a basis for CPDev package.

Language components

Common components of the five languages are the following:

- data types, e.g. BOOL, INT, REAL
- program organization units POU
- configuration elements.

POU units

Three kinds of POUs are defined in IEC:

- programs
- functions blocks
- functions

Whereas a function for the same input data always yields the same output, output of a block may be different, as it depends on actual state of this block. Therefore declaration of block instance to allocate memory for the state must precede usage of the block.

Configuration elements

Installation and configuration of programs is supported by:

- configuration
- tasks
- access paths
- resources
- global variables

Configuration is called a *project* in CPDev. Tasks and global variables are sufficient for configuration of single controller. Programs belong to tasks.

Structure of POUs

Structure of programs, functions and function blocks is the same, i.e.:

- POU type and name
- declaration of variables and function block instances
- program code

PROGRAM, FUNCTION BLOCK and FUNCTION keywords define POU type. Global and local variables are declared separately. Block instances are declared together with local variables (within VAR…END_VAR).

Identifiers (names)

They begin with a letter or underscore sign _. IEC standard does not make difference between lower and upper case letters, even in keywords. So the following identifiers (names) are the same: 1) START, Start, start (variable), 2) THEN, Then, then, 3) END_VAR, end_var.

CPDev automatically converts lower case letters into upper case (although the editor still shows them as originally entered).

Identical names in different libraries

Names must be unique within a project or library. If the same name, e.g. TON, denotes another block in another library than IEC_61131, declarations of corresponding instances in the program must indicate the library, so:

IEC_61131.TON Another_lib.TON

Otherwise *Multiple name found* or *Ambiguous...* error appears. Actual name preceded by name of the project or library is called *full name* in CPDev.

Data types and variables

Elementary data types

No.	Name	Data types	Size and range
1	BOOL	Boolean	1B (FALSE, TRUE ⇔ 0,1)
2	BYTE	byte	1B (0 ... 255)
3	WORD	word	2B (0 ... 65535)
4	DWORD	double word	4B (0 ... 2^{32}-1)
5	LWORD	long word	8B (0 ... 2^{64}-1)
6	SINT	short integer	1B (-128 ... 127)
7	INT	integer	2B (-32768 ... 32767)
8	DINT	double integer	4B (-2^{31} ... 2^{31}-1)
9	LINT	long integer	8B (-2^{63} ... 2^{63}-1)
10	USINT	unsigned short integer	1B (0 ... 255)
11	UINT	unsigned integer	2B (0 ... 65535)
12	UDINT	unsigned double integer	4B (0 ... 2^{32}-1)
13	ULINT	unsigned long integer	8B (0 ... 2^{64}-1)
14	REAL	real	4B, IEEE-754 format
15	LREAL	long real	8B, IEEE-754 format
16	TIME	duration	4B (-T#24d20h31m23s648ms... T#0s...#24d20h31m23s647ms)
17	DATE	date	4B (0001-01-01 ... 9999-12-31)
18	TIME_OF_DAY	time of day	4B (00:00:00.00... 23:59:59.99)
19	DATE_AND_TIME	date and time	8B (connection of DATE and TIME_OF_DAY types)
20	STRING	character string	variable length

STRING, USINT, UINT, UDINT and ULINT types are not implemented in CPDev yet..

Universal types

Groups of elementary types collected according to applications are called universal.

<table>
<tr><th colspan="5">ANY</th></tr>
<tr><td>ANY_BIT</td><td colspan="3">ANY_NUM</td><td>ANY_DATE</td><td rowspan="6">TIME,
STRING and derived types</td></tr>
<tr><td rowspan="5">BOOL
BYTE
WORD
DWORD
LWORD</td><td colspan="2">ANY_INT</td><td>ANY_REAL</td><td rowspan="5">DATE
TIME_OF_DAY
DATE_AND_TIME</td></tr>
<tr><td>INT</td><td>UINT</td><td rowspan="4">REAL
LREAL</td></tr>
<tr><td>SINT</td><td>USINT</td></tr>
<tr><td>DINT</td><td>UDINT</td></tr>
<tr><td>LINT</td><td>ULINT</td></tr>
</table>

Constants (literals)

Examples of constants of the types used most often are given below:

BOOL: TRUE, BOOL#1
INT: 13, INT#-1
REAL: 4.1415, REAL#18, 1.2E-6
TIME: T#1m3s250ms
TIME_OF_DAY: TOD#06:00:00

Single numerical constant without the dot is of type INT, whereas constant with the dot is of type REAL.

Other types than INT, REAL are chosen by putting type name and sign # before the number, e.g. DINT#-13, REAL#1.

Nondecimal numbers

Format of nondecimal number involves: 1) base of numerical system, e.g. 2, 8, 16, etc., 2) sign #, 3) alphanumeric string as value. For instance, 2#11111111, 8#377, 16#FF denote 255 decimal. WORD#16#00FF is another option (leading zeroes are not necessary).

Initial values

Default initial values are in the table:

Data type	Initial value
ANY_BIT, ANY_INT	0
ANY_REAL	0.0
TIME	T#0s
DATE	D#0001-01-01
TIME_OF_DAY	TOD#00:00:00

DAY_AND_TIME	DT#0001-01-01-00:00:00
STRING	'' (empty)

Other initial values are declared by means of assignment sign `:=,` for instance

```
lamp: BOOL := TRUE;
```

Attributes

CPDev package supports two attributes of variables:

RETAIN CONSTANT

RETAIN declares a retentive variable whose value is kept in memory during power brake (for warm restart). CONSTANT variable cannot be changed. Initial value of retentive variable applies for cold start only, whereas initial value of non–retentive one is also used for warm restart.

Declarations of variables

IEC standard defines a few kinds of variable declarations:

VAR VAR_IN_OUT VAR_ACCESS
VAR_INPUT VAR_EXTERNAL
VAR_OUTPUT VAR_GLOBAL

VAR declares local variables and function block instances. VAR_INPUT, VAR_OUTPUT and VAR_IN_OUT are used in function blocks and functions. VAR_EXTERNAL declares usage of variables defined in *Global variable list* (or, equivalently, by VAR_GLOBAL; see *For advanced users*). END_VAR terminates each kind of declaration.

Declarations VAR_EXTERNAL are allowed in programs only (not in function blocks or functions). RETAIN attribute may appear in *Global variable list* (or VAR_GLOBAL), in VAR and VAR_OUTPUT. VAR_ACCESS is not supported by CPDev.

Allocation of global variables

Allocation of single variable is determined by AT keyword followed by concatenation of the sign %, size prefix and logical address, e.g.:

```
pump AT %B0009 : BOOL;
```

Global variable list involves *Address* option instead of AT. Size prefixes are shown in the table.

Prefix	Data types	Size
B, X, none	BOOL, BYTE, SINT, USINT	1B
W	WORD, INT, UINT	2B
D	DWORD, DINT, UDINT, REAL, TIME, DATE, TIME_OF_DAY	4B
L	LWORD, LINT, ULINT, LREAL, DATE_AND_TIME	8B

Prefixes B, X and leading zeroes of the address may be dropped (as %9 for the pump above). Group declaration

```
A, B, C  AT %W0000:INT;
```

is equivalent to three individual declarations

```
A AT %W0000:INT;   B AT %W0001:INT;   C AT %W0002:INT;
```

The keyword AT *cannot* be used for local variables which are located automatically.

Memory addresses

Compiler determines number of bytes from size prefix and assigns memory for the variable beginning from the byte with address

```
byte address := logical address * size,
```

(logical address from *Global variable list* or AT declaration). For instance, declaration

```
counter AT %W0007: INT;
```

means that `counter` occupies 2·7=14th byte (and 15th). So the addresses of first bytes where variables are located have the following properties

Prefix	Byte address
B, X, none	number after prefix
W	even number (address)
D	number divisible by 4
L	number divisible by 8

Remark. Addresses of variables are needed to configure communication with host computer. They are shown in *Project report*.

If global variable is declared without selecting *Address* option in *Global variable list* (or without AT) the compiler locates it automatically filling empty spaces. Text *auto* appears in the list.

If variables are declared in groups, some of the addresses may overlap since the compiler checks whether address for first variable is free, and not the area for the whole group. Warning appears in case of overlapping.

Function block declaration

As mentioned before, instances of function blocks are declared locally within VAR ... END_VAR. For instance, if DELAY is going to be an instance of the TON block, it must be declared by:

```
DELAY : TON;
```

Programming in ST

Programs, function blocks and functions

The following keywords begin and terminate declarations of POU units:

POU	Limiting keywords
Program	PROGRAM ... END_PROGRAM
Function block	FUNCTION_BLOCK ... END_FUNCTION_BLOCK
Function	FUNCTION ... END_FUNCTION

A program may call (invoke) function blocks and functions; function block may call other blocks or functions. Recursive calls are not allowed.

ST language statements

They involve assignment, selections, loops, exits, function and function block calls (invocations).

- *Assignment*: variable := expression;
  Statements is terminated by semicolon ;.
- *Selections: IF, CASE*

IF

```
IF A>B THEN
B := A;
ELSIF A<B THEN
A := B;
ELSE A := 0; B:= 0;
END_IF
```

Semicolons are not necessary after END_IF, END_VAR and other ENDs.

CASE

```
CASE A OF
1: B:=1; A:=2;
2..10: A:=A+1;
       B:=A*1000;
11,13,15..21: A:=A+2;
              B:=A*10;
ELSE A:=1; B:=9999;
END_CASE
```

Selection variable must by of integer type (ANY_INT, BYTE, WORD...). Entries are constant values (or CONSTANT variables) of selector type, otherwise *Cannot match primitive function...* error appears (in line 0).

- *Loops: FOR, WHILE, REPEAT*

FOR

```
counter := 0;
FOR i:=1 TO 10 DO
counter:= counter+i;
END_FOR
```

WHILE

```
WHILE st1 OR st2
DO
pump := FALSE;
alarm := TRUE;
END_WHILE
```

REPEAT

```
REPEAT
B := B+1;
UNTIL B>10
END_REPEAT
```

If control variable of FOR loop must be increased by other number than 1, then BY… component is included into the statement, as in

```
FOR i:=1 TO 10 BY 2  DO ... END_FOR
FOR i:=10 TO 1 BY –1 DO ... END_FOR
```

(BY must be followed by a constant or CONSTANT variable).

- *E x i t s : EXIT, RETURN*
  EXIT interrupts FOR, WHILE or REPEAT loop. RETURN provides early exit from a function or function block (before `END`).

EXIT	RETURN

`WHILE i>0 DO`
`l := l+1;`
`IF l>MAX_l THEN`
`EXIT;`
`END_IF`
`i := i-1;`
`END_WHILE`
	`FUNCTION LINE: REAL`
	`VAR_INPUT`
	`a,x,b: REAL;`
	`END_VAR`
	`LINE:=a*x+b;`
	`RETURN;`
	`END_FUNCTION`

- *F u n c t i o n*
  Standard and system functions (next chapter) are called directly. To call user–defined functions corresponding libraries must be imported. Function call statement may look as follows:

```
Y := LINE(A1,X1,B1);
```

- *F u n c t i o n b l o c k*
  Suppose DELAY denotes instance of the standard timer TON. The following statements invoke DELAY and transfer its outputs:

```
DELAY(IN:=_input, PT:=t#5s);
motor := DELAY.Q;
bargraph := DELAY.ET;
```

The outputs can also be transferred directly in the call statement by means of the sign =>, i.e.:

```
DELAY(IN:=_input, PT:=t#5s, Q=>motor, ET=>bargraph);
```

Order of inputs and outputs does not matter in the call.

ST language operators

Expressions consist of operators and operands. The following table lists operators with priorities in descending order.

Symbol	Description	Function
( )	parentheses	–
F(x)	function evaluation	F(x)
**	exponentiation	EXPT
-	arithmetic negation	NEG
NOT	Boolean negation	NOT
*	multiplication	MUL
/	division	DIV

MOD	modulo	MOD
+	addition	ADD
-	subtraction	SUB
<, >, <=, >=	comparison	LT,...,GE
=	equality	EQ
<>	inequality	NQ
AND, &	Boolean multiplication	AND
XOR	exclusive OR	XOR
OR	Boolean sum	OR

The operators separated above by the dashed lines have the same priority, so they are executed in the order defined by expression (from left to right). Operators can be replaced by functions given in the table, as in:

```
x1 AND x2        AND(x1,x2)
```

Single–dimensional arrays

Program part

```
VAR
 T:ARRAY[0..5] OF INT;
END_VAR

FOR I:=1  TO 5 DO
 T[I-1]:=T[I];
END_FOR
T[5]:=A; S:=0;
FOR I:=0 TO 5 DO
 S:=S+T[I];
END_FOR
S:=S/I;
```

Compiler accepts single–dimensional arrays declared as *local variables*. The arrays cannot be used as inputs or outputs. Program on the left implements moving average filter for variable A.

FUNCTIONS

IEC standard defines large set of functions divided into groups. Most of IEC functions are available in CPDev (several data types are not supported, e.g. STRING).

Mathematic and logic functions

Group	Name	Operation	I/O types
Arithmetic	ADD*	add	ANY_NUM
	SUB	subtract	
	MUL*	multiply	
	DIV	divide	
	MOD	modulo	
	EXPT	exponentiation	
	NEG	negation	SINT, INT, DINT LINT, REAL
Numeric	ABS	absolute value	REAL, LREAL
	SQRT	square root	
	LN	natural logarithm	
	LOG	logarithm base 10	
	EXP	natural exponential	
	SIN	sine	
	COS	cosine	
	TAN	tangent	
	ASIN	arc sine	
	ACOS	arc cosine	
	ATAN	arc tangent	
Boolean	AND*	logic product	ANY_BIT
	OR*	logic sum	
	XOR*	exclusive OR	
	NOT	complement	
Bit shift	SHL	shift left, zero–filled	BYTE, WORD DWORD, LWORD
	SHR	shift right, zero–filled	
	ROL	left–rotated, circular	
	ROR	right–rotated, circular	
Comparison	GT	greater	ANY
	GE	greater or equal	
	EQ	equal	
	LT	less	
	LE	less or equal	
	NE	not equal	
Time	ADD	add	TIME
	SUB	subtract	

Explanations

- Star * after function name indicates varying number of arguments (up to 15).
- Bit shift functions have two arguments, ANY_BIT (without BOOL) and INT.
- Other operations on TIME data can be executed by conversion to REAL or DINT.
- Additional function RANDOML (not listed above) returns REAL number in 0.0...1.0 for rectangular probability distribution.

Selection functions

All elementary types are allowed (ANY).

Name	Operation	Description
SEL	binary selector (one of two)	SEL (G, IN0, IN1) OUT:=IN0 for G=FALSE OUT:=IN1 for G=TRUE Types: G – BOOL; IN0, IN1 - ANY
MAX	maximum	MAX (IN1, IN2)
MIN	minimum	MIN (IN1, IN2)
LIMIT	limiter	LIMIT (MN, IN, MX) OUT:=MIN (MAX (IN, MN), MX)
MUX*	multiplexer	MUX (K, IN0, IN1, …)
		OUT:=INi for K=i
		Types: K - INT, INO, IN1, … - ANY

MUX may switch up to 15 inputs.

Conversions

If the following table does not include a particular conversion, two steps are needed with some intermediate type.

Input	Function name	
INT	INT_TO_REAL INT_TO_BOOL	INT_TO_DINT INT_TO_WORD
REAL	REAL_TO_INT REAL_TO_LREAL TRUNC	REAL_TO_TIME ROUND
DINT	DINT_TO_REAL DINT_TO_DWORD	DINT_TO_TIME DINT_TO_INT
TIME	TIME_TO_DINT	TIME_TO_REAL
BYTE	BYTE_TO_SINT	
WORD	WORD_TO_INT	
BOOL	BOOL_TO_INT	
SINT	SINT_TO_BYTE	
LREAL	LREAL_TO_REAL TRUNC	 ROUND

LINT	LINT_TO_LWORD
DWORD	DWORD_TO_DINT
LWORD	LWORD_TO_LINT

Remarks. Depending on argument type, TRUNC and ROUND convert either to DINT or LINT. DEPR_INT_TO_DINT (not listed) converts INT to DINT by repeating MSB bit.

Real time

CPDev package provides:
- system time as TIME data
- RTC clock read and write
- daytime and date components
- days of the week.

System time and RTC functions are given in the table. CUR_TIME increments system time up to 24 days (a little more), then resets it to „negative" 24 days, and so on. Time interval is determined as the difference between two CUR_TIME readings.

Name	Function returns	Result type
CUR_TIME	current system time	TIME
READ_RTC	absolute time read from RTC clock	DT
WRITE_RTC	RTC clock update status	BOOL
GET_TST	absolute time of task start	DT
TASK_CYCLE	task cycle duration	TIME

Explanations

- READ_RTC, WRITE_RTC and GET_TST functions operate on DATE_AND_TIME data. WRITE_RTC returns status flag of RTC update operation (RTC functions depend on hardware platform).
- Task start time returned by GET_TST is used more often than the time returned by READ_RTC.
- TASK_CYCLE returns value set in the project (*Task properties* window).

Daytime and date components

Structure of DATE_AND_TIME data in shown below. Successive bytes denote: CC – hundredth parts of a second, SS – second, NN – minute, HH – hour, DD – day, MM – month, YY+YY – year.

DATE_AND_TIME

	CC	SS	NN	HH	DD	MM	YY	YY
Byte no.	0	1	2	3	4	5	6	7
	TIME_OF_DAY				DATE			

Functions from GET_HUNDSEC to GET_YEAR return INT value. Two types of input arguments are supported.

Name	Function returns	Argument type
GET_HUNDSEC	hundredths of second	DT, TOD
GET_SECOND	second	DT, TOD
GET_MINUTE	minute	DT, TOD
GET_HOUR	hour	DT, TOD
GET_DAY	day	DT, D
GET_MONTH	month	DT, D
GET_YEAR	year	DT, D
GET_DAYOFWEEK	day of week	DT, D

Status word

Bits of status word returned by GET_STATUS_WORD1 denote:

Bit	Mask	Description
0	16#01	task cycle time exceeded in the last run
1	16#02	read array index out of range
2	16#04	cold start (0 means normal operation or warm restart)

FUNCTION BLOCK LIBRARIES

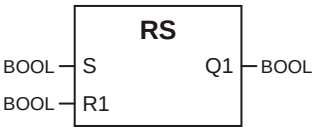
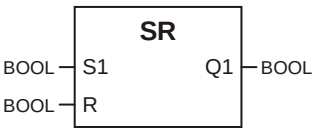
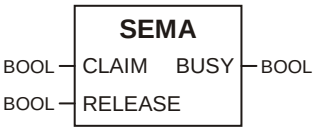
CPDev package involves two libraries with function blocks, IEC_61131 and Basic_blocks.

IEC_61131 library

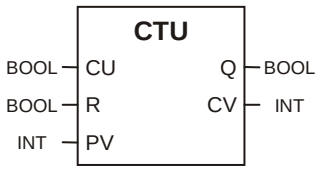
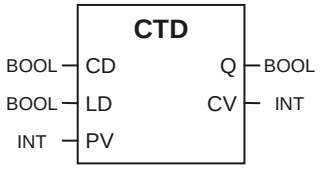
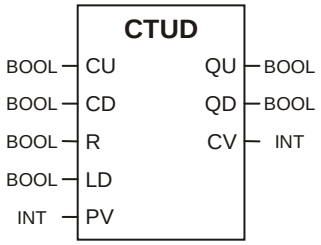
Symbols of inputs and outputs are as in the IEC standard, so:

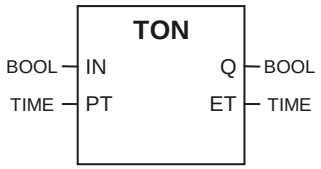
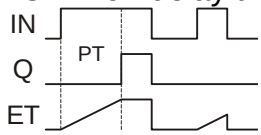
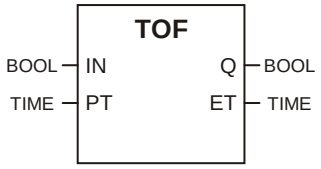
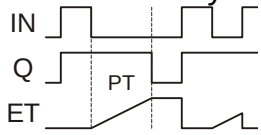
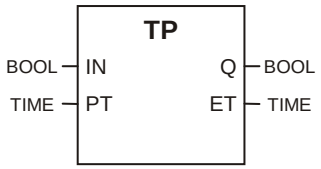
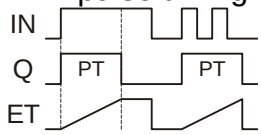
- R – reset input (logic)
- S – set input
- CLK↑ – rising edge at CLK input
- Q – output of BOOL type

Initial values of all inputs are zero.

Bistable elements	
	**RS** – RS flip-flop Q1 = NOT R1 AND ($Q1_{n-1}$ OR S)
	**SR** – SR flip-flop Q1 =S1 OR (NOT R AND $Q1_{n-1}$)
	**SEMA** – semaphore BUSY = TRUE for CLAIM=TRUE BUSY = FALSE for RELEASE=TRUE and CLAIM=FALSE

Edge detectors	
	**R_TRIG** – rising edge detector Q = ↑↓ for CLK↑
	**F_TRIG** – falling edge detector Q = ↑↓ for CLK↓

Counters	
	**CTU** – up counter CV = CV+1 for CU↑, CV<PV and R=FALSE CV = 0 for R=TRUE Q = TRUE for CV=PV
	**CTD** – down counter CV = CV-1 for CD↑, CV>0 and LD=FALSE CV = PV for LD=TRUE Q = TRUE for CV=0
	**CTUD** – up-down counter CV = CV+1 for CU↑, CV<PV and R=LD=FALSE CV = CV-1 for CD↑, CV>0 and R=LD=FALSE CV = 0 for R=TRUE CV = PV for LD=TRUE and R=FALSE QU = TRUE for CV=PV QD = TRUE for CV=0

Timers	
	**TON** – on-delay timing 
	**TOF** – off-delay timing 
	**TP** – pulse timing 

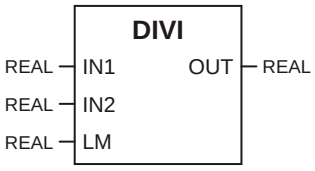
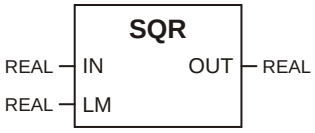
Remark. Recall that READ_RTC, WRITE_RTC and GET_TST functions handle RTC clock in CPDev.

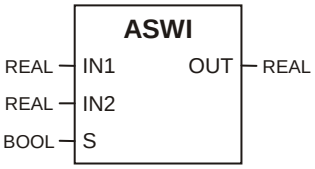
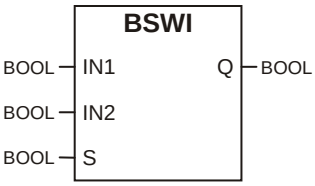
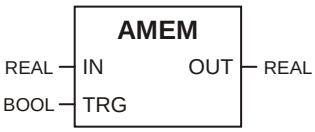
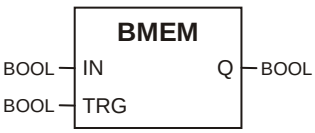
Basic_blocks library

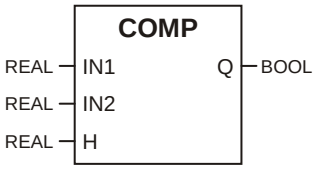
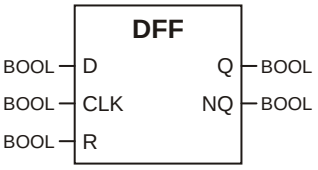
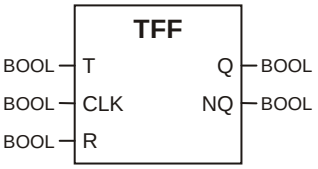
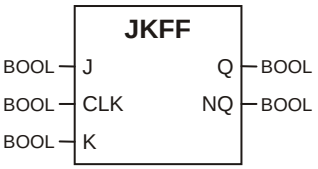
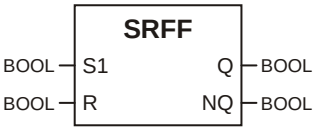
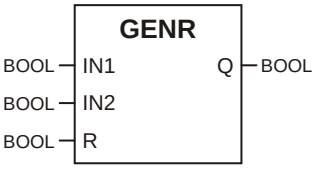
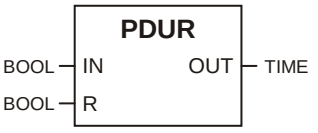
Notation:

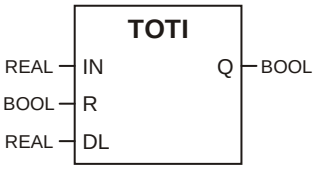
R	–	reset input for arithmetic and logic, or to set another value
S	–	selection or switching input, set input for flip–flops
IN↑	–	rising edge at IN input; edge at t_0 is denoted by t_0:IN↑
Q	–	output of BOOL type
OUT	–	output of REAL,TIME or other type.

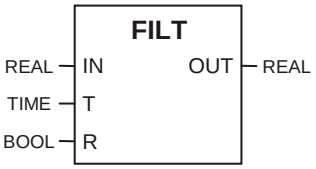
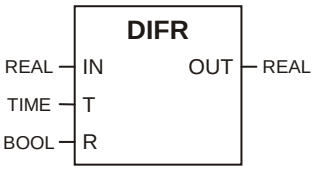
Initial values of all inputs are zero.

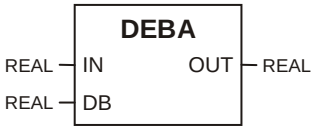
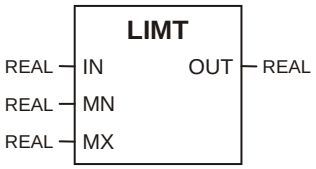
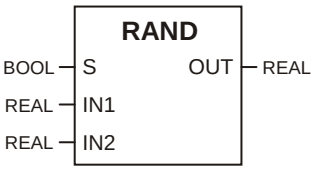
Mathematic blocks	
	**DIVI** – division with limited divisor OUT = IN1/IN2 LM – limit of IN2 before 0 If |IN2|<LM, then OUT=IN1/(±LM); ± is IN2 sign.
	**SQR** – square root with linear initial part OUT = $\sqrt{IN}$ for IN ≥ LM OUT = IN/$\sqrt{LM}$ for IN < LM

Switches, selectors	
	**ASWI** – analog switch OUT = IN1 for S = FALSE OUT = IN2 for S = TRUE
	**BSWI** – binary switch Q = IN1 for S = FALSE Q = IN2 for S = TRUE
	**AMEM** – analog memory OUT = IN for TRG = FALSE OUT = IN(t_0) for TRG = TRUE, t_0: TRG↑
	**BMEM** – binary memory Q = IN for TRG = FALSE Q = IN(t_0) for TRG = TRUE, t_0: TRG↑

	**COMP** – analog comparator with hysteresis Q = FALSE for (IN1-IN2) < -H/2 Q = TRUE for (IN1-IN2) > H/2
Flip–flops, pulsers	
	**DFF** – D flip-flop Q = D for CLK↑ and R = FALSE Q = FALSE for R = TRUE NQ = NOT Q
	**TFF** – T flip-flop Q = NOT Q_{n-1} for CLK↑ and R = FALSE Q = FALSE for R = TRUE NQ = NOT Q
	**JKFF** – JK flip-flop Q = f(J,K) for CLK↑ NQ = NOT Q J K Q 0 0 Q_{n-1} 0 1 0 1 0 1 1 1 $\overline{Q}_{n-1}$
	**RSFF** – RS flip-flop As RS in IEC, but with additional NQ output (NOT Q).
	**SRFF** – SR flip-flop As SR in IEC, but with additional NQ output (NOT Q).
	**DELS** – delay by one step (cycle) $Q_n = IN_{n-1}$
	**GENR** – alarm generator Q = ↑↓↑↓ for R = FALSE Q = FALSE for R = TRUE Frequency determined by IN1, IN2. IN1 IN2 Cycles 0 0 1 0 1 2 1 0 4 1 1 8
	**PDUR** – pulse duration OUT = 0 for IN↑ or R = TRUE OUT = t for IN = TRUE and R = FALSE

	**TOTI** – totalizer Q = ↑↓ (impulse) for $\int_{t_0}^{t} IN1(\tau)d\tau = \Delta$ R – integral reset DL – time interval Δ for integration

Filters	
	**FILT** – lag filter OUT = $\frac{1}{Ts+1}$ IN for R = FALSE OUT = IN for R = TRUE
	**DIFR** – lead filter (*differentiation*) OUT = $\frac{Ts}{Ts+1}$ IN for R = FALSE OUT = IN for R = TRUE

Others	
	**DEBA** – dead-band OUT = 0 for |IN| < DB OUT = IN ± DB for |IN| ≥ DB (- for IN>0, + for IN<0) OUT = IN for DB < 0
	**LIMT** – limiter OUT = IN for MN<IN<MX OUT = MN for IN<MN OUT = MX for IN>MX
	**RAND** – random OUT = random S = FALSE – normal distribution N(IN1, IN2) IN1 – average value IN2 – standard deviation S = TRUE – rectangular distribution <IN1, IN2> IN1 – low limit IN2 – upper limit

Remark. ASWI, BSWI and LIMT blocks can be replaced by SEL and LIMIT functions (see earlier). SEL automatically recognizes type of inputs.

System blocks

They are ”always available”, so no library is needed.

- *Alarms*

  R – reset input
  Q – alarm output

  Alarm condition is indicated by TRUE at the output Q. Setting R to TRUE cancels the alarm.

Alarms blocks	
	Warm restart (after power brake)
	Cold start (memory cleared, initial values)

Cold start is also initiated when memory test detects data error. Global variables are then set to initial values.

Example

Declarations

```
VAR
STATE:APON;  RESET:BOOL; ALARM:BOOL;
END_VAR;
```

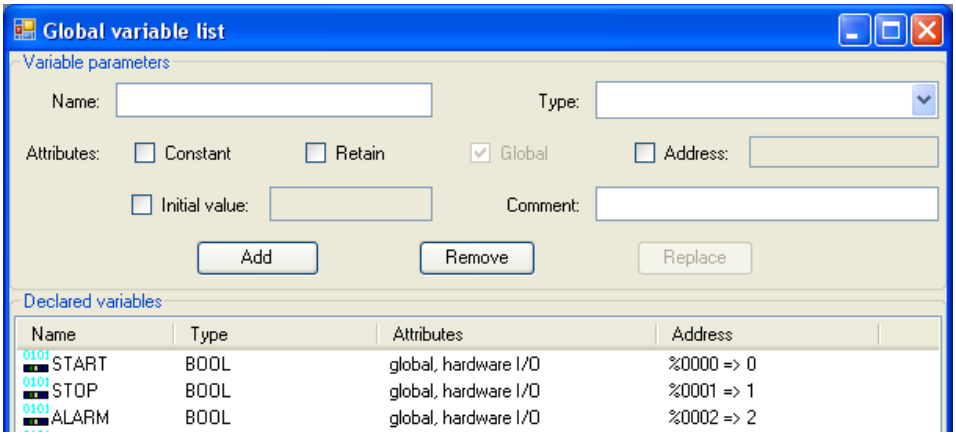
Usage

```
RESET:=FALSE; STATE(R:=RESET); ALARM:=STATE.Q;
```

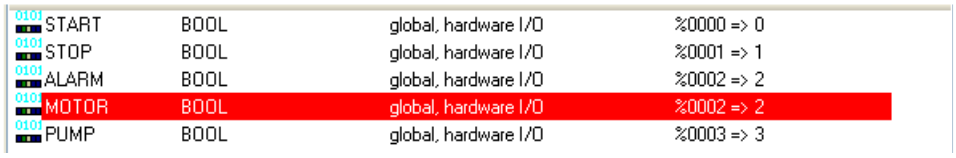
SUPPLEMENTS

Correcting variable list

Suppose the *Global variable list* looks initially as follows:

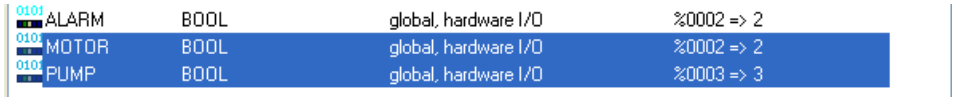


- *Incorrect address*
  New group of two variables, MOTOR and PUMP, is declared, the first one with wrong address 0002. Clicking *Add* supplements the list with the two variables, however the line MOTOR is shown in red indicating address collision.

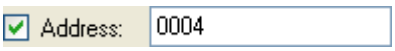


  As in the START_STOP project, MOTOR and PUMP should be located at 0008, 0009.

- *Group selection*
  Select the lines to be corrected, the second one with Shift or Ctrl. Names of variables, types and addresses appear in the upper cells (cell *Type* would be empty for different types).

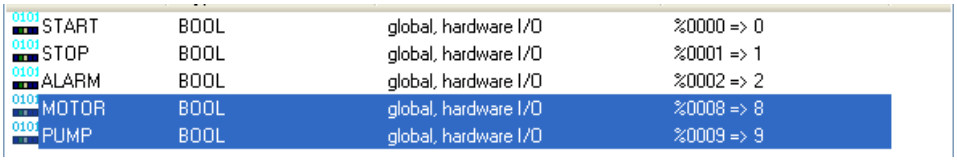


- *Corrections*
  Selection of *Address* option automatically displays first free address for the colliding MOTOR, so 0004 here.



  If you pressed *Replace* now, PUMP would remain at 0003 and MOTOR placed at 0004. However, we want 0008 instead of 0004.

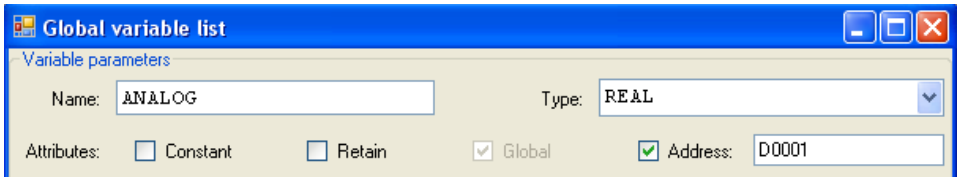
Pressing *Replace* corrects the variable list accordingly.



Note that five bytes from 0003 to 0007 remain empty.

Filling empty areas

Suppose we need another REAL variable called ANALOG. Enter name and type, select *Address* option. First free address D0001 is then indicated.

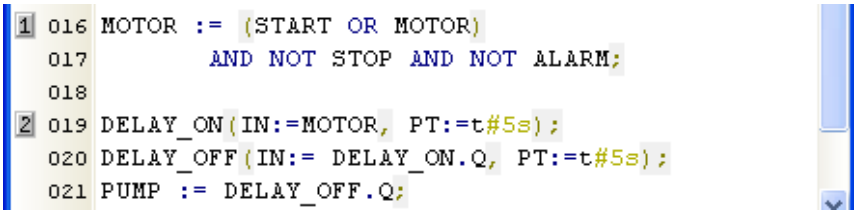


Since ANALOG occupies four bytes (REAL), so the address of its first byte is 0001*4=0004. Pressing *Add* displays the following list

Former empty area is almost full now.

Marks

Small rectangles with digits indicating portions of large programs, to improve clarity and navigation, are called marks (or bookmarks). Portion of a code with two marks is shown below.



The following shortcuts handle marks:

- Shift + Ctrl + 0,...,9 – create a mark 0,...,9 at the line indicated by the cursor
- Ctrl + 0,...,9 – place cursor at the line with mark 0,...,9

Key shortcuts

Shortcuts	Operation
Ctrl+Up	Scroll line up
Ctrl+Down	Scroll line down
Ctrl+PgUp	Scroll screen up
Ctrl+PgDown	Scroll screen down
Ctrl+Home	Editor top
Ctrl+End	Editor end
Ins	Toggle insert/enter mode
Ctrl+Ins	Copy selected part
Shift+Del	Delete selected part
Shift+Ins	Paste from clipboard
Ctrl+Bksp	Remove last word
Alt+Bksp	Undo
Shift+Alt+Bksp	Redo

Shortcuts	Operation
Shift+Ctrl+I	Block indent
Shift+Ctrl+U	Block unindent
Ctrl+M	Break line
Ctrl+H	Insert line
Ctrl+T	Delete word
Ctrl+G	Delete line
Shift+Ctrl+Y	Delete till end of line
Ctrl+0,...,9	Go to mark 0,...,9
Shift+Ctrl+0,...,9	Set mark 0,...,9
Shift+Ctrl+N	Select by lines
Shift+Ctrl+C	Select by columns
Shift+Ctrl+L	Select full lines
Shift+Ctrl+B	Match brackets

Errors, warnings, hints

Message list

Bottom area of interface window may show the following messages:

Icon	Meaning
	Error
	Warning
	Hint

Icon	Meaning
	Information
	Question
(none)	Nonrecognized text

Icons from left table are used by the compiler. An error interrupts compilation, warning indicates possibility of erroneous code (or another reason, e.g. outdated library). A hint may point out that global variable is hidden by local one with the same name.

Message format:

```
[icon] filename.cst@code_line message text
```

Context menu clears message list or removes some of its components.

Right table is reserved for future use in languages supported by .NET (e.g. C#).

Code line

A *.cst* file indicated in a message involves program code in ST language created by *Project > Build.* Double clicking the message opens POU editor with cursor at erroneous line. Sometimes however, the error may be somewhere else. If the

compiler is unable to find erroneous line, it indicates the line with number 0 or -1 (for instance, when task is not declared).

Omitting erroneous objects

The compiler operates similarly to a stack. So an error in a component of IF instruction in a function block generates three messages: 1) error in the component, 2) error in IF, 3) error in function block. In addition, if the option *Omit erroneous POU objects during compilation* has been selected, fourth message warns that the next object is being compiled without completing the previous one. In this next object, even for correct code, an error may be detected due to omitting the earlier code.

Autocomplete

Compilation of the project is a condition to display autocomplete list. It is convenient to compile the project after declaration of POUs to include datatype names, standard functions, etc. into the list. Second compilation should follow declaration of variables (clear message list before).

Library update

While opening an old project a warning may appear with information that library version of the project is different than the one being now used by CPDev. The library reference will be automatically updated if, while closing the project, you answer *Yes* to the question *Save changes in the project ...*

Compiler directives

Directives are optional commands for the compiler to simplify coding, determine access to variables, save comments, etc. Format is the same as for standard comments except additional sign $ after initial (*. Four most useful directives are described below.

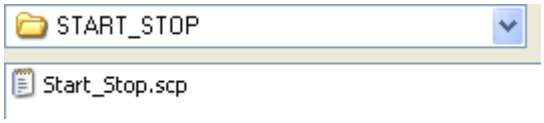
Directive	Meaning
(*$AUTO*)	Declaration VAR_EXTERNAL (*$AUTO*) END_VAR automatically inserts declarations from *Global variable list* into the program.
(*$READ*)	Variable declared in a program, as e.g. START: BOOL (*$READ*), is considered *read only* in this program. Other programs may write into it, however.
(*$WRITE*)	Variable declared in a program, as e.g. PUMP: BOOL (*$WRITE*), is considered *write only* in this program. Other programs may read it, however.
(*$VMASM*)	Part of a program written in Virtual Machine language.

Other directives govern internal operations of the compiler. Directives are highlighted by the editor.

Simulation session

All data for simulation, i.e. variable list, individual windows and control panels, can be saved in a file to repeat simulation session in future.

- *File > Save session* or click 
  Save as window involves default filename with *.scp* extension.



- Resuming the session
  File > Open session or click 

Session may be also resumed while opening *.dcp* file (provided that *.scp* is in the same folder). Answer *Yes* to the question *Do you want to open saved session as well?* One of CPSim *Program options* enables automatic resuming.

Save results

Simulation results may be saved in an *.out* file by selecting *Trace > Log output data.* Filename is determined in *Program options (Output file* tab with  and *Path)*. Symbol  in the status bar indicates logging. The *.out* file is a text file with variable values written in successive cycles. Variables from individual windows are logged only. Logging may be stopped by clicking the variable window with right button.

A part of *Start_Stop.out* file is shown below. START is set in 2nd and STOP in 11th second.

```
Time        START    STOP     ALARM    MOTOR    PUMP
200         0        0        0        0        0
400         0        0        0        0        0
...
2000        1        0        0        1        0
...
11000       1        1        0        0        1
...
16600       1        1        0        0        0
```

Time is given in milliseconds (200 ms task cycle). Columns are separated by *Tab*. The file can be processed by MS Excel.

Simulation controlled automatically

By selecting *Trace > Read input data* the simulator automatically sets values of variables from *.inp* file indicated in *Program options (Output file* tab*).* It is a text file (prepared earlier) of the same format as *.out*. Negative time terminates simulation.

```
Time          START         STOP          ALARM
0             1             0             0
10000         0             1             0
```

```
12000          0          1          0
20000          1          0          0
30000          1          0          1
35000          1          0          0
-40000
```

CPDev files

Programs and libraries of CPDev package exchange data through files with extensions given in the table. Name of *.xml* basic file is default name for the others.

Extension	Content
.xml	Basic file of the project
.cst	Program code in ST language (text file)
.hcp	Project header created during compilation
.dcp	Intermediate file for simulator and configurer created during compilation
.xcp	Binary code of compiled program for virtual machine VM (*runtime*)
.lcp	Semi–compiled library
.scp	Simulation session
.inp	Input data for session executed automatically (text file)
.out	Session results (text file), e.g. for MS Excel
.xmc	Communication parameters (for SMC controller)
.html	Project report
.htm	Communication report (for SMC: parameters, task table)

The *.cst* and *.xcp* files are created automatically during compilation. Recall that at the beginning it is convenient to create project folder for all files.

SOURCE CODES OF STANDARD BLOCKS

Implementations of IEC 61131–3 standard blocks are presented below, one for each of four groups. They may be of some help while learning ST programming using CPDev.

- SR flip–flop

```
FUNCTION_BLOCK SR
VAR_INPUT
    S1: BOOL;                          (* set input                *)
    R: BOOL;                           (* reset input              *)
END_VAR
VAR_OUTPUT
    Q1: BOOL;                          (* output                   *)
END_VAR
Q1 :=S1 OR (NOT R AND Q1);
END_FUNCTION_BLOCK
```

- R_TRIG rising edge detector

```
FUNCTION_BLOCK R_TRIG
VAR_INPUT
    CLK : BOOL;                    (* input                        *)
END_VAR
VAR_OUTPUT
    Q : BOOL;                      (* output                       *)
END_VAR
VAR
    CLKp : BOOL := FALSE;          (* previous value of CLK input  *)
END_VAR
Q := CLK AND NOT CLKp;
CLKp := CLK;
END_FUNCTION_BLOCK
```

- CTU up–counter

```
FUNCTION_BLOCK CTU
VAR_INPUT
    CU : BOOL;                     (* up-count input               *)
    R  : BOOL;                     (* counter reset                *)
    PV : INT;                      (* preset value – upper limit   *)
END_VAR
VAR_OUTPUT
    Q : BOOL;                      (* output set when limit reached *)
    CV : INT;                      (* current value                *)
END_VAR
VAR
    CUp : BOOL := FALSE;           (* previous value of CU input   *)
END_VAR
IF R THEN                          (* if R = TRUE                  *)
    CV := 0;
ELSE
    IF (CU AND NOT CUp) THEN       (* if rising edge at CU input   *)
        IF (CV < PV) THEN
            CV := CV + 1;          (* increment                    *)
        END_IF
    END_IF
END_IF
Q := CV >= PV;                     (* if CV >= PV, then Q := TRUE  *)
```

```
CUp := CU;                                  (* save CU as previous        *)
END_FUNCTION_BLOCK
```

- TP pulse timer (pulse of preset duration)

```
FUNCTION_BLOCK TP
VAR
    stime: TIME;                            (* start time                 *)
END_VAR
VAR_INPUT
    IN: BOOL;                               (* input                      *)
    PT: TIME;                               (* preset time                *)
END_VAR
VAR_OUTPUT
    Q: BOOL;                                (* output                     *)
    ET: TIME;                               (* elapsed time               *)
END_VAR
IF NOT Q THEN                               (* state 0 or 2:              *)
    IF IN THEN             (* if rising edge at IN or waiting for IN=0  *)
        IF ET = t#0s THEN                  (*if rising edge at IN          *)
        IF PT > t#0s THEN                  (* state 1: pulse time count    *)
                stime := CUR_TIME();       (* save start time              *)
                Q := TRUE;                 (* set the output Q             *)
            END_IF
        ELSE                               (* state 2: wait for IN=0       *)
            Q := FALSE;                    (* reset Q                      *)
        END_IF
    ELSE                          (* state 0: wait for rising edge at IN  *)
        ET := t#0s;                (* reset elapsed time                   *)
    END_IF
ELSE                                       (* state 1: pulse time count    *)
   ET := CUR_TIME() - stime;               (* elapsed time update          *)
    IF ET >= PT THEN                       (* if preset value reached      *)
        Q := FALSE;                        (* reset Q                      *)
        ET := PT;                          (* elapsed := preset            *)
    END_IF
END_IF
END_FUNCTION_BLOCK
```