ON-LINE PROGRAM CHANGE

ON-LINE PROGRAM CHANGE

2nd edition- March 2008

CONTENTS

1. Introduction

On-line program change is a feature of the central unit of the PLC Tecomat that enables to undertake modifications of the user program without stopping the technology operation, i. e. without the necessity to shut the operated technology during the PLC program changes. This feature allows the Tecomat system programmer to undertake modifications of the PLC program while the program is running. The responsibility for correctnes of changes undertaken in on the system programmer. The PLC central unit in cooperation with the Mosaic environment ensure the safe changes modification in one moment so, that the smoothnes of operation is not endangered.

On-line program change can be undertaken in the TC700 system with central units CP-7001 and CP-7002 from version SW v4.1. The support of on-line program change is integrated in the Mosaic programming environment from version v1.5.10. The behaviour during the on-line change can be also tested with the PLC simulator in Mosaic environment.

1.1 Basic principles of operation

For explanation of the basic principle, the following example will be used. Lets suppose that PLC Tecomat control the technology which shut-down means considerable economic loss, e. g.: burning kiln and the programmer´s task is to modify the PLC program. In this situation is quite indifferent if it will be a change of the wrong control algorithm or addition of a new function, e. g. for burning of another product line. The program for PLC needs to be changes and the oven must not be stopped. On-line program change offers the solution. The programmer accomplish the relevant PLC program changes and central unit of the PLC ensures the swith between the old program so, that the n cycle of the calculation is undertaken completely and the following cycle is undertaken according to the new program. The central unit, at the same time, ensures the neccessary operations connected with the changes done so, that the smoothness of control is not disturbed.

1.2 On-line changes options

In terms of on-line change, the PLC Tecomat system programmer is able to modify following program parts:

- Program code, i. e. optionable changes of all program parts (functions, function blocks, programs) including the insertion of new POU, or rather their omitting.
- POU interface modifications, i. e. changes of input and output POU variables including their adding or omitting
- Variable modification, i.e. insertion and omitting of all variable types (local and global) or change of variables as a e.g. change of the field size.
- Data types modifications, e. g. changes in structures, adding of new data types and omitting of obsolete data types.
- Remanent zone size modifications

Following modifications can not be within the on-line program changes undertaken:

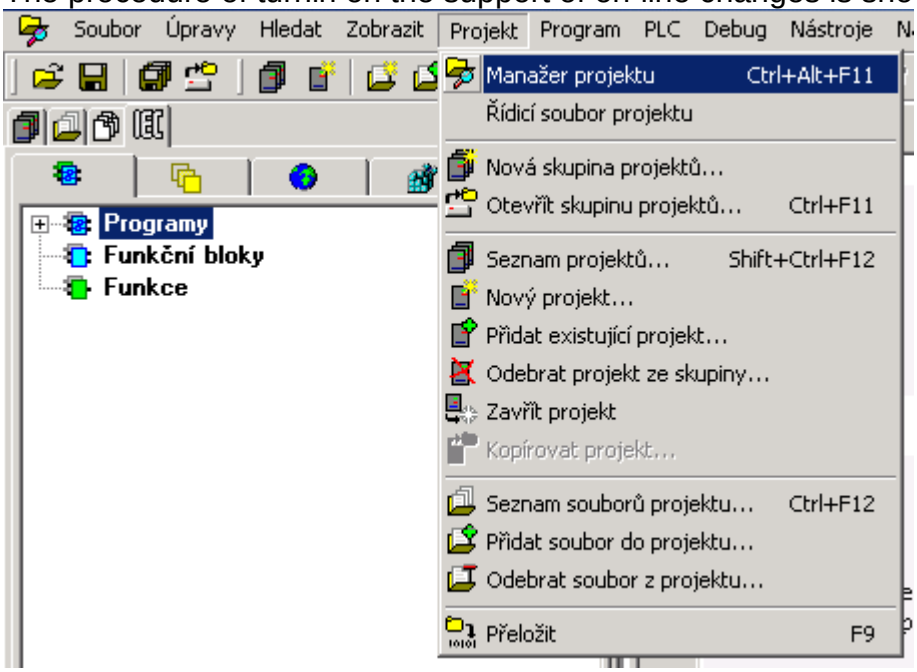
- system hw configuration changes, e.g.. adding of IO modules or change of the IO module type
- changes of IO modules settings
- changes in communication parameter settings for serial channels
- changes in the PLC network

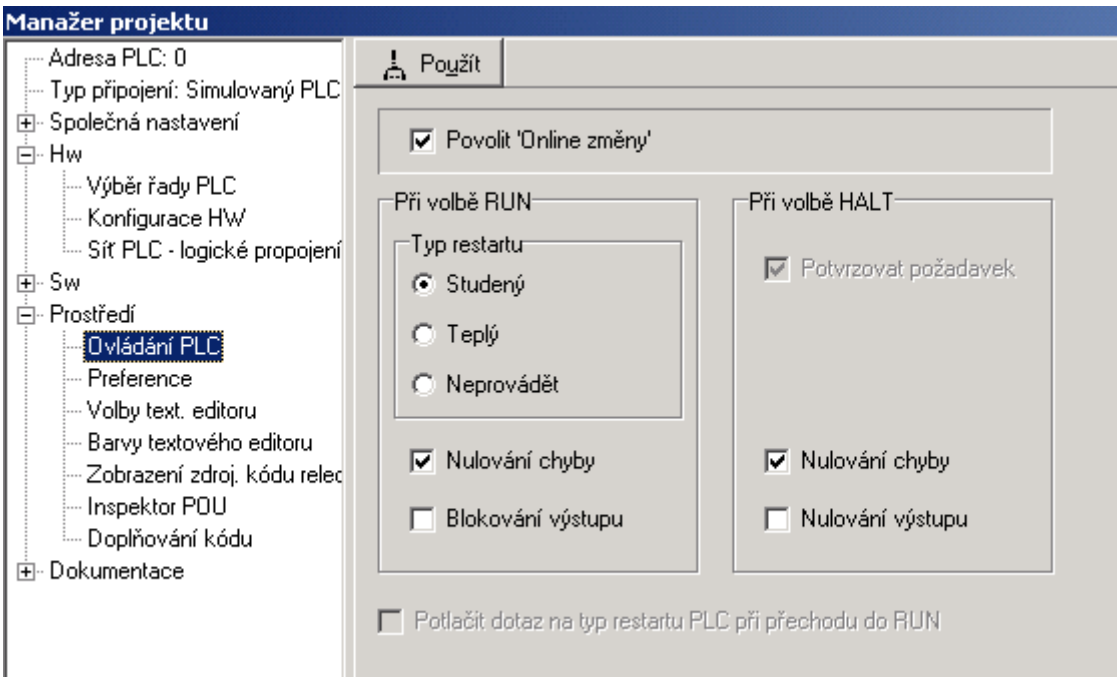
1.3 Turn on of on-line changes support within Mosaic

On-line program changes can be turn on using the following procedure:

- choose in the menu Project | Project manager (CTRL+ALT+F11)
- in the project tree choose the nod Environment | PLC control
- choose the item „Enable on-line changes“

The procedure of turnin on the support of on-line changes is shown in pictures bellow:



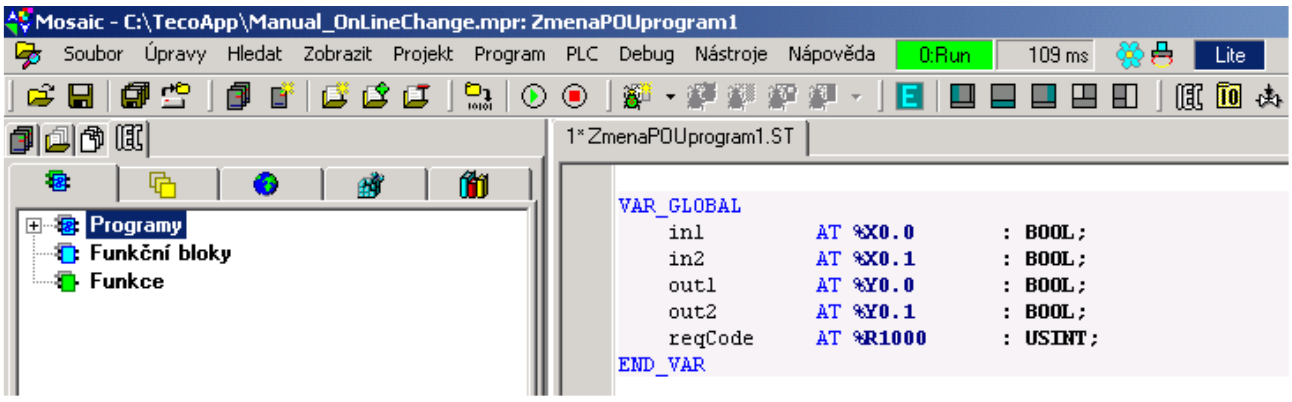


If the PLC central unit does not support on-line changes, the item enabling on-line changes will be grey and the thicking box will not be active.

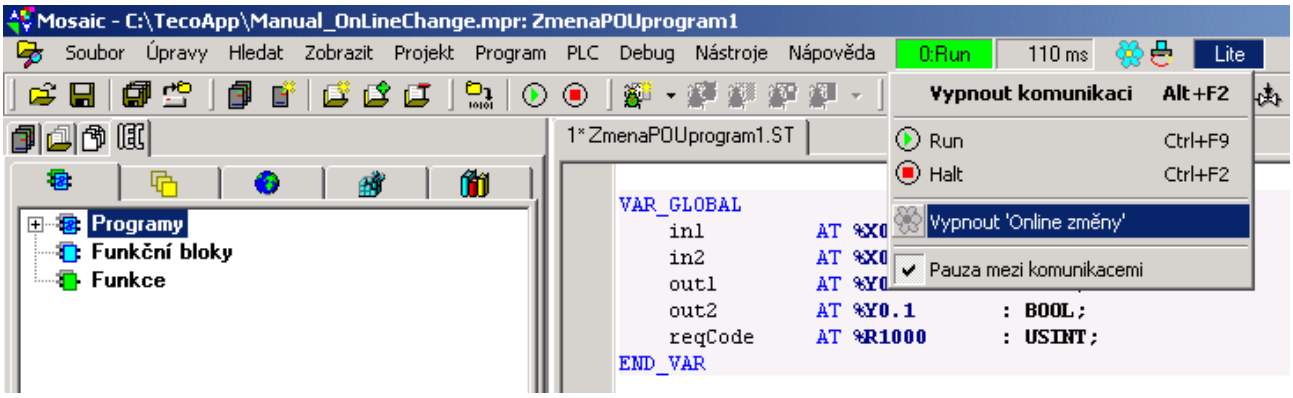
If on-line changes are swithed on, also the option „Confirm the request within the HALT option“ will be turn on automatically, which means that the switch-over of PLC to the HALT mode must be confirmed in a special dialogue and, therefore, the control can not be stopped by pressing the icon HALT only or using the keyboard shortcut CTRL+F2. Furthermore, it is not possible to suppress the query on the restart type after the first instalation of the new program to the PLC. The dialogue with the inquiry on restart type is displayed whenever the new program is loaded to the PLC for the first time (i. e. there is no program in the PLC towards which the changes could be evaluated).

1.4 Indication of on-line changes support within Mosaic

The switched support of on-line changes is in the Mosaic environment indicated on the bar Menu by an icon with the symbol of a flower  . If the icon is coloured, the support of on-line changes is switched on. If the icon is grey, on-line changes are switched off and each program change will lead to control stopping while the new program is loaded to the PLC.



On-line changes can be also switched on or off from the menu which appears after pressing the left mouse button above the icon of the on-line changes or above the icon with the symbol of the connected PLC as shown on the following picture.



Similarly to the previous description the on-line change can be switched on only in case when the central unit of the PLC support these changes. In other cases the option is not active (grey) and can not be selected.

2. On-line program changes in ST language

In this chapter, we will introduce on-line changes in the program that is written in ST language.

2.1 Opening of PLC programming while on-line changes are on

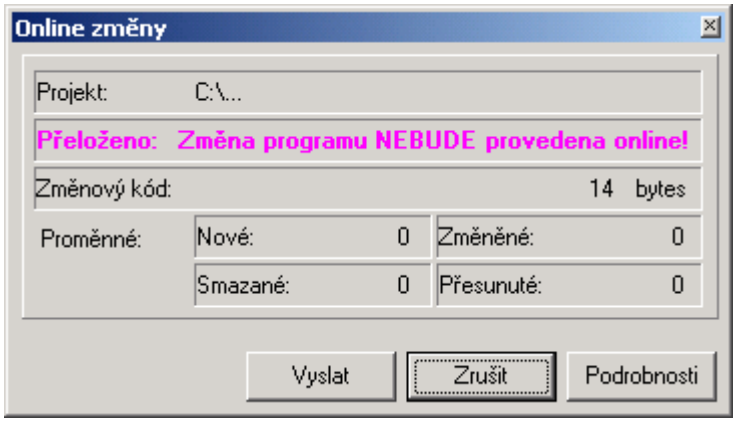
After we turn on the support for on-line program changes, it is neccessary to write the first program version, succesfully compile this program (e.g.: F9) and load it to the PLC central unit (e.g.: CTRL+F9). Before loading of the program to the PLC, the Mosaic environment check whether there exists, in the PLC, any previous version of the program loaded. If not (e.g.:it is a brand new program and new central unit), then the program is loaded to the PLC in a standard way. It means that the central unit is set to HALT mode, PLC outputs are blocked, program is loaded to the PLC and the central unit transfers to the RUN mode with preset reset type. Afterwards, the PLC central unit along with the programming Mosaic environment is ready to accept changes within the program without control cessation.

For further description, we will use the following example. Lets suppose that the program in the PLC switch on or off the cooling according to the central unit temperature data. Variable *temp_CP7002* shows temperature in degree Celsius (system registry S36). Output *cool* is switched on when the temperature overflow 50 degrees.

```
VAR_GLOBAL
  temp_CP7002    AT %S36      : SINT;   // CPU temperature
  cool           AT %Y0.0     : BOOL;   // cooling
```

```
END_VAR

PROGRAM Prog1

  CASE temp_CP7002 OF
    0..50   : cool := false;
    51..127 : cool := true;
  END_CASE;
END_PROGRAM

CONFIGURATION ExampleOnLineChange
  RESOURCE CPM
    TASK FreeWheeling(Number := 0);
    PROGRAM main WITH FreeWheeling : Prog1 ();
  END_RESOURCE
END_CONFIGURATION
```

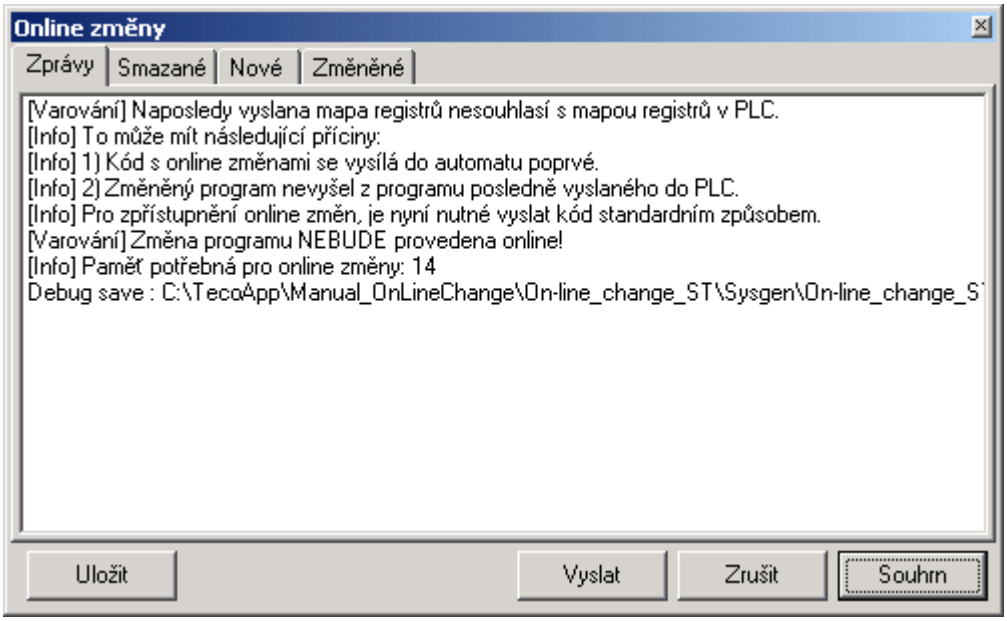
We compile the program (F9) and send it to the central unit (CTRL+F9). Before the program code is sent, the dialogue with information on on-line changes will appear.



If we choose „Send“ the program code will be sent in a standard way because it is a new program. This means that the central unit transfer to the HALT mode (where outputs are blocked standardly), the new program will load, restart is undertaken and then the central unit transfer to the RUN mode.

If we choose „Cancel“ the program code will not be sent and the central unit remains without changes with the original program and in the original mode.

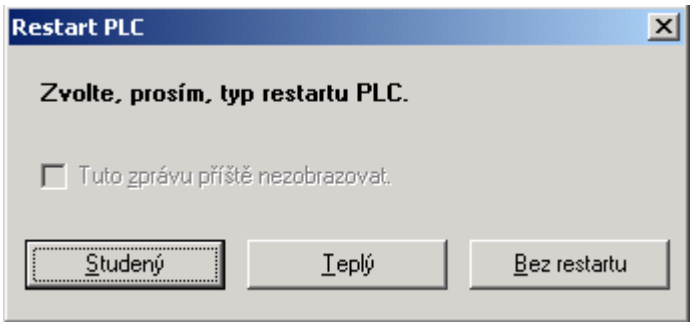
Option „Details“ enables to display additional information.



Options „Send“ and „Cancel“ have the same meaning as in the previous dialogue. Option „Summary“ undertakes the return to the previous dialogue. Option „Save“ saves all information about on-line change into the text file.

Bookmark „Messages“ contains summary information about on-line chage. Bookmarks „Deleted“, „New“ and „Changed“ contain information about variable changes and in our case are empty because we start the on-line change (we load new program).

We send a new code to the PLC using the option „Send“. The dialogue with the restart option will appear afterwards.



After the restart option is selected, the central unit transfer to the RUN mode. From this moment it is possible to edit the program without control cessation (thus on-line).

2.2 Changes in program code

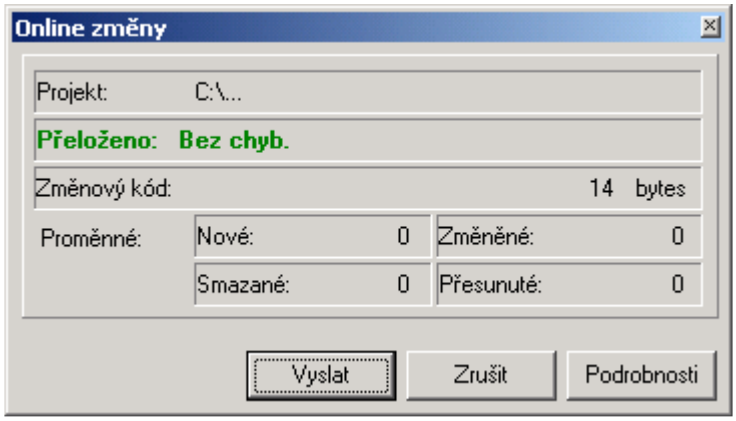
Lets suppose that the previous program is neccessary to be updated and heating control needs to be added. If the tepmerature measured by the central unit is bellow 0 degrees, we will switch the output for heating control.

The first thing that we must realize is the status of the editor windows within the Mosaic environment. After the new program is loaded and the RUN mode is on, all editor windows are in the DEBUG status which means that it is not possible to edit the text. This status is indicated by the icon  in the left bottom corner of the editor window. The switch-over to the EDIT status can be done by clicking the left mouse button on this icon or by using the hot-key ALT+F6. EDIT status is indicated by the icon  .

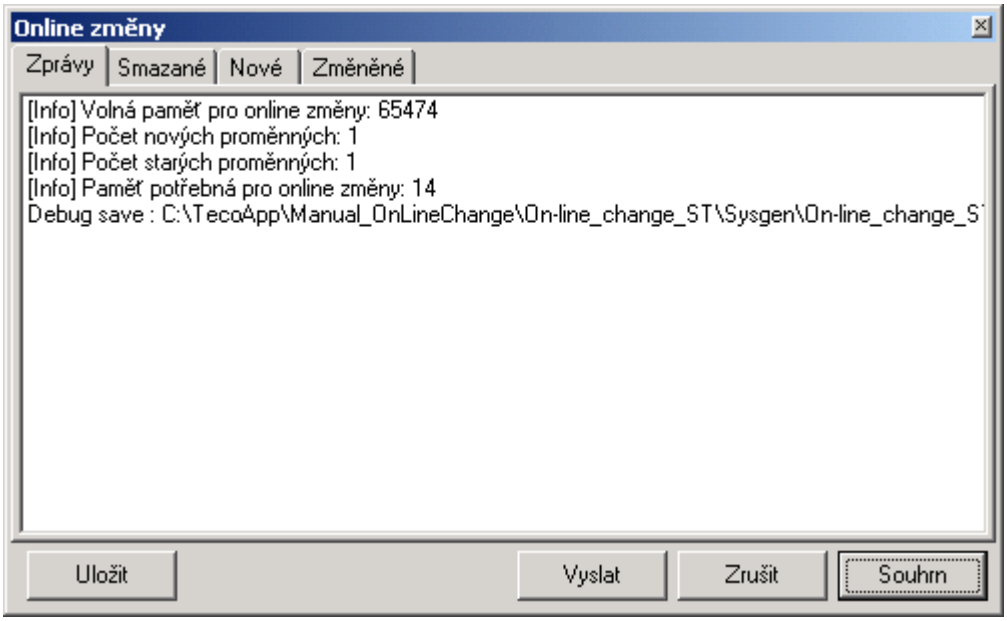
Now we can modify the program. We add the definition for heating output *heat* and add its control to the program. The added parts have turquiouse background.

```
VAR_GLOBAL
  temp_CP7002    AT %S36      : SINT;   // CPU temperature
  cool           AT %Y0.0     : BOOL;   // cooling
  heat           AT %Y0.1     : BOOL;   // heating
END_VAR

PROGRAM Prog1

  CASE temp_CP7002 OF
    0..50   : heat := false; cool := false;
    51..127 : heat := false; cool := true;
    ELSE
              heat := true;  cool := false;
  END_CASE;
END_PROGRAM
```

Further on the procedure is standard: compile the modified program and send the code to the PLC. And because we have the on-line changes switched on and in the central unit exists previous version of our program, the new program will be accepted by the central unit without control stopping. Before the program is sent to the PLC, the dialogue with information about on-line changes undertaken will appear. This time it will looks as follows.



The dialogue informs us that changes were undertaken in the program code only. This fact can be verified in details.



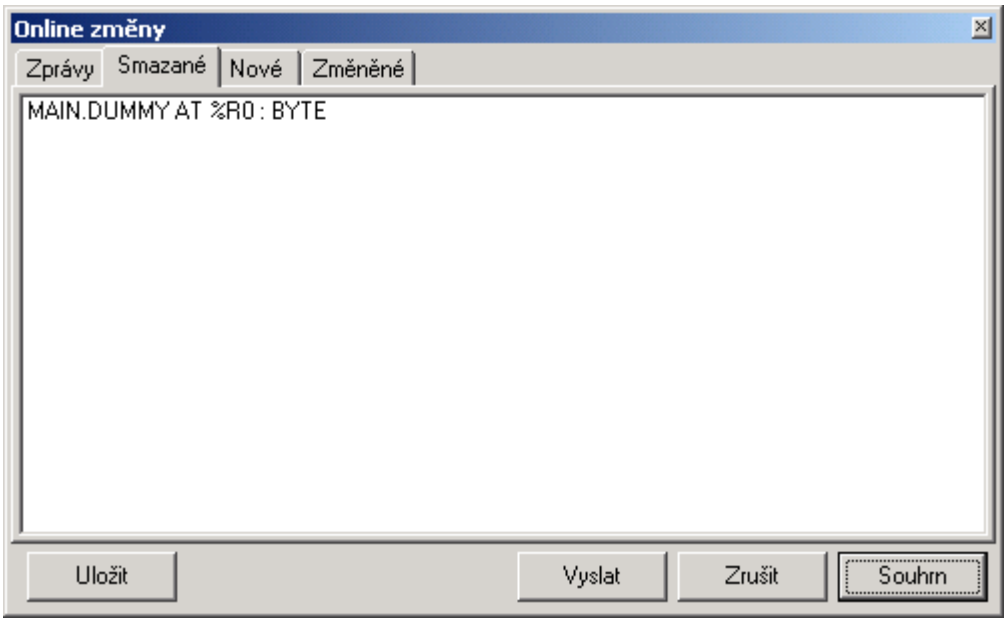
2.3 Changes in program variables

For illustration of this feature, we will complete the previously used program with the variable which will file the number of cases when the temperature of the central unit exceeded the ranges <0,50>. Consequently, the new variable with number of error states *errCounter* of *USINT* type, is added among local variables. The name of the added variable is *tm,* it is an instance of the function block *R_TRIG* and is used for evaluation of the change of signals *heat* and *cool*. Added parts have again the turquoise background.

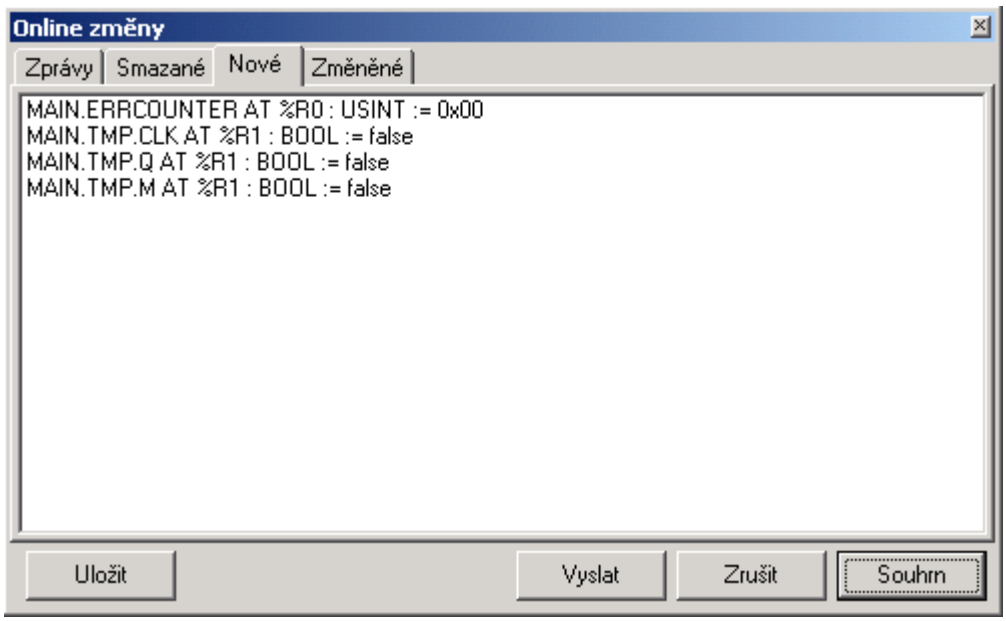
```
VAR_GLOBAL
  temp_CP7002    AT %S36       : SINT;    // CPU temperature
  cool           AT %Y0.0      : BOOL;    // cooling
  heat           AT %Y0.1      : BOOL;    // heating
END_VAR

PROGRAM Prog1
  VAR
    errCounter  : USINT;
    tmp         : R_TRIG;
  END_VAR

  CASE temp_CP7002 OF
    0..50   : heat := false; cool := false;
    51..127 : heat := false; cool := true;
    ELSE
              heat := true;  cool := false;
  END_CASE;

  tmp( CLK := heat OR cool);
  IF tmp.Q THEN  errCounter := errCounter + 1;  END_IF;
END_PROGRAM
```

Before the compiled code is sent, the dialogue with the summary of changes undertaken will appear.

From this dialogue, it is apparent that this change added to the program the total of 4 new variables and one variable from the original program was deleted. This can, on the first sight, seem illogical because we did not delete any variable from the program and we added only two new variables *errCounter* and *tmp*. The explanation can be found in details.



If the POU program has not any local variable declared, the compiler in the Mosaic environment set up automatically at least one empty variable *DUMMY*. In the moment when we add to the program at least one own variable, the variable *DUMMY* loses its purpose and the compiler deletes it. This explains the question of the deleted variable. The variance in the number of new variables is also explained in the details window.



From this window, it is apparent that to the variable *errCounter* corresponds really one new variable while under the variable *tmp* there are hidden three variables really because it is an instance of the funcion block of the type *R_TRIG*. Each instance of this function block will then contain variables *CLK*, *Q* and *M* of BOOL type.

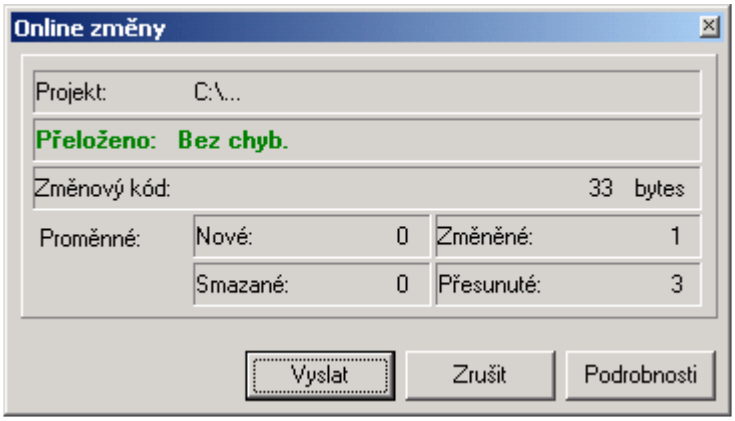
Moreover, it is shown in this window which initialization value was filled in new variables after their establishment. Generally applies that new variables are filled with the initialization value that the programmer states in the variable declaration. If the initialization is not set in the variable declaration then the variable is filled with the implicit initialization value for the correspondent data type.

Now imagine the following situation. Variable *errCounter* is of USINT type and maximum value of this variable can be 255. Lets suppose that it is too little for this case. Therfore, we will change the data type of the variable *errCounter* from USINT type to UDINT type. For better oriantation, there is, in this picture, the changed line violet. The rest of the program remained unchanged.

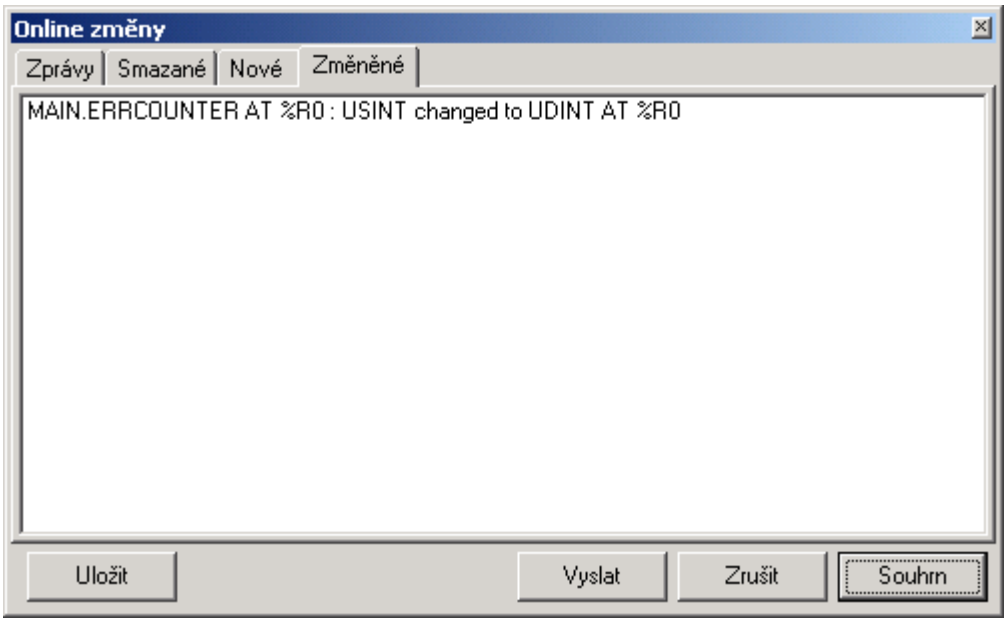
```
VAR_GLOBAL
  temp_CP7002    AT %S36      : SINT;   // CPU temperature
  cool           AT %Y0.0     : BOOL;   // cooling
  heat           AT %Y0.1     : BOOL;   // heating
END_VAR

PROGRAM Prog1
  VAR
    errCounter  : UDINT;
    tmp         : R_TRIG;
  END_VAR

  CASE temp_CP7002 OF
    0..50   : heat := false; cool := false;
    51..127 : heat := false; cool := true;
    ELSE
              heat := true;  cool := false;
  END_CASE;

  tmp( CLK := heat OR cool);
  IF tmp.Q THEN  errCounter := errCounter + 1;  END_IF;
END_PROGRAM
```

Further, we will assume that before the change the variable *errCounter* has a non-zero value. The aim of the on-line chage is, of course, to save all actual variable values in the program in both cases. Firstly, in case of change of variable data type, and, secondly, in case of change of variable location in the PLC memory. Both cases occured during this change. *errCounter* changed its data type and variable *tmp* changed its location within the memory.

The overall information displays,again, the dialogue Changes summary.



No variable was added or deleted, however, all four were changed. Changes of data types are described in details, changes in variable location within the memory are not displayed.



Described changes are managed automatically by central unit of the PLC in cooperation with the Mosaic environment after the program code is sent from Mosaic to PLC. Variables simply keep values independently of changes of their data type or memory location.

We can act upon the same manner as for local variables changes(VAR ... END_VAR), also for global variables (VAR_GLOBAL ... END_VAR resp. VAR_GLOBAL RETAIN ... END_VAR). Variables, consequently, can be added, deleted and data type of variables can be changed.

2.4 Changes of local variables to global variables

The example of this type of change can be the requirement on change of the local variable *errCounter*, from our previous example, to the global variable, e. g. because of the need of back up of this variable value during power failure. It seems that only a change of the original declaration of the local variable

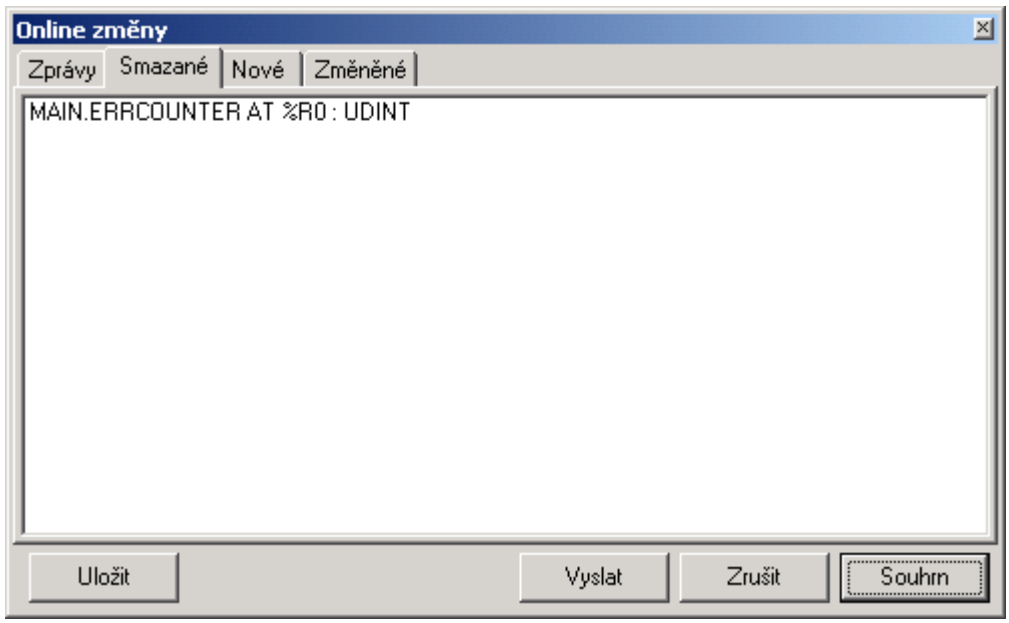
```
PROGRAM Prog1
  VAR
    errCounter  : UDINT; ;   // local variable
    tmp         : R_TRIG;
  END_VAR
```

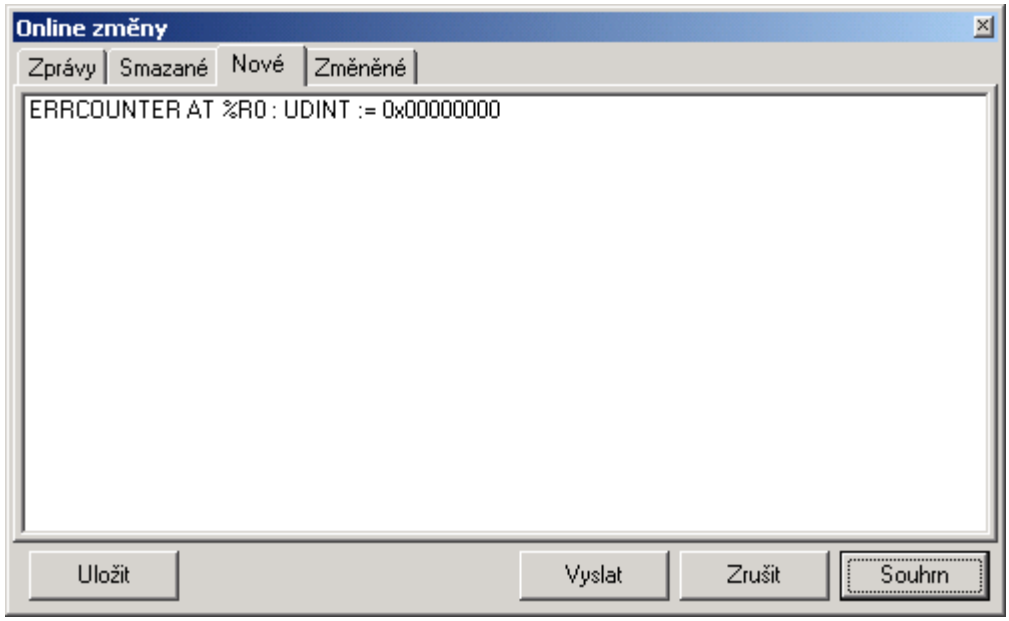
to new declaration of the global variable will be sufficient.

```
VAR_GLOBAL RETAIN
  errCounter  : UDINT;     // global variable
END_VAR

PROGRAM Prog1
  VAR
    tmp         : R_TRIG;
  END_VAR
```

In this case, unfortunately, appearances are deceptive. The warning shall be the summary dialogue displaying the undertaken changes.

Here, we can se, that even if we did not change the number of variables nor their names, one variable would be deleted from the program (omitted) and one new variable would be added. The reason will be obvious from the detailed information about on-line changes.





The local variable *errCounter* and global variable *errCounter* are not really called similarly. Because the full name of any local variable consists of the instance name and variable name. Thus, in our case the local variable *errCounter* is accualy called *main.errCounter* where main is the name of the program instance. And because names are not identical, they are treated during the on-line change as two different variables. Thus, the local variable *errCounter* will be omitted and new global variable will be created. The consequence will be the loss of the *errCounter* value because the newly created global variable is initialized to the 0 value.

So how to proceed during the change of the local variable to the global variable? This situation is neccessary to be treated in two steps. In the first stem we only add new global variable *retainErrCounter*. Then change the local variable *errCounter* to output variable and add its value into the newly created global variable. The modified program will appear as follows.

```
VAR_GLOBAL
  temp_CP7002    AT %S36      : SINT;   // CPU temperature
  cool           AT %Y0.0     : BOOL;   // cooling
  heat           AT %Y0.1     : BOOL;   // heating
END_VAR

VAR_GLOBAL RETAIN
  retainErrCounter    : UDINT;
END_VAR

PROGRAM Prog1
  VAR
    tmp         : R_TRIG;
  END_VAR
  VAR_OUTPUT
    errCounter  : UDINT;
  END_VAR

  CASE temp_CP7002 OF
    0..50   : heat := false; cool := false;
    51..127 : heat := false; cool := true;
    ELSE
              heat := true;  cool := false;
  END_CASE;

  tmp( CLK := heat OR cool);
  IF tmp.Q THEN  errCounter := errCounter + 1;  END_IF;
END_PROGRAM

CONFIGURATION ExampleOnLineChange
  RESOURCE CPM
    TASK FreeWheeling(Number := 0);
    PROGRAM main WITH FreeWheeling : Prog1 (errCounter => retainErrCounter);
  END_RESOURCE
END_CONFIGURATION
```

When the on-line change is undertaken, the output variable *errCounter* will keep the original value of the local variable *errCounter* becase, as we already know, their real names are the same, i. e. *main.errCounter*. The value of the output variable *errCounter* is copied in each cycle into the new global variable *retainErrCounter*.

The second step is omittance of the output variable *errCounter* and program change where we must replace the processing of the output variabe *errCounter* for the processing of the global variable *retainErrCounter*. The assignment of the output variable *errCounter* must be also done because it does not exist anymore. The program will appear as follows.

```
VAR_GLOBAL
  temp_CP7002    AT %S36      : SINT;   // CPU temperature
  cool           AT %Y0.0     : BOOL;   // cooling
  heat           AT %Y0.1     : BOOL;   // heating
END_VAR

VAR_GLOBAL RETAIN
  retainErrCounter    : UDINT;
END_VAR

PROGRAM Prog1
  VAR
```

```
    tmp         : R_TRIG;
  END_VAR

  CASE temp_CP7002 OF
    0..50   : heat := false; cool := false;
    51..127 : heat := false; cool := true;
    ELSE
              heat := true;  cool := false;
  END_CASE;

  tmp( CLK := heat OR cool);
  IF tmp.Q THEN  retainErrCounter := retainErrCounter + 1;  END_IF;
END_PROGRAM


CONFIGURATION ExampleOnLineChange
  RESOURCE CPM
    TASK FreeWheeling(Number := 0);
    PROGRAM main WITH FreeWheeling : Prog1 ();
  END_RESOURCE
END_CONFIGURATION
```

Renumeration for this complicated process is correctly undertaken on-line program change.

2.5 Risks during on-line changes in ST language

2.5.1 Variable renaming

While working with variables, it is neccessary to bear in mind that the compiler recognizes on-line changes of variables according to their names and data types. This applies also for repair of the variable name. The compiler does not evaluate grammar correctness of the name, it only compare the similarity of variable names between old and new program.

```
VAR_GLOBAL
  temp_CP7002    : SINT;   // CPU temperature
END_VAR
```

3. On-line changes of *.mos program

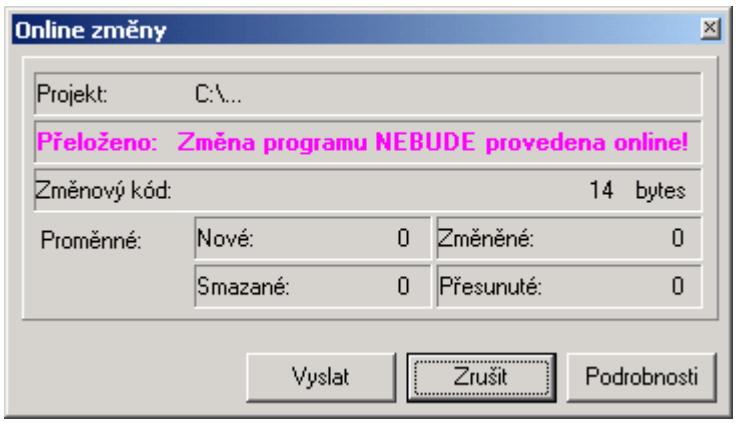
On-line changes can be also used for programs that are written in traditional instructions for PLC Tecomat. The following chapter is devoted to these changes.

3.1 On-line program changes in assembler

For further explanation we will use the same example as for the description of changes in ST language. Lets suppose that the program in the PLC switch on and off the cooling according to the central unit temperature data. Variable *temp_CP7002* displays temperature in Celsius degrees (system registry S36). Output *cool* is turned on when the temperature overflow 50 degrees. The program will appear as follows.

```
#def    temp_CP7002    %S36     ; CPU temperature
#def    cool           %Y0.0    ; cooling
;
P 0
        LD      temp_CP7002
        EXTB
        GTS     -1
        LD      temp_CP7002
        EXTB
        LTS     51
        OR
        NEG
        RES     cool
        ;
        LD      temp_CP7002
        EXTB
        GTS     50
        SET     cool
E 0
```

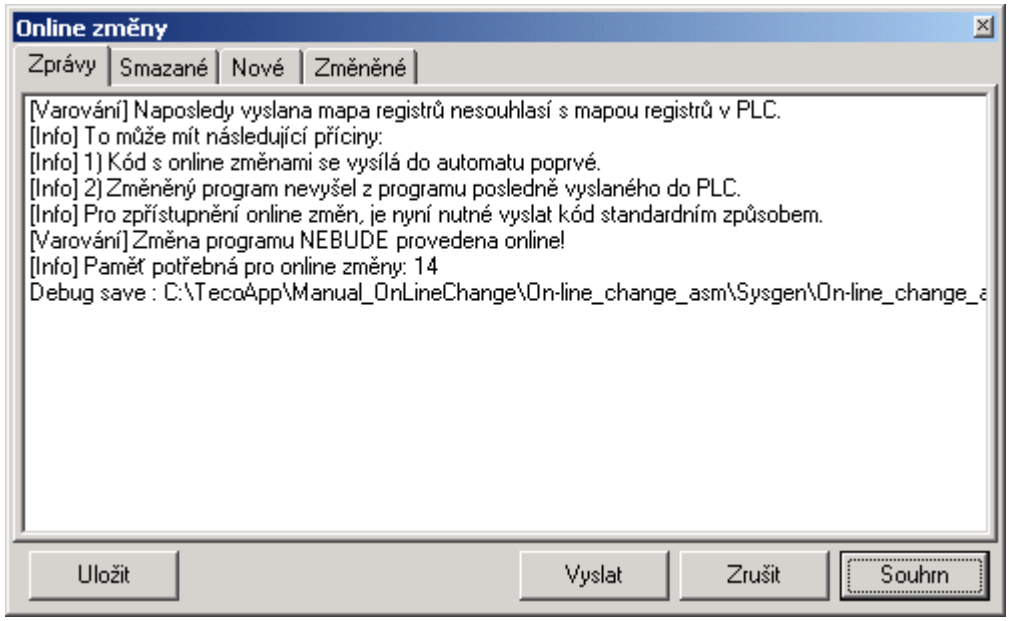
We switch on on-line changes, compile the program (F9) and send it to the central unit (CTRL+F9). Before the sending of the program code take place, the dialogue with information about on-line changes will appear.



If we choose „Send“, the program code will be send in a standard way because it is a new program. It means that the central unit will transfer to the HALT mode (where outputs are typically blocked), new program will be loaded, restart will be done and then the central unit will transfer to the RUN mode.

If we choose „Cancel“ the program code will not be send and the central unit will remain without a change with the original program and in the original mode.

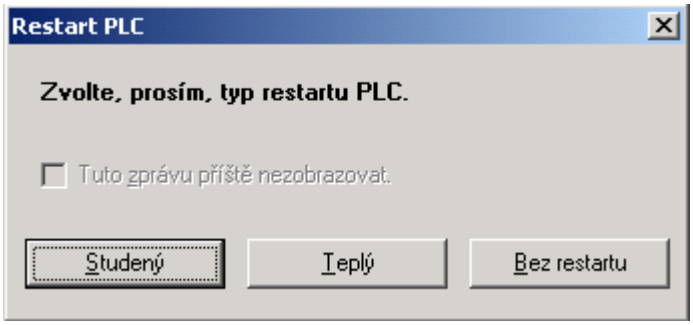
Option „Details“ enables to display additional information.



Options „Send“ and „Cancel“ have the same meaning as in the previous dialogue. Option „Summary“ accomplish the return to the previous dialogue. Option „Save“ saves all information about on-line chage into the text file.

Bookmark „Messages“ contains summary information about on-line change. Bookmarks „Deleted“, „New“ and „Changed“ carry information about changes of variables and, in our case, are empty because we initialize the on-line change (we load new program).

Therefore, we will send a new code to the PLC using the option „Send“. Afterwards, the dialogue with the restart option will appear.



After the selection of the restart type, the central unit transfers to the RUN mode. From this moment it is possible to modify program without control stopping (thus on-line).

3.2 Changes in program code

Lets suppose that the previous program needs the heating control to be added. If the temperature measured by the central unit is bellow 0 degrees, the output for heating control is switched on.

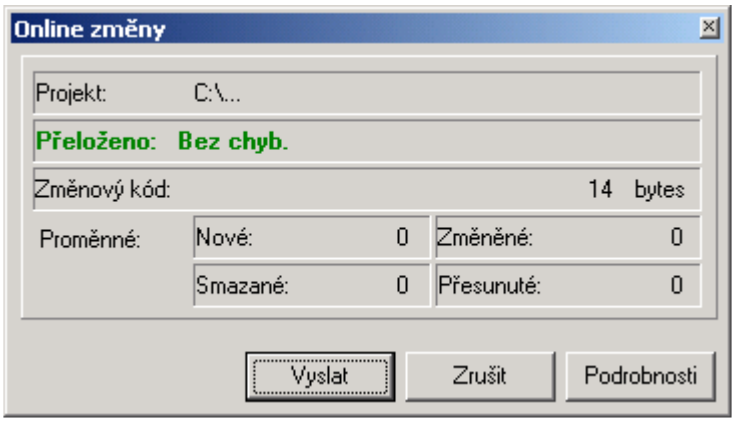
The first thing that we need to realize is the status of editor windows in the Mosaic environment. After the new program is loaded and RUN mode is active, all windows of the editor are in the DEBUG status which means that it is not possible to edit text. This status is indicated by the icon  in the left bottom corner of the editor window. The switch-over to the EDIT status can be done by clicking the left-mouse button on this icon or by a hot key ALT+F6. EDIT status is indicated by the icon  .

Now we can modify the program. We add the definition for heating output *heat* and add its control to the program. Added parts have turquoise background.

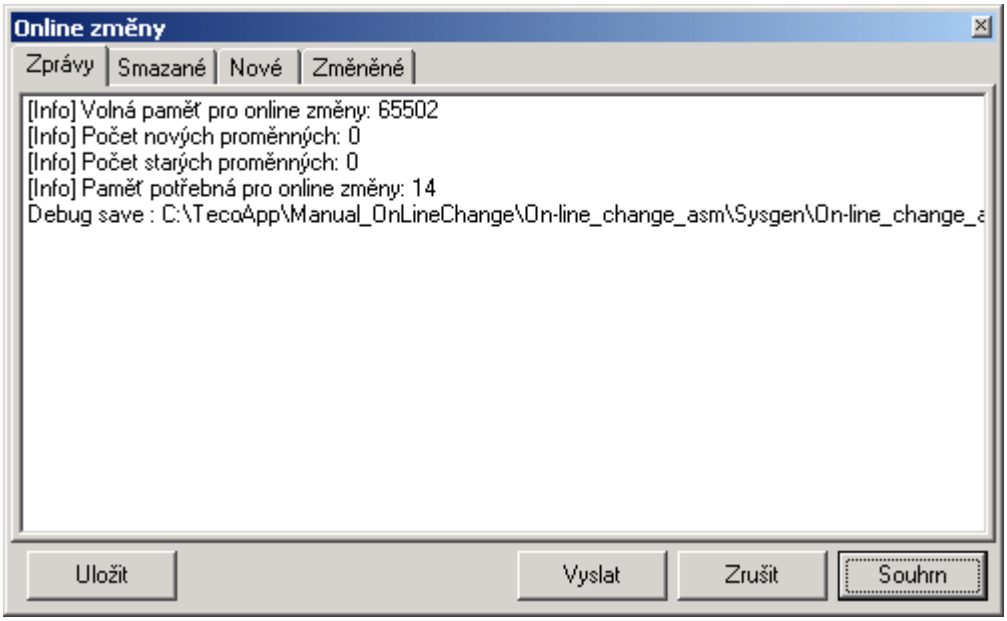
```
#def    temp_CP7002    %S36     ; CPU temperature
#def    cool           %Y0.0    ; cooling
#def    heat           %Y0.1    ; heating
;
P 0
        LD      temp_CP7002
        EXTB
        GTS     -1
        LD      temp_CP7002
        EXTB
        LTS     51
        OR
        NEG
        RES     cool
        RES     heat
        ;
        LD      temp_CP7002
        EXTB
        GTS     50
        SET     cool
        RES     heat
```

```
        ;
        LD      temp_CP7002
        EXTB
        LTS     0
        RES     cool
        SET     heat
E 0
```

Further, the procedure is standard: compile the modified program and send the code to the PLC. And because the on-line changes are on and in the central unit exists the previous version of our program then the new program will be accepted by the central unit without control cessation. Before the program is sent to the PLC, the dialogue is displayed again with information about on-line changes undetaken. This time it will be as follows.



The dialogue informs us that changes were done within the program code only. This can be verified in details.



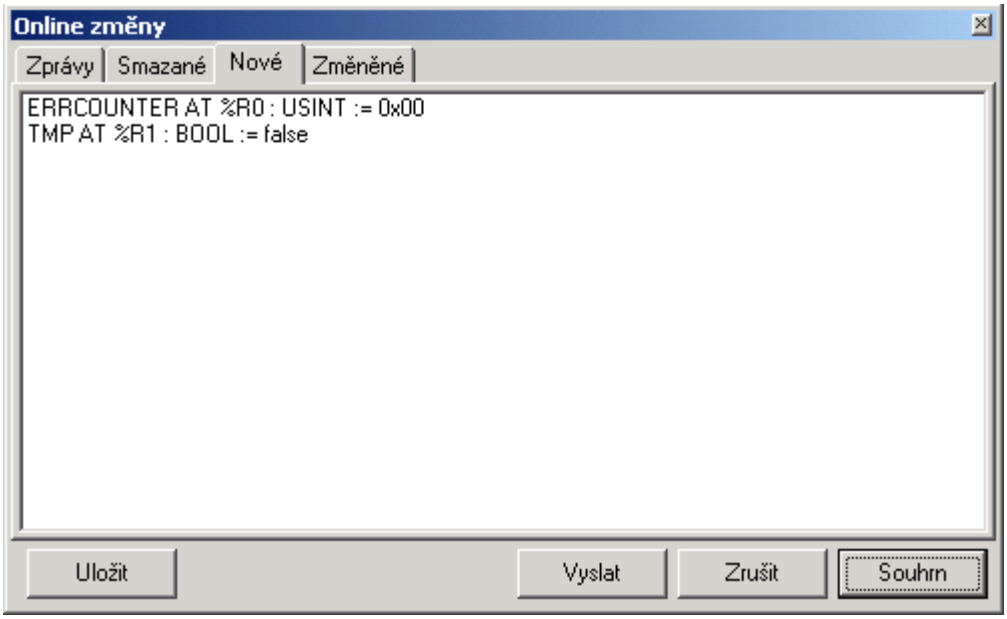
3.3 Changes in program variables

For illustration of this feature, we will complete the previously used program with the variable which will file the number of cases when the temperature of the central unit exceeded the ranges <0,50>. Consequently, the new variable with number of error states *errCounter* of *USINT* type, is added among local variables. The name of the added variable is *tmp,* and it is a subsidiary variable for LET instruction which is used for evaluation of.the change of signals *heat* and *cool*. Added parts have again the turquoise background.

```
#def    temp_CP7002   %S36     ; CPU temperature
#def    cool          %Y0.0    ; cooling
#def    heat          %Y0.1    ; heating
;
#reg    USINT         errCounter
#reg    BOOL          tmp
;
P 0
        LD      temp_CP7002
        EXTB
        GTS     -1
        LD      temp_CP7002
        EXTB
        LTS     51
        OR
        NEG
        RES     cool
        RES     heat
        ;
        LD      temp_CP7002
        EXTB
        GTS     50
        SET     cool
        RES     heat
        ;
        LD      temp_CP7002
        EXTB
        LTS     0
        RES     cool
        SET     heat
        ;
        LD      heat
        OR      cool
        LET     tmp
        AND     1
        LD      errCounter
        ADD
        WR      errCounter
E 0
```

Before the compiled code is send, the dialogue with the summary of changes undertaken will appear.

In this dialogue it is apparent that this change added to the program the total of two new variables. These are our variables *errCounter* and *tmp*. We can find out information on this in details.



This window states which variables were newly created and which initialization value were added to these variables after their creation. Variables created according to the dire-citve *#reg* have initialization value 0.

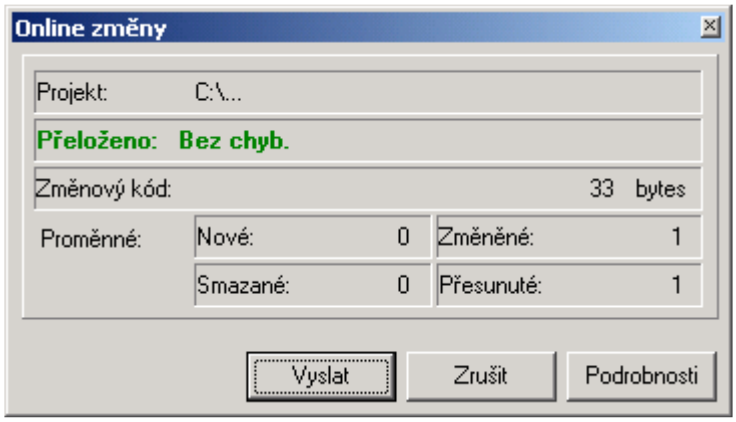
Now imagine following situation. Variable *errCounter* is of an USINT type and maximum value of this variable can be 255. Lets assume that it is too little in this case. Therefore, we will change the data type of the variable *errCounter* from USINT type (variable of the size of 1 byte, without a sign) to UDINT type (variable of the size of 4 bytes, without a sign). The changed line in this text is violet for better orientation. The rest of the program is un-changed.

```
#def    temp_CP7002   %S36     ; CPU temperature
#def    cool          %Y0.0    ; cooling
#def    heat          %Y0.1    ; heating
;
#reg    UDINT         errCounter
#reg    BOOL          tmp
;
```

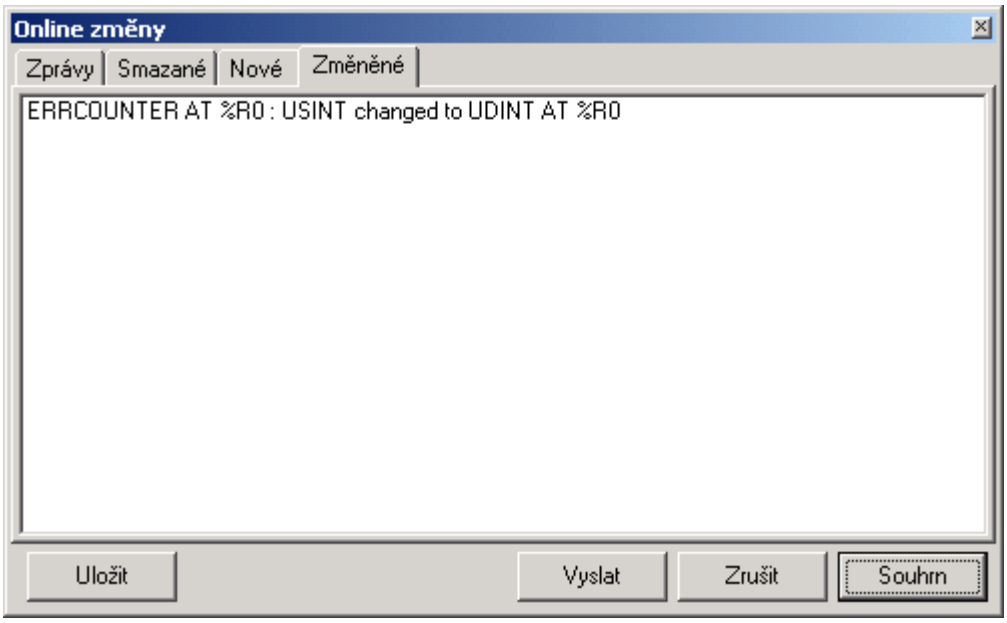
```
P 0
        LD      temp_CP7002
        ...
```

Further, we will suppose, that before the change the variable *errCounter* has a non-zero value. The aim of the on-line change is, of course, to retain all actual values of program variables both in case of data type change or in case of change of variable location within the PLC memory. Both cases occured during this change. Variable *errCounter* changed its data type and variable *tmp* changed its location within the memory.

The overall information is shown, again, in the changes summary dialogue.



No variable was added or deleted, however, all were changed. In details, the data type changes are described, variable location changes are not described, only their number is edited.



Described changes are undertaken automatically by the PLC central unit in cooperation with the Mosaic environment after the program code is sent from Mosaic to the PLC. Variable values remain the same independently of whether their data type or memory location was changed.

3.4 Risks during on-line changes in assembler

3.4.1 Directive #def

The compiler „knows nothing“ about directives *#def* in the program. This directive is understood by the compiler as a macro for text substitution. What does this mean for on-line changes in process will show us the following example.

```
#def    SQ7     %X0.0    ; switch

P 0
        LD      SQ7
E 0
```

In the above mentioned example, the compiler substitute during the program compilation all present words *SQ7* with string *%X0.0*. Nothing else will happen. This means that the change of the directive *#def* is not filed anyhow during the on-line change. Therefore, the programmer himself is liable for the continuity of variable´s content that are defined according to the directive *#def*

3.4.2 Access to the absolute addresses of variables

Access to the absolute addresses of variables in the user program is also not filed in on-line changes.

```
; old program
P 0
        LD      %R100    ; error_counter
E 0
```

If we change the above shown program during the on-line change, only the program code will be changed.

```
; new program
P 0
        LD      %R200    ; error_counter
E 0
```

Variables %R100 and %R200 are not during the on-line change influenced which means that the new program can (and probably also will) continue in calculation with a different value *error_counter* than the program before the change.

If we need, from any reason, to access the variable on the particular address in the memory (e.g.: owing to linkage to the visualisation program) and we want to retain advantages fo on-line changes, we can use the following procedure.

```
; old program
#reg byte   100,error_counter     ; == %R100
```

```
P 0
        LD      error_counter
E 0
```

If we change the above shown program during the on-line change, the variable *error_counter* will keep its original value also after the on-line program change.

```
; new program
#reg byte   200,error_counter     ; == %R200
P 0
        LD      error_counter
E 0
```

The variable, thus, must be declared by the directive *#reg*.

3.4.3 Timers, counters, shift registers

In the instruction file of Tecomat systems are instructions for timers, counters and shift registers that use for their functioning internal subsidiary variable. It is used as a memory of entering edges of counted inputs etc. And is not visible from the user program. It is, however, vital for the correct functioning. Internal subsidiary variable is used by following instructions:

- timers TON, TOF, RTO, IMP
- counters CTU, CTD, CNT
- shift registers SFL, SFR

If these instructions should be functioning correctly during the on-line change, their operand must be formulated in the program by symbolic name of the variable. In this case the Mosaic environment in cooperation with the PLC central unit ensure the retention of the variable value including internal subsidiary variable.

4. List of errors during the on-line change

70	xx	xxxx	Errors announced by the central unit during the on-line change (technology control continue with the original program)
70	05	0000	incorrect map lenght of the new user program
70	06	0000	incorrect security sign (CRC) of the map of the new user program
70	07	0000	incorrect security sign (CRC) of the whole user program
70	09	0000	New program is compiled for a different line of central units
70	24	0000	List of on-line changes missing
70	25	0000	List of on-line changes have faulty CRC

70 31 rr pp	In definition of I/Omodule is missing the initialization table
70 43 rr pp	In definition of I/Omodule is exceeded max. rack number

where rr ... rack number and pp ... position of I/O module on the rack