Communication server PLCComS

User manual

11. edition - may 2014

History of changes

Date	Version	Description of the changes
March 2010	1	First version.
April 2010	2	Processing variables of types *date*, *dt*, *time* a *tod*. Extending the example of processing of file attributes.
August 2010	3	Links to shared module were added. Extension and modification of the configuration file. Color highlighting examples in “C” language. Chapter with examples of application server and licenses was added.
June 2011	4	Extending the configuration file..
August 2011	5	Extended description of chapter **3.2 User public file** . Table **5.1 List of messages** corrected.
September 2011	6	Chapter **7 Licence** corrected and table **8 Supported Operating Systems** extended.
October 2011	7	Chapters **2.1 Example of the configuration file**, **4 Client protocol** and table **4.1 List of commands** edited.
March 2013	8	Chapter **4.2.2 Command SET:** edited.
August 2013	9	Chapter **2.1 Example of the configuration file** edited and table **8 Supported Operating Systems** extended.
May 2014	10	Chapter **2.1 Example of the configuration file** edited. Table **4.1 List of commands** and **5.1 List of messages** extended. Chapter **4.2.6 Command GETMEM:** added and chapter **4.2.9 Command GETINFO:** updated.
May 2014	11	Chapter **2.1 Example of the configuration file** edited.

Content

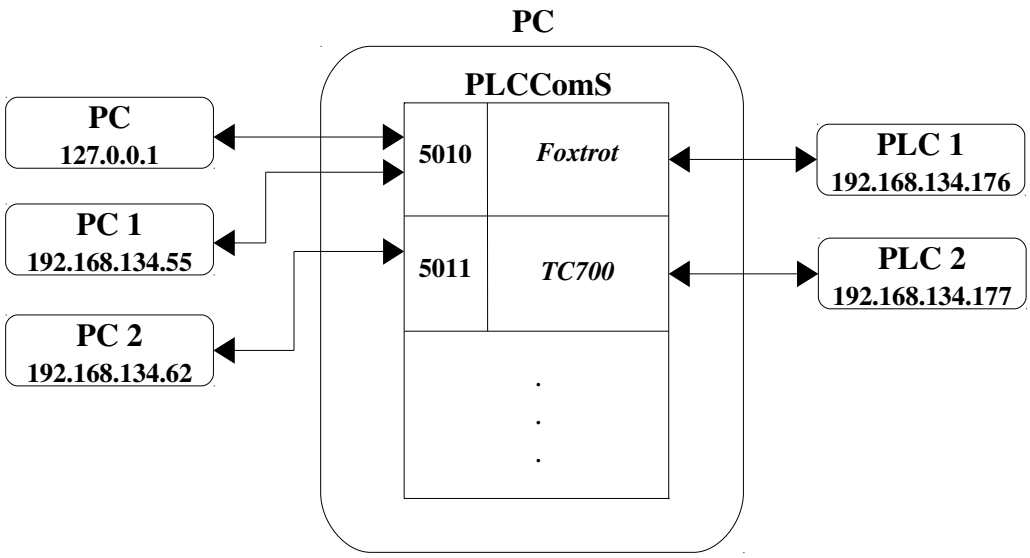
1 System description

Communication server provides TCP/IP connection between client device and PLC.

Server communication with clients is solved using a simple text-oriented protocol REQUEST/ANSWER. Therefore, the client queries the server by commands (*Chap. 4.1*) for the values of variables whose names are symbolic and are described in *public* file (*Chap. 3*). Server sends to client only error messages and variables whose values have changed.

Server communicates with the PLC using the optimized protocol EPSNET or with shared module in case of SoftPLC. The values of variables are polled by absolute addressing and periodically in time increments of 100ms.

Configuration is set in *ini* file.



Server can be run with the following parameters:

PLCComS [-dh] [-c <configuration_file>] [-l <log_file>]

-d, --daemon	*Server runs in the background.*
-c, --config	*Configuration file.*
-l, --log	*Log file.*
-h, --help	*Displays help.*

In case of run with parameter *-d* the server runs in the background and all reports are forwarded to a log file. Unless its name is specified, it is set to “PLCComS.log”. This option is only available in a version for the Linux operating system.

In case the parameter was not specified *-c*, its name is set for “PLCComS.ini”.

Unless the parameter *-l*, is not specified all the reports are sent to the terminal server from which it was run.

2 Configuration *ini* file

It consists of sections where the section marked as “[*]” is mandatory and contains global settings. Behind this section the others follow describing the setting for the particular PLC. There must be at least one such section. The name of such section can be any except “*”. The name is used in the log file entries for convenience.

2.1 Example of the configuration file

```
[*]

COMM_LOOP_DELAY  = 100   # The delay in the main loop ([1 - 1000]ms)
NET_CONNECT_MAX  = 128   # Maximum number of client connections (Maximum is
                           1024)
MEM_BLKSIZE      = 4096  # Block size in bytes for transfer PLC memory
                           (Maximum is 65536)
FFILE_BLKSIZE    = 16384 # The block size in KB for file transfer
                         # (Maximum is 65536)
FFILE_TIMEOUT    = 300   # The time limit of the file (Maximum is 3600)
FFILE_MAXRECS    = 256   # Number of files stored in the memory
                           (Maximum is 1024)
END_LINE_CRLF    = Yes   # End character of the line (Yes = DOS [\r\n],
                           No = UNIX[\n])
PF_VAR_DISABLED  = Yes   # The default state of variables
DIFF_VAR_ENABLED = Yes   # Disables or enables listings DIFF: messages
SYNC_VAR_ENABLED = Yes   # Enable or disable synchronization variables with
                           PLC while downloading files.
LIM_OF_DECIMALS  = No    # Restricts precision floating point numbers to
                           generate DIFF: messages (Yes = restrict)
NUM_OF_DECIMALS  = 10    # The number of decimal digits that will be displayed
                           or restricted ([1 - 6] REAL, [1 - 15] LREAL)
SCIENT_NOTATION  = No    # Scientific notation for REAL a LREAL (Yes =
                           scientific [-]d.ddd e[+/-]ddd, No = normal
                           [-]ddd.ddd)

#[Symbolic name of PLC]

#IPADDR        = IP address PLC
#SERVER_PORT   = The port number on which the client will communicate with the
                 PLC
#SERIAL_DEVICE = Name of serial line
#SERIAL_SPEED  = Communication speed for serial line
#SERIAL_RTS    = Number of GPIO pin for flow control. If not set, hardware
                 flow control will be used.
#SHM_NAME      = Name of the shared module (library *.dll or *.so)
#SHM_SOCKET    = Socket number for communicating with a shared module
#SHM_PORT      = Port number for communicating with a shared module
#PUBFILE_CRC   = Turn On/Off the CRC check of public file. [Yes/No]
#PUBFILE       = Name of public file.
#PUBFILE_FIXED = Name of fixed public file.

[Foxtrot]

IPADDR        = 192.168.134.176
SERVER_PORT   = 5010
PUBFILE_CRC   = Yes
PUBFILE_FIXED = FIXED_Foxtrot.pub
PUBFILE       = //www/webmaker.pub
```



```
[TC700]

IPADDR        = 192.168.134.177
SERVER_PORT   = 5011
PUBFILE_CRC   = Yes
PUBFILE_FIXED = FIXED_TC700.pub
PUBFILE       = //www/webmaker.PUB

[TC700 RS232]

SERIAL_DEVICE = /dev/ttyS0
SERIAL_SPEED  = 115200
SERVER_PORT   = 5012
PUBFILE_CRC   = Yes
PUBFILE_FIXED = FIXED_TC700.pub
PUBFILE       = //www/webmaker.PUB

[SoftPLC]

SERVER_PORT   = 5013
SHM_NAME      = ShmSrv.dll
SHM_SOCKET    = 0
SHM_PORT      = 5
PUBFILE_CRC   = Yes
PUBFILE_FIXED = FIXED_SoftPLC.pub
PUBFILE       = //www/webmaker.pub
```


3 Configuration *public* file

Describes the correlation between the variable name and physical address register in the PLC. Two types of file are used, fixed and user. Files can be stored either in the directory from which the server is running or in PLC. When you open a file the file that is stored locally has the priority. If not found, it tries to open a file in the PLC. If the file name starts with "//" the root directory is mentioned. In case of local storage of the file the root directory is that one from which the server is running. By specifying a relative path, it is possible to access local files stored outside this root directory.

3.1 Fixed *public* file

Is the file needed for the actual operation of the server. It contains a variables with which it is possible to work in the same way as with variables from the user *public* file. With the exception of command "*DI:*", which ignores that variables.

3.1.1 List of variables

Name	Data type	Meaning
__PLC_RUN	BOOL	Status of PLC. (1 = RUN, 0 = HALT)
__PF_CRC	DWORD	CRC value of user *public* file. when it is changed, the server automatically retrieves if by new *public* file.

3.2 User *public* file

Is the file generated by the *Mosaic* user program development package. The variables in this file are connected by the server to variables from *fixed* file. In case of a collision of names the variables in this file are ignored.

The file is provided with a CRC value that is equal to the value of the variable __PF_CRC. Changing the value of this variable represents the change in the PLC project (i.e. user program). Thus the server monitors this change and if it occurs, will load a new file and report this to the client by message "*WARNING:250 Changed public file: 'Test.pub'*".

4 Client protocol

A text-based protocol, where each statement is terminated by character ":" and each end of line either by "\r\n" (DOS) or "\n" (UNIX). Type a line terminator is selectable in configuration file or by the command. Settings using the command affects only the client that set it out. The command names are not case sensitive, and therefore perhaps any combination is allowed. To test the connection, or diagnostics *Telnet* program can be used, where it is possible to close the connection using the escape sequence "ctrl+d".

4.1 List of commands

Command	Description
LIST:\n	List all variables from *public* files.
SET:<variable_name, value>\n	Sets a variable in the PLC to the specified value.
GET:<variable_name>\n	Gets the value of a variable of the PLC.
EN:<variable_name>\n	Enable the variable(s) from *public* file.
DI:<variable_name>\n	Enable the variable(s) from *public* file.
GETMEM:<variable_name mem_size>	Get memory block from PLC.
GETFILE:<file_name>\n	Gets the file from PLC
GETFILEINFO:<file_name>\n	Gets the information about the file stored in PLC.
GETINFO:[name]\n	Lists information about the communication server.
Name	
version	Version of the communication server.
epsnet_version	Version of Epsnet library.
version_ini	Version of *ini* parsing library.
version_plc	PLC version.
ipaddr_plc	IP address of PLC.
serial_device	Settings of serial line.
pubfile	The names of actual *public* files.
network	List of connected clients.
SETCONF:<variable_name, value>\n	Change the value of the variable specified in the configuration *ini* file.
variable name:	
ipaddr	IP address of PLC.
serial_device	Name of serial line.
serial_speed	Communication speed for serial line.
pubfile	Public file.

crlf	End of line character (yes = DOS, no = UNIX).
diff	Suppress message list "*DIFF:*" (yes / no).
HELP:\n	List the help

When entering the name of variables you can use wild cards. The character "*" in the name replaces any number of characters. Number of characters "*" is not limited.

The server response always begins with the name of the command that caused it. With exception of "*SET:*" command, where the answer starts with "*DIFF:*". Thus in case that this response was not disabled (*Chap. 2*, *4*). In case of command, where the answers can give multiple results, the list of server responses is terminated by the command name itself.

Example of list of variables whose name starts with "test_" string:

Query:

*GET:test_**\n

Answer:

GET:test_1,123\n
GET:test_2,1.234500\n
GET:\n

Example query to specific variable:

Query:

GET:test_1\n

Answer:

GET:test_1,123\n

4.2 Client protocol - examples

4.2.1 Command LIST:

Query:

LIST:\n

Answer:

LIST:test_1\n
LIST:test_2\n
LIST:test_string\n
*LIST:test_btn**\n
LIST:\n

The character "*" at the end of the variable name means the prohibited variable. Handling with such variable by commands e.g. "*GET:*" or "*SET:*" results the error *(Chap. 5).*

4.2.2 Command SET:

Query:

SET:test_1,123\n

Answer:

DIFF:test_1,123\n

Query:

SET:test_string,"Hello!"\n

Answer:

DIFF:test_string,"Hello!"\n

Answer beginning with string "*DIFF:*" is server response for the change of some variable in PLC. In combination with the command "*SET:*" it can be used to check if the variable was really changed. Values of variables of type *string* are placed in quotation marks. Values of variables of type *bool*, is possible to set by the expression *true* (1) or *false* (0). The answer is always value 1 or 0.

4.2.3 Command GET:

Query:

GET:test_1\n

Answer:

GET:test_1,123\n

Query:

GET:test_string\n

Answer:

GET:test_string,"Hello!"\n

In case of query for value of type *date* or *dt* the value is sent as 64-bit unsigned integer, even if the standard define the variable as type real. It is because of possible rounding during transmission. The fractional part contains the amount of milliseconds.

Examples in "C" language:

```
time_t t;
struct tm *tm;
unsigned int ms;
unsigned long long int ldate;
double date;
```



```
...

sscanf (msg, "%llu", &ldate);

date = *(double *) &ldate;

ms = (unsigned int) ((date - (unsigned long long int) date) * 1000);

t  = (time_t) date;
tm = gmtime (&t);

if (tm == NULL)
  return 1;

printf ("Date: %02d.%02d.%d", tm->tm_mday, tm->tm_mon + 1, tm->tm_year +
                              1900);
printf ("Time: %02d:%02d:%02d.%03d", tm->tm_hour, tm->tm_min, tm->tm_sec, ms);

...
```

If the variable is of type *time* or *tod* the server sends the value as 32-bit unsigned integer as is defined in the standard.

Example in “C” language:

```
struct tm tm;
unsigned long time;

...

sscanf (msg, "%lu", &time);

memset (&tm, 0, sizeof (struct tm));

tm.tm_hour  = time / 3600000;
time -= (tm.tm_hour * 3600000);
tm.tm_min   = time / 60000;
time -= (tm.tm_min * 60000);
tm.tm_sec   = time / 1000;
tm.tm_isdst = time - (tm.tm_sec * 1000);

printf ("Time: %02d:%02d:%02d", tm.tm_hour, tm.tm_min, tm.tm_sec);

...
```


4.2.4 Command EN:

Query:

EN:\n

4.2.5 Command DI:

Query:

DI:\n

Commands "*EN*" and "*DI*" enable or disable variables in communication table of the server. If the variables are disabled, server does not ask the PLC for them. Thus the volume of data to be transferred is decreased. After the start of the server all variables are enabled. Disabling does not affect the variables in the fixed *public* file.

4.2.6 Command GETMEM:

Query:

GETMEM:test_string 6\n

Answer:

GETMEM:test_string[6]=Hello!\n
GETMEM:test_string[0]=\n

The contents of the memory is transferred in blocks of size given by variable MEM_BLKSIZE, which is set in configuration *ini* file. The actual length of the block is indicated in brackets at the end of the variable name.

4.2.7 Command GETFILE:

Query:

GETFILE://www/TEST.TXT\n

Answer:

GETFILE://www/TEST.TXT[20]=This is test string.\n
GETFILE://www/TEST.TXT[0]=\n

The contents of the file is transferred in blocks of size given by variable FFILE_BLKSIZE, which is set in configuration *ini* file. The actual length of the block is indicated in brackets at the end of the file name.

4.2.8 Command GETFILEINFO:

Query:

GETFILEINFO://www/TEST.TXT\n

Answer:

GETFILEINFO://www/TEST.TXT[35]=21 32 59391128503405 5939112850
3405\n

The actual length of the block is indicated in brackets at the end of the file name.
Structure of the message:

size attributes time_creation time_change

size - size of file in bytes.

attributes - 32 bit number interpreted per bits.

time_creation - 64 bit number which can be interpreted by the structure *ttida*.

time_change - 64 bit number, which can be interpreted by the structure *ttida*.

4.2.8.1 List of *attributes*

Hexadecimal value	Meaning
0x00000001	Only for reading
0x00000002	Hidden file
0x00000004	The system file
0x00000010	The file is a directory
0x00000020	The archive file

structure *ttida* - Size 8 bytes (year, month, day, hour, minute, second, nothing, nothing).

Example in “C” language:

```
#define FILE_ATTRIBUTE_READONLY         0x00000001
#define FILE_ATTRIBUTE_HIDDEN           0x00000002
#define FILE_ATTRIBUTE_SYSTEM           0x00000004
#define FILE_ATTRIBUTE_DIRECTORY        0x00000010
#define FILE_ATTRIBUTE_ARCHIVE          0x00000020

struct ttida {char year; char mon; char day; char hour; char min; char sec;
              char none1; char none2;} tida_c, tita_m;

...

long size, attr;

sscanf (msg, "%ld %ld %llu %llu", &size, &attr,
        (unsigned long long int *)&tida_c, (unsigned long long int *)&tida_m);
```



```
printf ("File size  : %ld bytes\n", size);

printf ("Attributes : %c%c%c%c%c\n", (attr&FILE_ATTRIBUTE_READONLY )?'R':'W',
                                     (attr&FILE_ATTRIBUTE_HIDDEN   )?'H':'-',
                                     (attr&FILE_ATTRIBUTE_SYSTEM   )?'S':'-',
                                     (attr&FILE_ATTRIBUTE_DIRECTORY)?'D':'-',
                                     (attr&FILE_ATTRIBUTE_ARCHIVE  )?'A':'-');

printf ("Create time : %02d.%02d.%04d %02d:%02d:%02d\n", tida_c.day,
       tida_c.mon,tida_c.year + 1900, tida_c.hour, tida_c.min, tida_c.sec);

printf ("Modify time : %02d.%02d.%04d %02d:%02d:%02d\n", tida_m.day,
       tida_m.mon, tida_m.year + 1900, tida_m.hour, tida_m.min, tida_m.sec);

...
```

4.2.9 Command GETINFO:

Query:

GETINFO:\n

Answer:

GETINFO: VERSION,Ver 3.2 Apr 30 2014 08:15:16\n
GETINFO: VERSION_EPSNET,Ver 2.0 Apr 11 2014 10:54:08\n
GETINFO: VERSION_INI,Ver 3.2 Dec 18 2012 07:41:30\n
GETINFO:VERSION_PLC,CP1005K B 2.7 5.1 \n
GETINFO:IPADDR_PLC,192.168.134.176 \n
GETINFO:PUBFILE,5/5 [FIXED_Foxtrot.pub] [//www/webmaker.pub] \n
GETINFO:NETWORK,1/10 [127.0.0.1] \n
GETINFO: \n

Query:

GETINFO:version_plc\n

Answer:

GETINFO:VERSION_PLC,CP1005K B 2.7 5.1 \n

5 Error messages

Server sends two types of messages. The error ones that begin with ”*ERROR:*” and warning ones that begin with “*WARNING:*”. Messages are subdivided into groups each of ten messages.

5.1 List of messages

Error code	Text of message	Group	Type
10	Unable to connect to PLC.	Network communication	Error
11	Maximum connections reached.	Network communication	Error
20	Unable to connect to PLC.	Communication with PLC	Error
30	Bad client request.	Client queries	Error
31	Incomplete client request.	Client queries	Error
32	Unknown command name:	Client queries	Error
33	Unknown register name:	Client queries	Error
34	Disabled register name:	Client queries	Error
40	Unable to get information about file:	File operations	Error
41	Unable to get file:	File operations	Error
50	Unknown name:	Command “*SETCONF:*”	Error
60	Unknown name:	Command “*GETINFO:*”	Error
70	Unable to connect to SHARED module.	Shared module	Error
80	Unable to get data from address:	Command “*GETMEM:*”	Error
250	Changed public file:	Command “*SETCONF:*”	Warning
1024	Unknown error.	Not specified	Error

Examples of error messages:

Query:

GET:test\n

Answer:

ERROR:33 Unknown register name: 'test'\n

Query:

SETCONF: pubfile, Test.pub\n

Answer:

WARNING:250 Changed public file: 'Test.pub'\n

6 Examples of application

Application of the communication server *PLCCOmS* can be demonstrated on a few simple example programs.

6.1 Example of the simulated PLC in an environment *Mosaic*

Consider a simple program written in the language of structured text “ST”, which will read information about the system time and date, for example, and will calculate the value of a continuous function *sine* and *cosine* depending on run-time. In addition, we have a variable *positive* signaling the positive half-waves of the function *sinus* and one variable of type *STRING* containing the text string.

As testing PLC we choose *FOXTROT CP-1016*.

Example in “ST” language:

```
VAR_GLOBAL // Global public variables
  dat      {PUBLIC} : DATE;
  tim      {PUBLIC} : TIME;
  sinus    {PUBLIC} : LREAL;
  cosinus  {PUBLIC} : LREAL;
  c        {PUBLIC} : LREAL;
  positive {PUBLIC} : BOOL;
  text     {PUBLIC} : STRING;
END_VAR

VAR_GLOBAL CONSTANT
  PI : LREAL := 3.14159265358979323846;
END_VAR

PROGRAM prgMain
  VAR
  END_VAR
  VAR_TEMP
  END_VAR

  dat := GetDate();    // Returns the actual date
  tim := GetTime();    // Returns actual time

  c := c + 0.001;      // The time course for the calculation of sine and
                       // cosine

  sinus   := SIN(c);   // Returns sine value of argument
  cosinus := COS(c);   // Returns cosine value of argument

  IF sinus > 0.0 THEN
    positive := TRUE;
  ELSE
    positive := FALSE;
  END_IF;

  text := 'Have a nice day';

END_PROGRAM
```


Server allows you to monitor only those system variables that are defined as (PUBLIC). So as variables that can be accessed from outside the program. Public variables can be defined by directive "{PUBLIC}" directly or by dialogue window "*Variable definition*", where it is necessary to tick the option of the same name (*Fig. 6.1.1*).

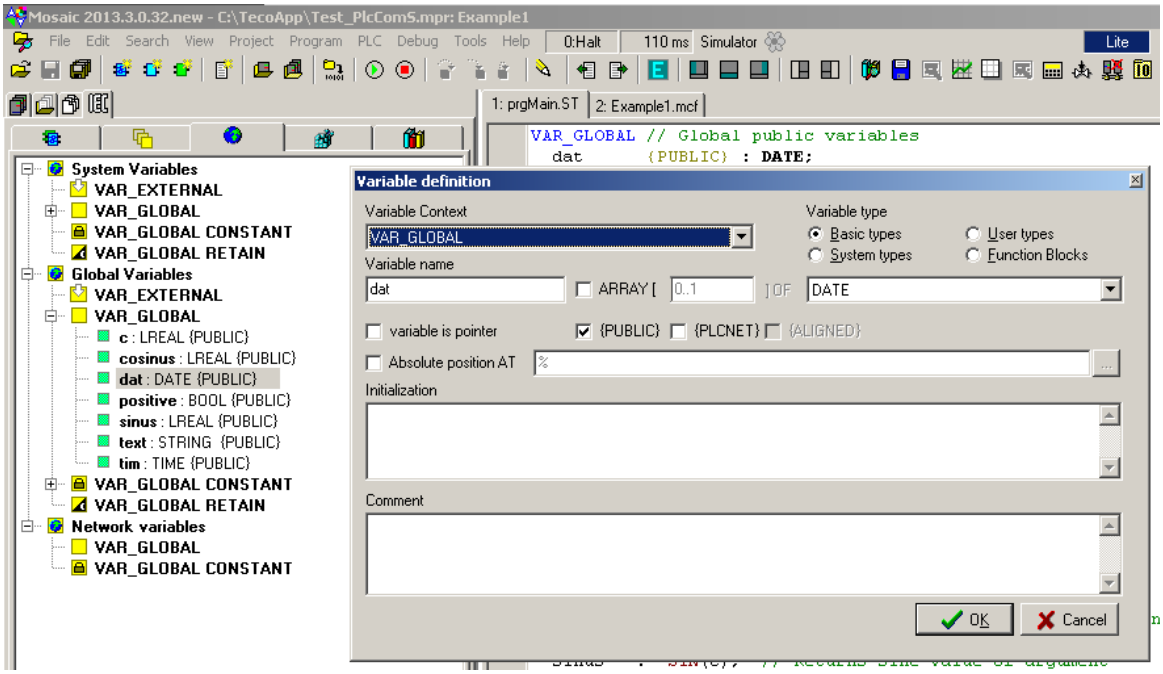


Fig. 6.1.1: The dialog box "Define variables"

If you already have defined all public variables that you want to monitor, you must generate a *public* file (**.pub*), a file containing information about the declared public variables. This file may not be automatically generated by the *Mosaic* package. In such case it is necessary to set the generation of that file in "*Project manager*" in the setting menu branch "*Sw – Export files*", where it is necessary to check the option "*IEC 1131*" (file *.exp*) and option "*Assembler*" (file *.pub*) (*Fig. 6.1.2*).

The *public* file generated after the user program compilation is stored in the project directory under the name "*project_name.pub*". The name of this file is used then as the configuration parameter of *ini* file for application of the server *PLCCOmS*.

In case that the file name does not include the relative path in the project directory, it is necessary to copy the *public* file in the same directory from where the server is started!

The finished testing program has to be stored. In "*Project manager*" it is necessary to choose "*Type of connection*" (if the PLC is connected, it has to be disconnected by the "*Disconnect*" button) "*Simulated PLC"* with the checked option "*Mosaic PLC"* and to connect PLC by button "*Connect"* (*Fig. 6.1.3*). We compile the program and run it in simulation mode.

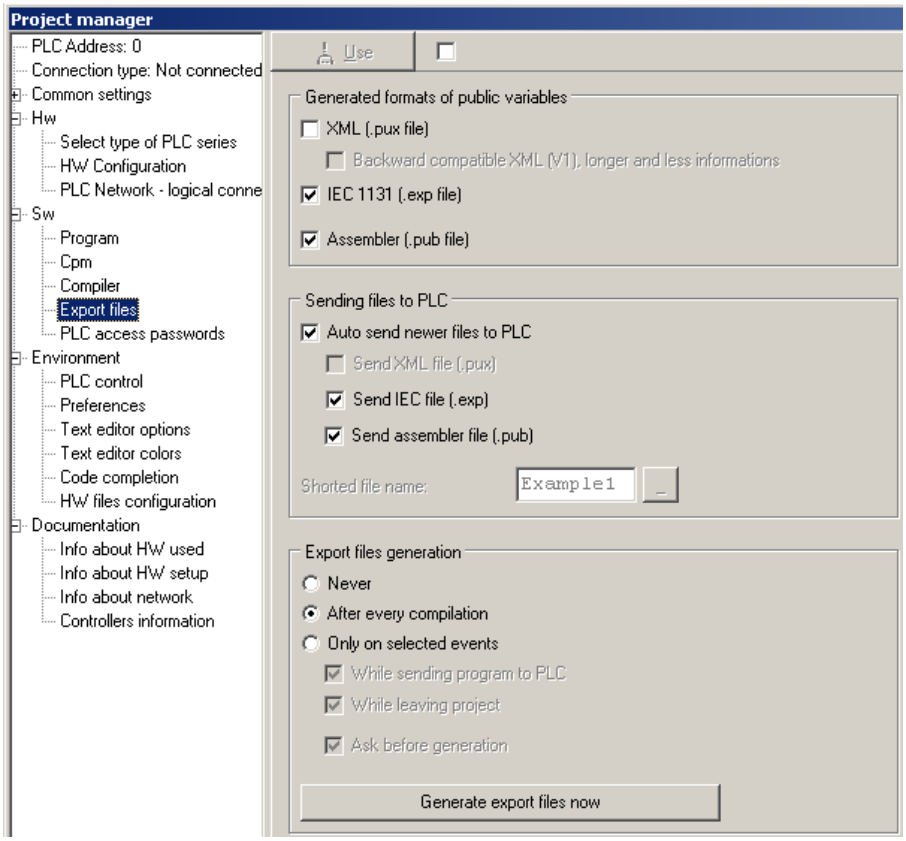


Fig. 6.1.2: Setting of generating export files

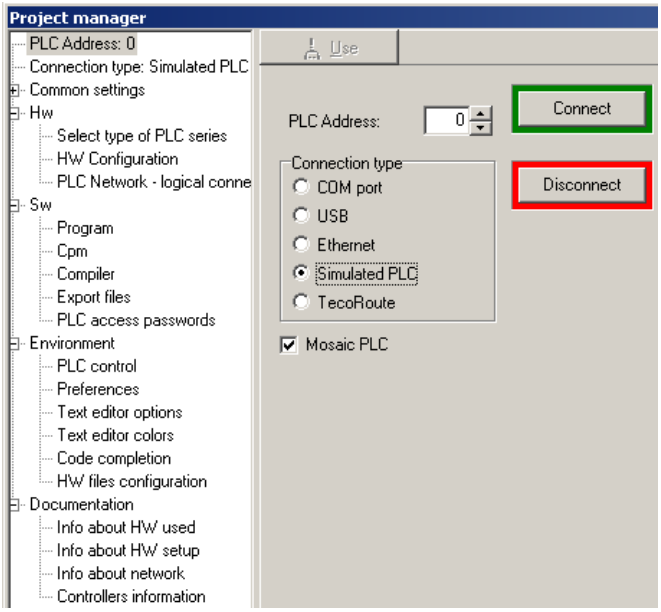


Fig. 6.1.3: Connection of the simulated PLC

Run server with parameters *-c* and *-l*. It will therefore be necessary to specify the name of the configuration (**.ini*) and log (**.log*) file. In the case of existence of log file the messages of the server are attached to the end of file. The configuration file must be defined. For our example, when observed PLC is simulated by *Mosaic* on the local station (*localhost – 127.0.0.1*) the contents of the configuration file looks like this:

The contents of the configuration file:

```
# Configuration file for communication server

[*]

NET_CONNECT_MAX = 10          # Maximal amount of clients (Max 32)
FFILE_BLKSIZE   = 1024        # Maximal size of block for files (Max 65536)
FFILE_TIMEOUT   = 300         # Time in seconds for the keeping file
                                in memory (Max 3600)
FFILE_MAXRECS   = 256         # number of files stored in the memory (Max 1024)
END_LINE_CRLF   = Yes         # End of line character (Yes = DOS [\r\n], No =
UNIX [\n])
PF_VAR_DISABLED = Yes         # Default status for variables

[FOXTROT]                     # Alias of used PLC

IPADDR        = 127.0.0.1     # IP address PLC
SERVER_PORT   = 5010          # Port of server
PUBFILE_CRC   = No            # not to verify the checksum for PUBFILE

PUBFILE_FIXED = <fixedpubfile_name.pub>   # Fixed PUBFILE
PUBFILE       = <pubfile_name.pub>        # PUBFILE
```

If we have the configuration file prepared, we can run the server from the command prompt:

```
..>PLCCOmS.exe -c <config_name> -l <log_name>
```

The success of server startup can be watched in the log file or in case of missing *-l* parameter in the terminal window.

With the running server *PLCCOmS* can someone communicate by the program *Telnet*. For this purpose can be used widespread application *Putty*, through which one can be conveniently connected to the server (*Fig. 6.1.4*).

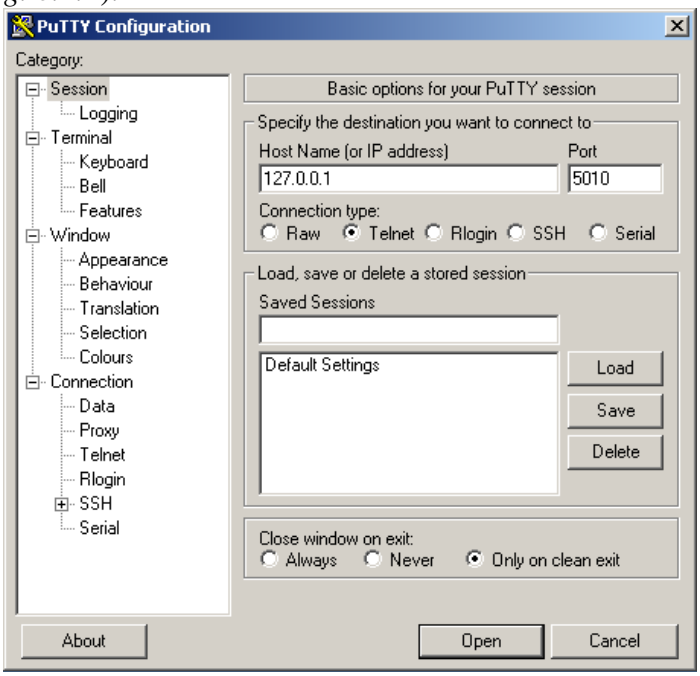


Fig. 6.1.4: Connecting to the PLCCOmS server using Putty

In the program *Putty* only IP address of the connecting server (if the server *PLCCOmS* runs on the local station, the address is *127.0.0.1*) and the port number on which the server is listening (in our case it is port *5010*). Pressing the button "*Open*" will open the connection.

It is now possible to monitor all public variables that we have in our virtual PLC defined. For these purposes it is necessary to use commands defined by application *PLCCOmS* (e.g. command "*LIST:*" returns a list of all the variables in the PLC).

Server activity can be terminated by the escape sequence "ctrl+c". All the important events of server activity are recorded in the log file.

6.2 Example with the real PLC

We will use the same program as in the previous case. Only instead of a simulated run now we use a real machine, e.g. *FOXTROT CP-1004*.

The set-up steps in the *Mosaic* (generating export files *PUBFILE*) and in the program itself *PUBLIC* variables) are identical to the first example. The only steps the procedures differ is the type of connection in "*Project manager*", where the option "*Simulated PLC*" is not used, but the real PLC on the local ETHERNET network is used defined by its IP address (*Fig. 6.2.1*). Connect the PLC by the button "*Connect*", compile the program and turn the PLC into "RUN" mode.

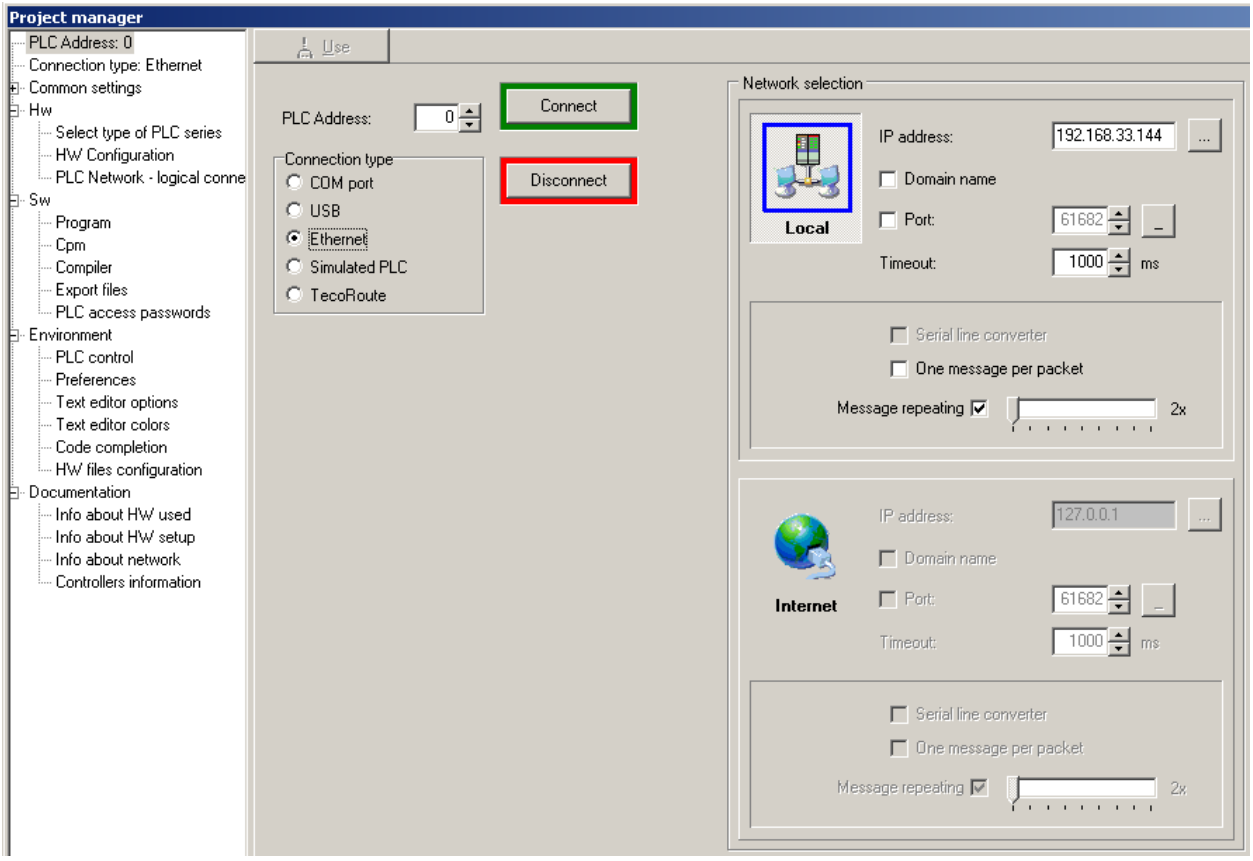


Fig. 6.2.1: Connecting the remote PLC

The configuration file of the *PLCCOmS* application undergo only minor changes comparing to the first example. The only item that is necessary to amend is *IPADDR* (IP address of the PLC), which we change from the local address *127.0.0.1* (for simulated mode) on IP address of PLC local network (e.g. *192.168.33.144*).

Now you can start the *PLCCOmS* server with the specified parameters of configuration file and log file using the command from the prompt (*Example 6.1*).

For the communication with the server is again recommended to use the program *Putty* which supports TELNET with the appropriate settings from the previous example (*Fig. 6.1.4*). Warning: do not confuse the IP address of the host station, that is running the server *PLCCOmS*, with address of PLC.

Through the *Putty* program we connect the server, e.g. the IP *127.0.0.1*, if the server is running on the local station.

7 Licence

Communication server *PLCComS* is the software product (the software) Teco a.s. Kolín (the manufacturer). It is distributed free of charge and its use is possible under the following conditions:

I. Action that is especially forbidden:

a) to analyse the software in any way, change it, translate it to other programming or national languages or into the source code or into assembler, include it into other software and to distribute the resulting products derived from the original software and interfere in the internal structure with the exception of the cases referred in this agreement or explicitly permitted by copyright law.

b) any charged distribution of this software.

II. Responsibility of the manufacturer

Using the software is based on an “as is” without warranty of any kind. The manufacturer does not assume any responsibility for loss of income, profits, data, or indirect special consequential or incidental damages. The manufacturer makes no warranty on the software performance, nor to its capability for any specific use, application or purpose. In particular the manufacturer shall not be liable for any damages caused by improper operation of the software contrary with the terms specified in the user documentation.

Neither the manufacturer nor its contractors shall not be liable in any case for loss of profit or any other commercial loss, including and without limitation the special, incidental, punitive and other damages, even if the manufacturer or its contractors are prewarned of the possibility of such damage. Any risk arising from the quality and performance of the Software is transferred to the user's software. If the software proves to be defective, the user software, and not the manufacturer, shall assume all costs associated with any repairs and maintenance.

III. Duration of the agreement

This agreement shall remain in force unless the user software does not violate the terms of this agreement. If the user software violates the terms of the agreement, the agreement automatically loses the force.

IV. Final provisions

This agreement in connection with the use of software is complete and entire agreement between the user and the software manufacturer. This arrangement is a replacement for all previous, current, spoken and written communication between the user and the software manufacturer and is crucial in resolving disputes or additionally agreed terms, offers, orders the parties involved in this agreement for its validity. Any modifications to this Agreement shall be permitted only in the case of writing a new contract or addendum, which will be signed by both parties represented.

8 Supported Operating Systems

OS	Architecture	Compilator
Windows	x86	i586-mingw32msvc-g++
Linux	x86	i586-linux-gnu-g++
Linux	x86_64	x86_64-linux-gnu-g++
Linux	Armel	arm-linux-gnu-g++, arm-none-linux-gnueabi-g++
Linux	Armhf	arm-linux-gnueabihf-g++

