InternetLib library

TXV 003 54.02
first edition
October 2009
subject to alterations

Changes history

Datum	Vydání	Popis změn
October 2009	1	First edition

CONTENT

1 INTRODUCTION

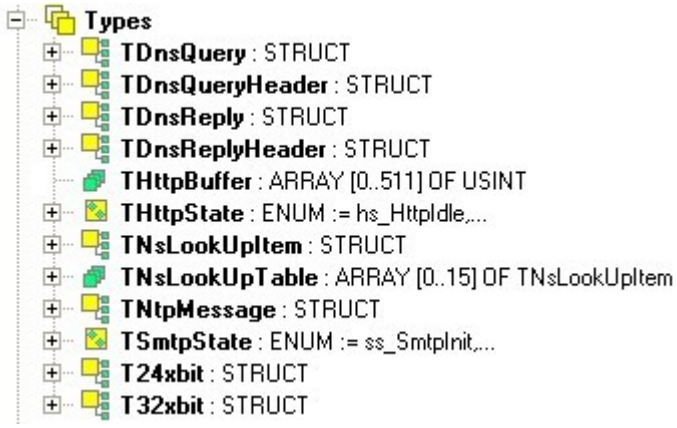
InternetLib library contains set of functions for operations with services accessible within the Internet network. The library can be used together with systems of the K line central unit with the firmware version 4.9 and higher.

Function blocks included realize the translation of domain names to IP addresses, synchronize the time with the time servers, send e-mails via SMTP protocol and basic queries of http protocol.

The library uses particular structures, functions and function blocks from libraries FileLib (TXV 003 41) and ComLib (TXV 003 51). To ensure correct functioning, these libraries must be filed in the project before the InternetLib library.

2 DATA TYPES

There are defined following data types in the InternetLib library:



Type	*Description*	*Basic type*
TDnsQuery	Query on DNS server structure	STRUCT
TDnsQueryHeader	Query on DNS server header structure	STRUCT
TDnsReply	DNS server reply structure	STRUCT
TDnsReplyHeader	DNS server reply header structure	STRUCT
THttpBuffer	Field for data received by HTTP protocol	ARRAY [0..511] OF USINT
T24xbit	24 bit words per bytes structure	STRUCT
T32xbit	32 bit words per bytes structure	STRUCT
THttpState	HTTP protocol communication status	ENUM
TSmtpState	SMTP protocol communication status	ENUM
TNsLookUpItem	Pair IP address domain name with attributes	STRUCT
TNsLookUpTable	Field of pairs IP address domain name with attributes	ARRAY [0..31] OF TNsLookUpItem

Enumeration values signification:

THttpState - HTTP protocol communication status		
0	hs_HttpIdle	Connection not set up, communication not active
1	hs_HttpSetIP	IP address setup
2	hs_HttpConnect	Waiting for connection set up
3	hs_HttpSend	Sending the prompt on server
4	hs_HttpReceivingData	Receiving data from server

TSmtpState - SMTP protocol communication status		
0	ss_SmtpInit	Initialization
1	ss_SmtpIdle	Connection not set up, communication not active
2	ss_SmtpSetIP	IP address setup
3	ss_SmtpTxConnect	Server connection establishment
4	ss_SmtpRxConnect	Waiting for server response number 220
5	ss_SmtpTxHelo	Sending a HELO command
6	ss_SmtpRxHelo	Waiting for server response number 250
7	ss_SmtpTxAuthlogin	Sending an AUTH command (authorized login requirement)
8	ss_SmtpRxAuthlogin	Waiting for server response number 334
9	ss_SmtpTxUserName	Sending user name
10	ss_SmtpRxUserName	Waiting for server response number 334
11	ss_SmtpTxPassword	Sending user name
12	ss_SmtpRxPassword	Waiting for server response number- 235
13	ss_SmtpTxMailFrom	Sending the e-mail sender address (command MAIL FROM)
14	ss_SmtpRxMailFrom	Waiting for server response number 250
15	ss_SmtpTxRcptTo	Sending recipients addresses
16	ss_SmtpRxRcptTo	Waiting for server response number 250 or 251
17	ss_SmtpTxData	Sending the command DATA
18	ss_SmtpRxData	Waiting for server response number 354
19	ss_SmtpTxDataFrom	Sending the message body - sender
20	ss_SmtpTxDataTo	Sending the message body - recipient
21	ss_SmtpTxDataSubject	Sending the message body - subject
22	ss_SmtpTxMultipart	Sending the message body – parts separator
23	ss_SmtpTxDataText	Sending the message body - text
24	ss_SmtpTxAttachement	Sending the message body – attachement separator
25	ss_SmtpTxAttachementBody	Sending the message body - attachement
26	ss_SmtpTxEndOfMail	Sending the message body – e-mail end

TSmtpState - SMTP protocol communication status		
27	ss_SmtpRxAck	Waiting for server response number 250
28	ss_SmtpTxQuit	Sending the command QUIT to end the connection
29	ss_SmtpRxClose	Waiting for server response number 221
30	ss_SmtpRxTimeout	Communication Timeout elapsed
31	ss_SmtpRxError	Error occurred during communication

3 CONSTANTS

There are no public constants defined in the InternetLib.

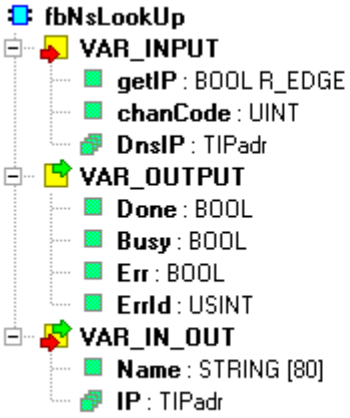
4 DOMAIN NAMES COMPILATION

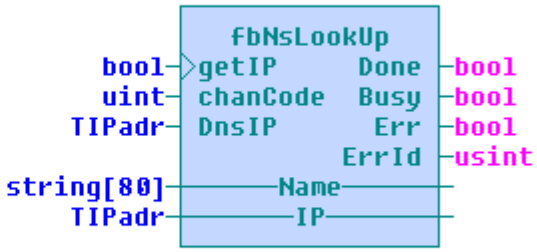
Compilation of domain names uses hierarchical system of domain names DNS (Domain name system) to get the IP address of servers with the domain name.

IP addresses of DNS servers are used to be similar, in local networks, to the address of the home portal, router or proxy server. Apart form addresses of local servers also addresses assigned by the connection provider or public DNS servers can be used. In following examples, the public DNS server provided by OpenDNS company for free is used.

4.1 Function block fbNsLookUp

library: *InternetLib*

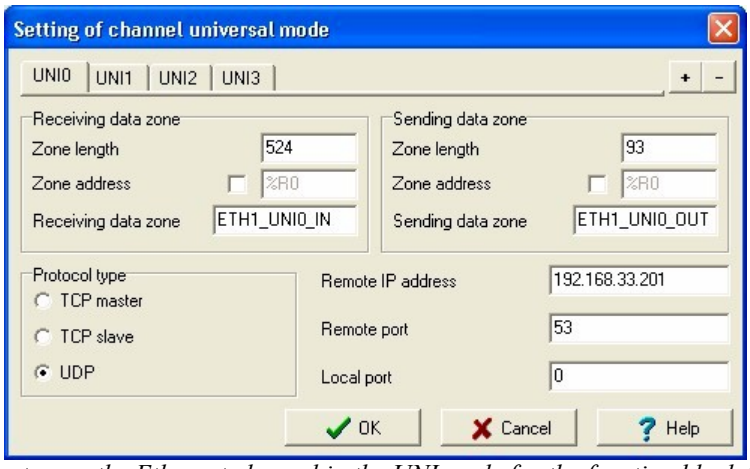




Function block *fbNsLookUp* is used to acquire the IP address according to the domain name. The request for IP address is invoked by setting the input *getIP* to the TRUE value. The request is done via the connection on the Ethernet channel in the UNI mode according to the constant on the input *chanCode*. The connection must have following parameters: UDP mode, the length of reception zone 524 bytes, length of the sending zone 93 bytes. If the connection is not active or does not have correct zone lengths, the block indicates error on outputs *Err* by TRUE value and *ErrId by value* 255.

IP address of the DNS server is transferred on the input *DnsIP*, domain name that we want to compile to the IP address is entered via the variable on the input *Name*.

During the request on the DNS server, the output *Busy i*s set. In case the request is successful, one cycle is set to output *Done*. If the request fail from any reason, outputs *Err* and *ErrId* are set. The value *ErrId* determines error type that have occured. Particular values are described in the variable descriptions. If more IP addresses is neccessary to be get form the DNS server, it is better to use the block *fbNsLookUpByTable*.



Connection setup on the Ethernet channel in the UNI mode for the function block fbNsLookUp

Variable description :

	Variable	*Type*	*Signification*
VAR_INPUT			
	getIP	BOOL R_EDGE	Control variable. Rising edge (transfer from FALSE value to TRUE value) initiate the request for IP address acquirement
	chanCode	UINT	Connection code ETH1_uni0, ETH1_uni1,...
	DnsIP	TIPadr	IP address of the DNS server
VAR_IN_OUT			
	Name	STRING	Domain name
	IP	TIPadr	IP address gained from the DNS server
VAR_OUTPUT			
	Done	BOOL	Has the value TRUE at the moment when the IP address is retrieved Otherwise, returns FALSE
	Busy	BOOL	Address acquirement process flag
	err	BOOL	Error flag If operation was successful, it has value FALSE, otherwise, TRUE.
	errID	USINT	Error code: *errID* = 0 operation was successful *errID* = 1 time for server response elapsed *errID* = 2 server did not return valid address for the name entered *errID* = 254 zero address of the DNS server *errID* = 255 error connection setup to the Ethernet channel

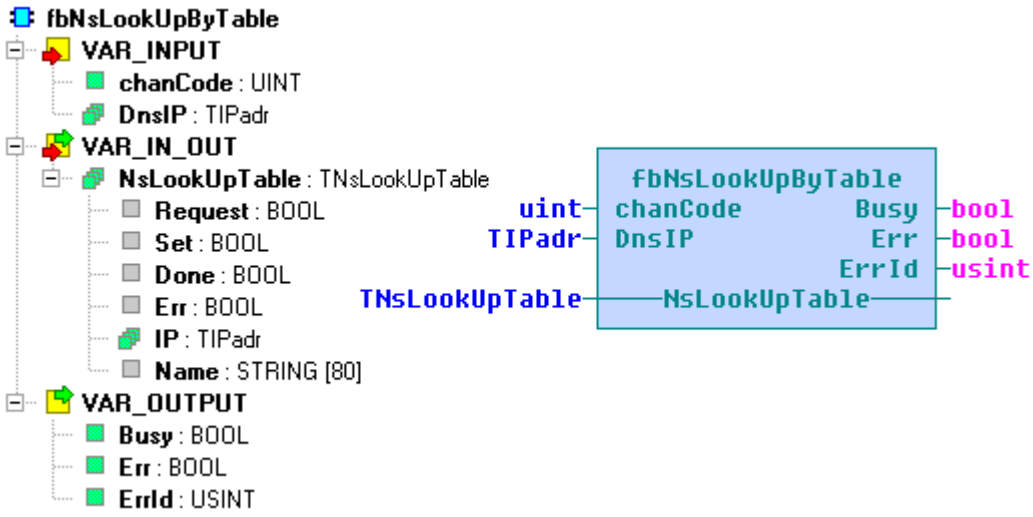
The example of the program with the function block *fbNsLookUp* call:

Variable *GetNtpIp* invokes the request for IP address of the time server which domain name is set by the variable *DomName*. In the case of successful address receipt, the bit *NtpIpReady* is set to the TRUE value.

```
VAR_GLOBAL
  GetNtpIP   : BOOL;
  NtpIpReady : BOOL;
END_VAR

PROGRAM prgExampleNsLookUp
  VAR
    NsLookUp : fbNsLookUp;
    DomName  : STRING := 'cz.pool.ntp.org';
    ServerIP : TIPadr;
    RSReady  : RS;
  END_VAR

  NsLookUp(getIP := GetNtpIP,
           chanCode := ETH1_uni0,
           DnsIP := STRING_TO_IPADR('208.67.222.222'),
           Name := DomName,
           IP := ServerIP);

  RSReady(S := NsLookUp.Done, R1 := NsLookUp.Err, Q1 => NtpIpReady);

END_PROGRAM
```

4.2 Function block fbNsLookUpByTable

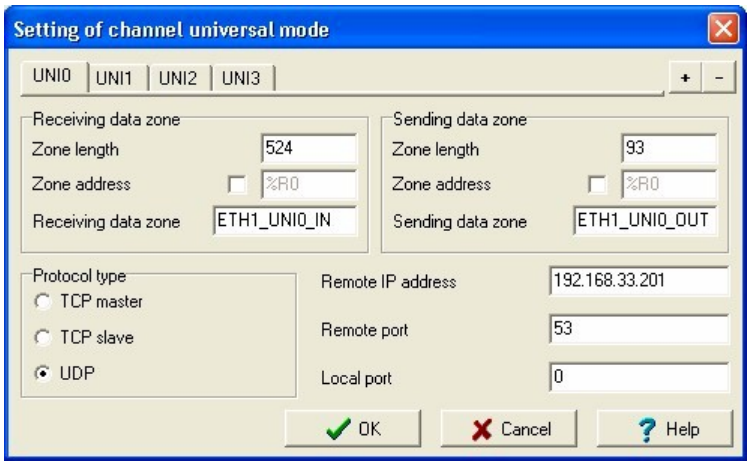
library: *InterneLib*



Function block *fbNsLookUpByTable* is used to get more IP addresses according to domain names via one connection. The block requires connection on the Ethernet channel in the UNI mode. The connection must have following parameters: UDP mode, the length of receiving zone 524 bytes, length of sending zone 93 bytes. If the connection is not active or does not have correct zone lengths, the block indicates error on outpus *Err* by the TRUE value and *ErrId by the value* 255.

The block has three inputs. The input *chanCode* determines via which connection will the block operate, *DnsIP* address of the DNS server which information will be get from and *NsLook-UpTable* refers to the structure with requirement flags, domain names and IP addresses.

The structure *NsLookUpTable* have a capacityof up to 32 pairs of domain name, IP address. Each of these pairs is equipped with a set of bit flags. By setting a bit *Request,* the request for particular IP address acquirement is filed. This bit is reset immediately after reception. At the moment when the IP address is obtained, the *Done* and *Set* bit is set. Bit *Done* is reset in the following cycle, bit *Set* after next requirement is set. In case of an error, the variable *Err* is set. Together with the *Err* variable the outpu blocks *Err* and *ErrId* with error specification code are set. During the communication the output *Busy* is set.



Connection setup on the Ethernet channel in the UNI mode for the function block fbNsLookUpByTable

Variable description:

	Variable	*Type*	*Signification*
VAR_INPUT			
	chanCode	UINT	Connection code ETH1_uni0, ETH1_uni1,...
	DnsIP	TIPadr	IP address of the DNS server
VAR_IN_OUT			
	NsLookUpTable	T*NsLookUpTable*	Table of domain names, flags and IP address
	Request	BOOL	Address request bit flag
	Set	BOOL	Successful IP address acquirement bit flag
	Done	BOOL	Rising edge of the flag *Set*
	Err	BOOL	Error bit flag during IP address acquirement
	IP	TIPadr	IP address obtained from the DNS server
	Name	STRING	Domain name which IP address is searched to
VAR_OUTPUT			
	Busy	BOOL	Has the TRUE value during the communication with the DNS server. Otherwise, FALSE is returned.
	Err	BOOL	Error flag If the last operation was successful, it has a FALSE value, otherwise, TRUE.
	ErrID	USINT	Error code: *errID* = 0 operation was successful *errID* = 1 time for the server response elapsed *errID* = 2 server did not return the valid address for the name entered

The example of the program with the function block fbNsLookUpByTable call:

In the following example, there are obtained three IP addresses of the bellow mentioned addresses after the system start (this is ensured by initialization of bites *Request*). The example does not show the use of flag bites *Done* and *Set*. These flags can be used anywhere further within the program. The flag *Done* can be used for the action initialization immediately after the IP address is gained. The command *Set* can be used to control whether the IP address is gained successfully and is possible to use it for further communication.

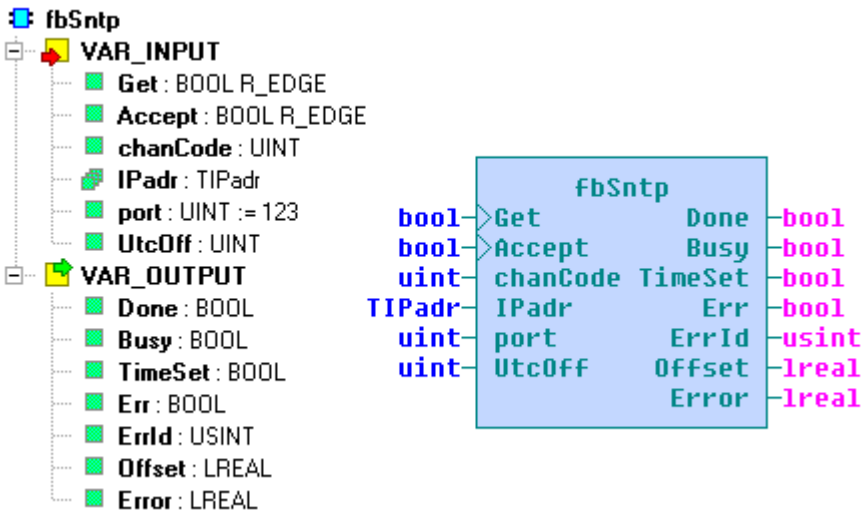
```
VAR_GLOBAL
  LookUpTable : TNsLookUpTable :=
    [(Request:= true, Name:= 'cz.pool.ntp.org'),
     (Request:= true, Name:= 'smtp.iol.cz'),
     (Request:= true, Name:= 'kamera.mukolin.cz')];
END_VAR

PROGRAM prgExampleNsLookUpByTable
  VAR
    NsLookUpByTable : fbNsLookUpByTable;
  END_VAR

  NsLookUpByTable(chanCode := ETH1_uni0,
                  DnsIP := STRING_TO_IPADR('208.67.222.222'),
                  NsLookUpTable := LookUpTable);

END_PROGRAM
```

5 TIME SYNCHRONIZATION

Time synchronization uses SNTP (Simple Network Time Protocol) protocol to acquire time difference of the internal clock compared to time of the time server. This difference can be used for system time synchronization. The time server can be operated in the local network or public servers can be used. The list of public servers can be found on the internet address support.ntp.org.

5.1 Function block fbSntp

library: *InternetLib*

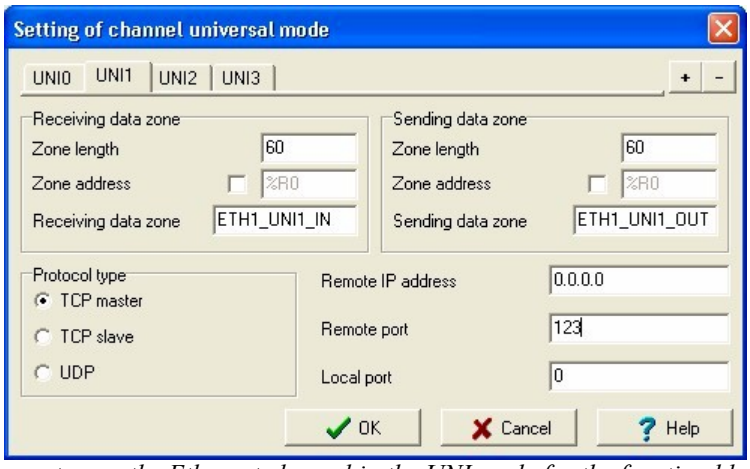


Function block *fbSntp* is used to obtain the time difference between sever and system time of the PLC. The requirement on time difference is invoked by setting the input *Get* to the value TRUE. Request is undertaken via the connection on the Ethernet channel in the UNI mode according to the constant on the input *chanCode*. The connection must have following parameters: UDP mode, length of receiving and sending zone 60 bytes. If the connection is not active or does not have the correct zone lengths, the block indicates an error on outputs *Err* by the value TRUE and *ErrId* by the value 255.

The address of the time server is transferred on the input *IPadr* and port where server receive requirements is set by input *port* (default value for the SNTP protocol is 123). On the input *UtcOff* the time zone shift is expected compared to GMT in minutes.

During the request for the time difference, the output *Busy* is set. When the operation is finished successfully, the obtained time difference appears on the output *Offset*, on the output *Error* the maximum error of the difference gained and the output *Done* is set for the time of one cycle. In case of failure the output *Err* and *ErrId* is set where the error specification code is.

After successful time difference acquirement, the system time of the PLC can be synchronized by setting the input *Accept* to the value TRUE. If the input *Accept* is set to the value TRUE, the system time is set immediately after a successful time difference acquirement. If the time difference was obtained successfully, the system time correction is undertaken with the rising edge on the input *Accept*. The successful setup of the PLC system time according to the time difference obtained is indicated by setting the output *TimeSet*.



Connection setup on the Ethernet channel in the UNI mode for the function block fbSntp

Variable description :

	Variable	***Type***	***Signification***
VAR_INPUT			
	Get	BOOL R_EDGE	Control variable. Rising edge initiates the request on time difference
	Accept	BOOL R_EDGE	Time setup according to offset gained.
	chanCode	UINT	Connection code ETH1_uni0, ETH1_uni1,...
	IPadr	TIPadr	IP address of the time server
	port	UINT	Port of time server (default value for protocol SNTP is 123)
	UtcOff	UINT	Time zone shift compared to GMT in minutes
VAR_OUTPUT			
	Done	BOOL	Has a value TRUE at the moment when time difference is obtained. Otherwise, FALSE is returned
	Busy	BOOL	Has a value TRUE during time difference acquirement
	TimeSet	BOOL	Has a value TRUE if the last obtained time difference was used for system time setup
	Err	BOOL	Error flag If the last operation was successful, it has a FALSE value, otherwise, TRUE.
	ErrId	USINT	Error code: *errID* = 0 operation was successful *errID* = 1 server response time elapsed *errID* = 2 time difference was not determined form the server response *errID* = 254 zero address of the time server *errID* = 255 error connection setup on the Ethernet channel
	Offset	LREAL	Time difference obtained
	Error	LREAL	Max error of the time difference obtained

Following example shows the use of the function block *fbSntp* for precise time acquirement. Program requires, each day five minutes to midnight, the IP address of the time server according to which the system time is set. The example uses the function *GetTime* from the SysLib library.

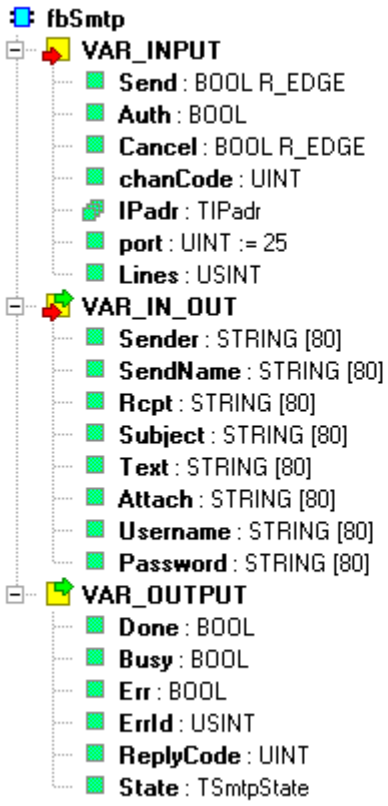
```
VAR_GLOBAL
  NtpName : STRING := 'cz.pool.ntp.org';
  NtpIP : TIPadr;
END_VAR

PROGRAM prgExampleSntp
  VAR_INPUT
  END_VAR
  VAR
    NsLookUp : fbNsLookUp;
    Sntp     : fbSntp;
    now      : TIME;
  END_VAR
  VAR_OUTPUT
  END_VAR
  VAR_TEMP
  END_VAR

  now := GetTime();

  NsLookUp(getIP := now > T#23:55:00.0, chanCode := ETH1_uni0,
           DnsIP :=  STRING_TO_IPADR('208.67.222.222'),
           Name := NtpName,
           IP := NtpIP);

  Sntp(Get := NsLookUp.Done, Accept := Sntp.Done, chanCode := ETH1_uni1,
      IPadr := NtpIP, UtcOff := 60);

END_PROGRAM
```

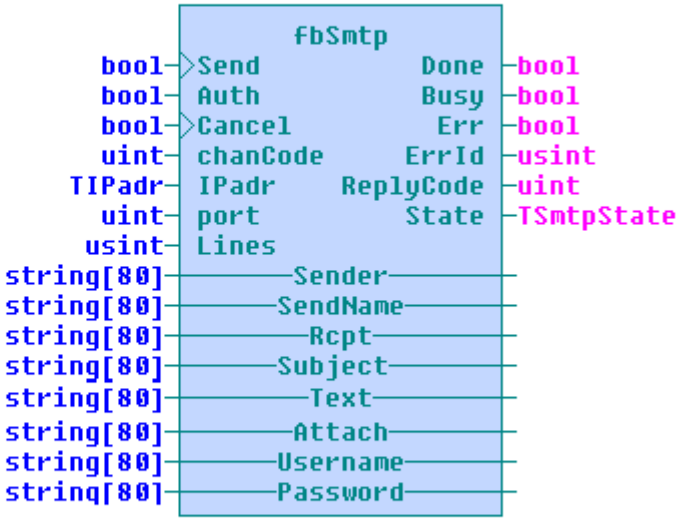
6 ELECTRONIC MAIL OPERATIONS

The library offers the block for sending an electronic mail via the SMTP protocol. Names of SMTP servers are published by e-mail services providers.

6.1 Function block fbSmtp

library: *InternetLib*





Function block *fbSmtp* is used for sending e-mail messages via SMTP protocol. Message sending is initiated by setting the input *Send* to the value TRUE. Sending is done via the connection on the Ethernet channel in the UNI mode according to the constant on the input *chanCode*. The connection must have following parameters: mode TCP master, length of receiving and sending zone 255 bytes. If the connection is not active or does not have correct zone lengths, the block indicates an error on outputs *Err* by the TRUE value and *ErrId* by the value 255.

Address of the SMTP server is transferred on the input *IPadr* and port where the sever receives requirements is set by the input *port* (default value for protocol SMTP is 25).

On the input *Sender,* the variable with the e-mail address of the sender is awaited, on the input *SendName,* the variable with the sender name that should be displayed to the recipient is awaited and on the input *Rcpt,* the variable with recipients addresses separated by semi-colons is awaited.

The message itself is transferred via variables on the input *Subject* where the message subject is awaited and on the input *Text*. The message body must have a form of a text strings of a standard length (ARRAY [1..*n*] OF STRING) where *n* is a number of message lines. On the input *Text,* the first line of the message body is transferred. The number of lines that will be really sent is stated on the input *Lines*. The value of the input *Lines* can be less or equal to *n*.

The file from the PLC memory card can be attached to the message sent. The file name is transferred via the variable on the input *Attach*. To send an e-mail without the attachement, it is necessary to transmit the variable with an empty string to the input *Attach*.

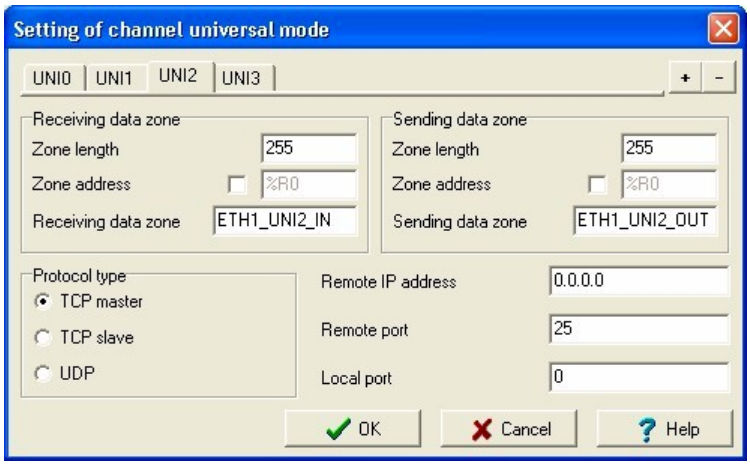
If the server requires authorization using the user name and password, it is vital to set the input *Auth* to the TRUE value and on inputs *UserName* and *Password* transfer variables with the user name and password. If the server does not require the authorization, the variable with an empty string can be transmitted.

During message sending, the output *Busy* is set to the value TRUE. On the output *State,* the communication status with the server is updated (see enumeration *TSmtpState*). Server reply codes are returned on the output *ReplyCode*. The meaning of particular codes is described and explained in detail in the RFC 2821. General principal is that the first number of the reply determines its type in the following way:

- **1**yz – **Preliminary positive reply** – the command was accepted but its execution is postponed. This reply is used only by extended SMTP commands which are not used by the function block
- **2**yz – **Positive reply** – the command was accepted and executed. (e.g. The connection with the server ends with the code 221)
- **3**yz – **Positive immediate reply** – the command was accepted, further informations are expected. (e.g. Replies when the user is verified 334 or during sending of the message body 354)
- **4**yz – **Temporary negative reply** – the command was not accepted, the reason is not permanent, it is possible to try the command again. (Replies of this type usualy indicate that the mail server is busy or does not have enough tools)
- **5**yz – **Permanent negative reply** – the command was not accepted, the reason is permanent, it is not recommended to repeat the request with the same paremeters. (This code is most often in replies when the server did not get the verification that was required or when the user verification fails.)

In case of successful message sending, the output *Done* is set for the period of one cycle.

In case of an error, the output *Err* and ErrId is set where the error code specification is.



Connection setup on the Ethernet channel in the UNI mode for the function block fbSmtp

Variable description :

	Variable	*Type*	*Signification*
VAR_INPUT			
	Send	BOOL R_EDGE	Control variable. Rising edge starts e-mail sending.
	Auth	BOOL	Switch on the function for verification by user name and password
	Cancel	BOOL R_EDGE	Rising edge ceases the sending in progress precociously.
	chanCode	UINT	Connection code ETH1_uni0, ETH1_uni1,...
	IPadr	TIPadr	IP address of the SMTP server
	port	UINT	Port of the time server (default value for the protocol SMTP is 25)
	Lines	USINT	Number of lines of the text to be sent
VAR_IN_OUT			
	Sender	STRING	Sender e-mail address
	SendName	STRING	Sender name displayed to the recipient (can contain only basic characters, no diacritics)
	Rcpt	STRING	Recipients e-mail addresses separated by semi-colons
	Subject	STRING	Message subject
	Text	STRING	First line of the message body
	Attach	STRING	Name of the file to be attached to the e-mail message
	Username	STRING	User name
	Password	STRING	User password
VAR_OUTPUT			
	Done	BOOL	Has the value TRUE at the moment when the e-mail is sent successfully. Otherwise, FALSE is returned.
	Busy	BOOL	Has the value TRUE during the e-mail sending.
	Err	BOOL	Error flag, if the last operation was successful, it has a FALSE value, otherwise, TRUE.
	ErrId	USINT	Error code: *errID* = 0 operation was successful *errID* = 1 server reply time elapsed *errID* = 2 unexpected server reply (for more see ReplyCode) *errID* = 3 the file can not be open, e-mail will be sent without the attachement *errID* = 254 zero address of the SMTP server *errID* = 255 faulty setup of the connection on the Ethernet channel
	ReplyCode	LREAL	Code of the SMTP server reply
	State	TSmtpState	State of the communication with the server (see enumeration TSmtpState)

Following example shows the use of the function block *fbSmtp* for sending an e-mail message. The variable *HeatingIsOn* represents the status of the heating (on/off) that is compared to the last status saved into the local variable *LastHeatingState*. In case of the status change, the query on the DNS server for the IP address of the SMTP server is send and a message is written. The base of the message is defined by the constant *BodyTemplate* where the actual date and temperature of PLC outpus is filled. Format functions from the ToStringLib library are used for the message body modification, actual date and time is obtained using the function *GetDateTime* from the library SysLib.

After the successful request on the DNS server is undertaken, the message is sent.

```
VAR_GLOBAL
  SmtpName     : STRING := 'smtp.seznam.cz';
  SmtpIP       : TIPadr;
  TempOutdoor AT r0_p3_AI0.ENG : REAL;
  TempIndoor  AT r0_p3_AI1.ENG : REAL;
  TempHeating AT r0_p3_AI2.ENG : REAL;
  HeatingIsOn : BOOL;
END_VAR

VAR_GLOBAL CONSTANT
  NumberOfLines : USINT := 5;
END_VAR

TYPE
  TEmailBody : ARRAY [1..NumberOfLines] OF STRING;
END_TYPE

VAR_GLOBAL CONSTANT
  BodyTemplate : TEmailBody := ['Status report %TDD.MM.YYYY$A0hh:mm',
                                'Heating is switched ',
                                'Outdoor temperature is %5.1f°C',
                                'Indoor temperature is %5.1f°C',
                                'Heating temperature is %5.1f°C'];
END_VAR

PROGRAM prgExampleSmtp
  VAR
    NsLookUp          : fbNsLookUp;
    Smtp              : fbSmtp;
    LastHeatingState  : BOOL;
    Sender            : STRING := 'TestPLC@seznam.cz';
    SenderName        : STRING := 'Do not reply';
    UserName          : STRING := 'TestPLC@seznam.cz';
    Password          : STRING := '******';

    Recipient         : STRING := 'notavailable@seznam.cz';
    Subject           : STRING := 'Heating status report';
    Attachement       : STRING;
    Body              : TEmailBody;

  END_VAR

  IF LastHeatingState <> HeatingIsOn THEN
    Body[1] := DT_TO_STRINGF(in := GetDateTime(),
                             format := BodyTemplate[1]);
    IF HeatingIsOn THEN
      Body[2] := CONCAT(BodyTemplate[2], 'on');
    ELSE
      Body[2] := CONCAT(BodyTemplate[2], 'off');
    END_IF;
    Body[3] := REAL_TO_STRINGF(in := TempOutdoor,
                               format := BodyTemplate[3]);
```

```
    Body[4] := REAL_TO_STRINGF(in := TempIndoor,
                               format := BodyTemplate[4]);
    Body[5] := REAL_TO_STRINGF(in := TempHeating,
                               format := BodyTemplate[5]);
  END_IF;

  NsLookUp(getIP := LastHeatingState <> HeatingIsOn,
           chanCode := ETH1_uni0,
           DnsIP :=  STRING_TO_IPADR('208.67.222.222'),
           Name := SmtpName,
           IP := SmtpIP);

  LastHeatingState := HeatingIsOn;

  Smtp(Send := NsLookUp.Done, Auth := true,
       chanCode := ETH1_uni2, IPadr := SmtpIP,
       Lines := NumberOfLines, Sender := Sender,
       SendName := SenderName, Rcpt := Recipient,
       Subject := Subject, Attach := Attachement,
       Username := UserName, Password := Password,
       Text := Body[1]);

END_PROGRAM
```

7 HTTP PROTOCOL COMMUNICATION

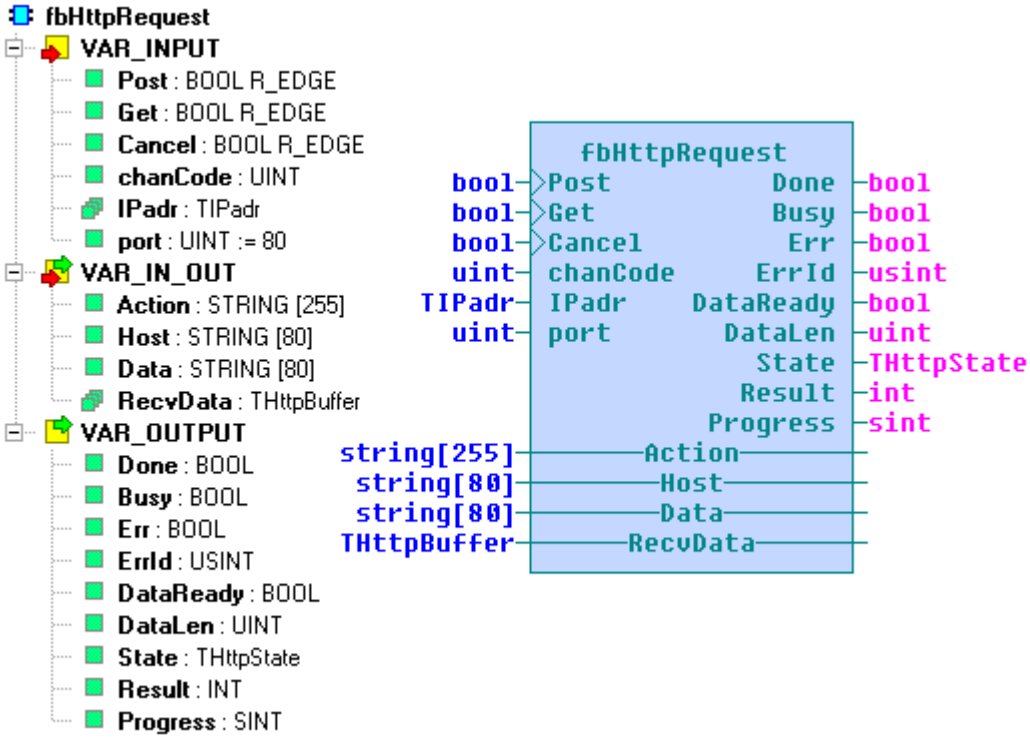
The library offers the function block for communication with the web server via the protocol HTTP. The block implements methods GET and POST from the method file HTTP.

Method GET is used for data acquirement from the web server. Typically, it can be used for IP camera picture acquirement, for downloads of recipes form control server or for data acquirement from public servers (weather forecast etc.).

Method POST is used for data sending to the web server. Typical use is automatic data capturing via the sending to the central sever.

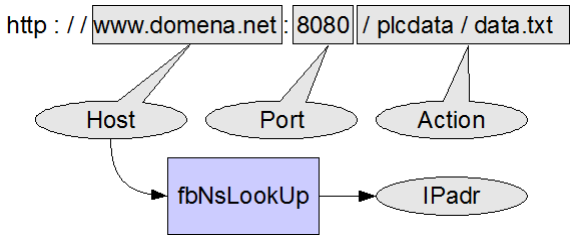
7.1 Function block fbHttpRequest

library: *InternetLib*



Function block *fbHttpRequest* is used for communication with the web server via the protocol HTTP 1.0. The block implements methods GET and POST from the method file HTTP. Communication runs via the connection on the Ethernet channel in the UNI mode according the the constant on the input *chanCode*. The connection must have following parameters: mode TCP master, length of receiving and sending zone 255 bytes. If the connection is not active or it does not have correct zone lengths, the block indicates an error on outputs *Err* by the value TRUE and *ErrId* by the value 255.

The address of data downloaded is transferred within four outputs. On the output *Ipadr,* the server address is awaited (typically gained from the domain server name by the block *fbNsLookUp* or *fbNsLookUpByTable*), on the output *Port* the port number is transmitted where server attends to (default value for HTTP protocol is 80). On the input *Host,* the variable with the domain server name is awaited and on the input *Action* , the variable with the path to the sever data is awaited (the path always starts with a slash character). On the picture bellow it is indicated how data on the address line of the web browser relates to values transmitted on individual inputs.



Data transfer from the address line of the web browsed onto inputs of the function block (Port does not have to be stated, in such case the port has a default value 80)

Communication is according the the method selected initiated by setting the status *Get* or *Post*. Method *Post* expects compared to the method *Get* extra data in the variable transferred onto input *Data*. For easy elaboration on the server side, the variable on the input *Data* should have the following format:

ValueName1	=	Value1	&	ValueName2	=	Value_2	&	...	&		ValueName N	=	Value N

E.G.: `temp1=20.4&state=1&error=0`

Strings in variables on inputs *Action* and *Data* must be in the format URI (Uniform Resource Identifier) according to the RFC 2396. It applies generally that these strings can contain only numbers and characters without diacritics, other symbols including spaces should be coded in the form % followed by two hexadecimal numbers which represents the value of character in the ASCII table (e.g. „%20" is an alternative code for a space).

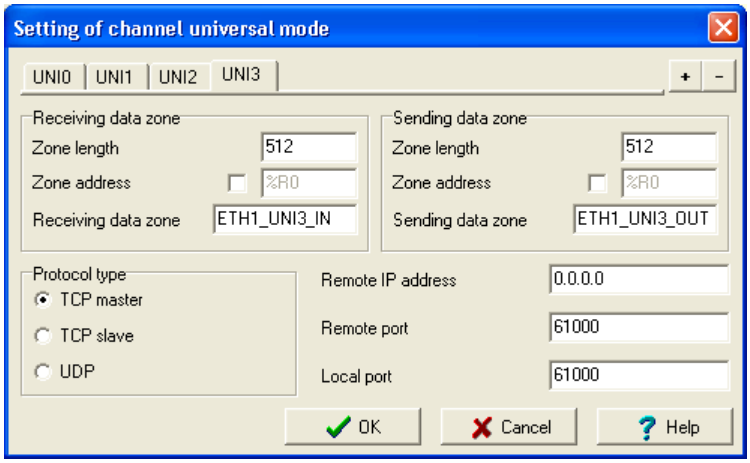
During communication the output *Busy* is set to the value TRUE. In case of a successful cessation is set for one cycle the output *Done*. In case of failure, the output *Err* and *ErrId* is set which contains specific error number.

The output *State* indicates actual communication status. When the message head is uploaded from the server, the output *Result* with the status code is set (see the table bellow) and when the length of next data is accessible, the process of download in per cents (0 up to 100) is returned on the output *Progress*. In all other statuses or when the length is not known the *Progress* returns the value -1.

Signification of the most frequent status codes on the output *Result*

code	**signification**
200	OK – data found
302	Found – data transferred
403	Forbidden – access denied
500	Internal Server Error – internal server error
For other codes see the RFC 2616	

Data from the server comes in consecutive blocks. Each cycle can be returned for one block. The presence of new data is indicated by the value TRUE on the output *DataReady*. The data block- is returned to the variable on the input *RecvData* and its length is indicated on the output *DataLen*.



Connection setup on the Ethernet channel in the UNI mode for the function block fbHttpRequest

Variable description :

	Variable	*Type*	*Signification*
VAR_INPUT			
	Post	BOOL R_EDGE	Rising edge initiates communication using the method POST
	Get	BOOL R_EDGE	Rising edge initiates communication using the method GET
	Cancel	BOOL R_EDGE	Rising edge interrupts running communication
	chanCode	UINT	Connection code ETH1_uni0, ETH1_uni1,...
	IPadr	TIPadr	IP address of the web server
	port	UINT	Port of the web server (default value for the protocol HTTP is 80)
VAR_IN_OUT			
	Action	STRING	Path the the server data (always starts with /)
	Host	STRING	Domain server name
	Data	STRING	Data for the method POST
	RecvData	THttpBuffer	Block of data received
VAR_OUTPUT			
	Done	BOOL	Has the value TRUE at the moment when the communication with the server is ceased successfully. Otherwise,FALSE is returned.
	Busy	BOOL	Has the value TRUE during the communication with the server
	Err	BOOL	Error flag If the last operation was successful, it has the value FALSE, otherwise, TRUE.
	ErrId	USINT	Error code: *errID* = 0 operation was successful *errID* = 1 server reply time elapsed *errID* = 2 all server data was not obtained *errID* = 254 zero address of the web server *errID* = 255 fault connection setup on the Ethernet channel
	DataReady	BOOL	The value TRUE indicates a new block on the input *RecvData*
	DataLen	UINT	Lenght of block data received
	State	THttpState	Status of the communication with the server (see enumeration THttpState)
	Result	INT	Status code returned by the server
	Progress	SINT	Indicates the progress 0 up to 100% during the data dawnload from the server. Otherwise, returns -1

Following example shows the use of the function block *fbHttpRequest* for download of the web camera picture. The example uses the function block *WriteToFileSeq* from the libraty *FileLib* for saving data received onto the memory card. The picture download is initiated by setting the variable *GetPicture* and is conditioned by a successful creation of the path for file saving which is indicated by the variable *PathOk*. The format funciton from the library *ToStringLib* is used for creation file names with incremental index.

```
VAR_GLOBAL CONSTANT
  PathTemplate     : STRING := 'WWW/PICT/';
  FileNameTemplate : STRING := PathTemplate + 'PICT%04d.JPG';
END_VAR

VAR_GLOBAL
  HttpIP       : TIPadr;
  HttpName     : STRING := 'posta.mukolin.cz';
  Action       : STRING := '/axis-cgi/jpg/image.cgi?resolution=CIF';
  Path         : STRING := PathTemplate;
  FileName     : STRING;

  GetPicture   : BOOL;
  PathOk       : BOOL;
  PictIndx     : INT;
END_VAR

PROGRAM prgExampleHttpGet
  VAR
    NsLookUp    : fbNsLookUp;
    HttpRequest : fbHttpRequest;
    WriteToFile : WriteToFileSeq;
    CPath       : CreatePath;
    Empty       : STRING[1];
    Data        : THttpBuffer;
  END_VAR

  CPath(exec := NOT PathOk, fileName := Path, done => PathOk);

  NsLookUp(getIP := PathOk AND GetPicture, chanCode := ETH1_uni0,
           DnsIP :=  STRING_TO_IPADR('208.67.222.222'),
           Name := HttpName,
           IP := HttpIP);

  HttpRequest(Get := NsLookUp.Done, chanCode := ETH1_uni3,
              IPadr := HttpIP,
              Action := Action,
              Host := HttpName,
              Data := Empty, RecvData := Data);

  FileName := INT_TO_STRINGF(in := PictIndx,
                             format := FileNameTemplate);

  WriteToFile(fileName := FileName,
              srcVar := void(Data),
              write := HttpRequest.Result = 200 & HttpRequest.DataReady,
              close := HttpRequest.Done OR HttpRequest.Err,
              size := UINT_TO_UDINT(HttpRequest.DataLen));

  IF HttpRequest.Done THEN
     PictIndx := PictIndx + 1;
  END_IF;
END_PROGRAM
```

The second example shows the use of the function block *fbHttpRequest* for sending data to the database on the web server using the method POST. The variable *HeatIsOn* represents the heating status (on/off) that is compared with the last status saved into the local variable *LastHeatState*. In case of the status change, the request on the DNS server for the IP address of the web server is sent and the string with data is created. After data are sent, the string „OK“ is searched in the reply which returns the script on the server, shown bellow, in case of successful data saving. The function *memcpy* from the library *SysLib* is used for copying data received from the buffer into the string.

```
VAR_GLOBAL
  TempOut     AT r0_p3_AI0.ENG : REAL;
  TempIn      AT r0_p3_AI1.ENG : REAL;
  TempHeat    AT r0_p3_AI2.ENG : REAL;
  HeatIsOn                     : BOOL;

  HttpPostIP     : TIPadr;
  HttpPostName   : STRING := 'foxtrot.howto.cz';
  HttpPostAction : STRING := '/index.php';
END_VAR

PROGRAM prgExampleHttpPost
  VAR
    NsLookUp      : fbNsLookUp;
    HttpRequest   : fbHttpRequest;
    DataIn        : THttpBuffer;
    DataInString  : STRING;
    DataOut       : STRING;
    LastHeatState : BOOL;
    PostSuccesful : BOOL;
  END_VAR

  NsLookUp.getIP := LastHeatState <> HeatIsOn;
  LastHeatState := HeatIsOn;

  IF NsLookUp.getIP THEN
    PostSuccesful := false;
  END_IF;

  NsLookUp(chanCode := ETH1_uni0,
           DnsIP :=  STRING_TO_IPADR('208.67.222.222'),
           Name := HttpPostName,
           IP := HttpPostIP);
  DataOut := 'Heat=' + BOOL_TO_STRING(HeatIsOn) +
             '&TempOut=' + REAL_TO_STRING(TempOut) +
             '&TempIn=' + REAL_TO_STRING(TempIn) +
             '&TempHeat=' + REAL_TO_STRING(TempHeat);
  HttpRequest(Post   := NsLookUp.Done, chanCode := ETH1_uni3,
              IPadr  := HttpPostIP,
              Action := HttpPostAction,
              Host   := HttpPostName,
              Data   := DataOut, RecvData := DataIn);
  IF HttpRequest.DataReady THEN
    Memcpy(length := min(80, HttpRequest.DataLen),
           source := VOID(DataIn),
           dest := VOID(DataInString));
    IF FIND(IN1 := DataInString, IN2 := 'OK') > 0 THEN
      PostSuccesful := true;
    END_IF;
  END_IF;
END_PROGRAM
```

The following PHP script on the server side saves data sent by the method POST into the SQL database. Apart from this function the script in addition generates as a reaction to the method GET the overview table with all data recorded during the day. The file is saved on the server in the file *index.php* which the example above refers to. Variables *$db_server*, *$db_name*, *$db_user*, *$db_pass* contain information for connection to the sever with the SQL database. Data sent from the PLC are approached via the global variable *$POST* where the value name is used as an index.

```
<?php
  $db_server = "mysql.ic.cz";
  $db_name   = "ht_foxtrot";
  $db_user   = "ht_foxtrot";
  $db_pass   = "**********";

  $link = mysql_connect($db_server, $db_user, $db_pass)
          or die("ERR - " . mysql_error());
  mysql_select_db($db_name) or die("ERR - unable to select database");

  if(!empty($_POST)) {
    header("Content-type: text/plain");

    $query = "INSERT INTO plc_data VALUES ('".date("Y-m-d-H:i:s").
             "', '".$_POST['Heat']."', '".$_POST['TempOut']."', '".
             $_POST['TempIn']."', '".$_POST['TempHeat']."');";
    $result = mysql_query($query) or die("ERR - " . mysql_error());

    echo "OK";
  } else {

    echo "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">";
    echo "<html><head><title>PLC data</title></head><body><center>";

    $query = "SELECT * FROM plc_data WHERE datetime LIKE '".date("Y-m-d")."%';";
    $result = mysql_query($query) or die("ERR - " . mysql_error());

    print "<br><br><table border='1' cellpadding='2' cellspacing='1'>";
    print "<tr><th>Time stamp</th><th>State</th><th>Outdoor temperature</th>".
          "<th>Indoor temperature</th><th>Heating temperature</th></tr>";
    while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
      print "<tr><td>".$line['datetime']."</td><td>".$line['Heat'].
            "</td><td>".number_format($line['TempOut'],1)."</td><td>".
            number_format($line['TempIn'],1)."</td><td>".
            number_format($line['TempHeat'],1)."</td></tr>";
    }
    print "</table></center>";
    echo "</body></html>";
  }
  mysql_close($link);
?>
```

Table in the database used by the PHP script was saved by the following SQL command:

```
CREATE TABLE `plc_data` (
  `datetime` timestamp NOT NULL default CURRENT_TIMESTAMP,
  `Heat` tinyint(1) NOT NULL, `TempOut` double NOT NULL,
  `TempIn` double NOT NULL, `TempHeat` double NOT NULL,
  UNIQUE KEY `datetime` (`datetime`)
);
```

For more information please contact:
Teco a. s. Havlíčkova 260, 280 58 Kolín 4, Czech Republic
tel.: +420 321 737 611, fax: +420 321 737 633, teco@tecomat.cz, www.tecomat.com

TXV 003 54.02