```
VAR_GLOBAL
FromSl : ARRAY[ 0..15 ] OF DATAFROMSLAVE; (* Data from slaves *)
ToSl : ARRAY[ 0..15 ] OF DATATOSLAVE; (* Data to slaves *)
END_VAR
```

PROGRAM MasterModbus

```
VAR
Fp : ARRAY[ 0..1 ] OF FILEP; (* File pointer *)
ABf : BOOL; (* Auxiliary buffer *)
Sm : SYSSERIALMODE; (* Serial mode *)
MMdb : ModbusMaster; (* Modbus master communication *)
MIDx : USINT; (* Modbus index *)
SktLsn : SysSktListen; (* FB Socket listen *)
SData : STRING[ 260 ]; (* Socket data *)
DExch : ModbusTCPGateway; (* Data stream exchange *)
MdbEr : ARRAY[ 0..15 ] OF BOOL; (* Modbus error *)
CaseNr : USINT; (* Case gestione *)
NErCheck : USINT; (* Node error check *)
END_VAR
```

```
(* ********************************************************************** *)
(* PROGRAM "MasterModbus"                                                 *)
(* ********************************************************************** *)
(* Questo programma gestisce la comunicazione modbus con i vari slaves.   *)
(* ---------------------------------------------------------------------- *)
(* Eseguo inizializzazioni. *)

    IF (SysFirstLoop) THEN

        CaseNr:=0; (* Case gestione *)

        (* Apro ed inizializzo porta seriale. *)

        Fp[0]:=Sysfopen('COM0', 'rw'); (* File pointer *)
        ABf:=SysGetSerialMode(ADR(Sm), Fp[0]); (* Auxiliary buffer *)
        Sm.Baudrate:=115200; (* Baud rate *)
        Sm.Parity:='E'; (* Parity *)
        Sm.DataBits:=8; (* Data bits *)
        Sm.DTRManagement:=DTR_AUTO_WO_TIMES; (* DTR management *)
        ABf:=SysSetSerialMode(ADR(Sm), Fp[0]); (* Auxiliary buffer *)

        (* Eseguo apertura socket. *)

        Fp[1]:=Sysfopen('TCPSKT', 'rw'); (* File pointer *)

        (* Inizializzo FB comunicazione modbus. *)

        MIDx:=0; (* Modbus index *)
        MMdb.File:=Fp[0]; (* File pointer *)
        MMdb.Type:=0; (* Modbus type (RTU) *)
        MMdb.IFTime:=286; (* Tempo tra frames *)
        MMdb.Timeout:=100; (* Communication timeout *)
        MMdb.Delay:=10; (* Communication delay *)

        (* Inizializzo FB scambio dati tra socket e seriale. *)
        (* Notare il timeout lungo per permettere la programmazione dei *)
        (* nodi slave tramite connessione RS485. *)

        DExch.FpRTU:=Fp[0]; (* File pointer (Stream RTU) *)
        DExch.FpTCP:=Fp[1]; (* File pointer (Stream TCP) *)
        DExch.SpyOn:=TRUE; (* Spy On *)
        DExch.IFTime:=286; (* Tempo tra frames *)
        DExch.Timeout:=1000; (* Communication timeout *)
```

```
 44     END_IF;
 45
 46     (* ---------------------------------------------------------------------- *)
 47     (* GESTIONE CONTROLLO STATO SOCKET TCP                                    *)
 48     (* ---------------------------------------------------------------------- *)
 49     (* Forzo socket in condizione di listening. *)
 50     (* "LifeTm" piccolo, forza chiusura socket su mancanza comunicazione. *)
 51     (* "FlushTm" deve essere più grande del tempo di loop programma. *)
 52
 53     SktLsn.File:=Fp[1]; (* Flusso dati stream *)
 54     SktLsn.MyPort:=2000; (* Porta in ascolto su socket *)
 55     SktLsn.LifeTm:=60; (* Tempo di vita socket (S) *)
 56     SktLsn.FlushTm:=10; (* Tempo di flush socket (mS) *)
 57     SktLsn.RxSize:=TO_UINT(SIZEOF(SData)); (* Dimensione buffer Rx dati socket *)
 58     SktLsn.TxSize:=TO_UINT(SIZEOF(SData)); (* Dimensione buffer Tx dati socket *)
 59     SktLsn(Enable:=TRUE); (* Forzo socket in listening *)
 60
 61     (* ---------------------------------------------------------------------- *)
 62     (* GESTIONE SCAMBIO DATI TRA SOCKET TCP E SERIALE                         *)
 63     (* ---------------------------------------------------------------------- *)
 64     (* Se client è connesso al socket non viene più eseguita la comunicazione *)
 65     (* modbus con i nodi slaves ora i dati arrivano dalla connessione TCP/IP. *)
 66
 67     DExch(Enable:=SktLsn.Connect); (* Data stream exchange *)
 68     IF (SktLsn.Connect) THEN CaseNr:=0; MIDx:=0; RETURN; END_IF;
 69
 70     (* ---------------------------------------------------------------------- *)
 71     (* GESTIONE COMUNICAZIONE MODBUS                                          *)
 72     (* ---------------------------------------------------------------------- *)
 73     (* Gestione comunicazione modbus. *)
 74
 75     MMdb(); (* Modbus master communication *)
 76
 77     (* Su esecuzione lettura passo a gestire nuovo nodo, esco per propagare *)
 78     (* la disabilitazione della FB di comunicazione modbus. *)
 79
 80     IF (MMdb.Done) THEN
 81         MMdb.Enable:=FALSE; (* FB enable *)
 82         CaseNr:=CaseNr+1; (* Case gestione *)
 83         RETURN;
 84     END_IF;
 85
 86     (* Eseguo gestione errore di comunicazione. *)
 87
 88     IF (MMdb.Ok) THEN MdbEr[MMdb.Node-1]:=FALSE; END_IF;
 89     IF (MMdb.Fault) THEN MdbEr[MMdb.Node-1]:=TRUE; END_IF;
 90
 91     (* ---------------------------------------------------------------------- *)
 92     (* GESTIONE NODI MODBUS                                                   *)
 93     (* ---------------------------------------------------------------------- *)
 94     (* Eseguo lettura struttura "Rd" dal nodo slave. *)
 95
 96     CASE (CaseNr) OF
 97
 98         (* ---------------------------------------------------------------- *)
 99         (* Eseguo lettura struttura "DATAFROMSLAVE" dal nodo slave. *)
100         (* Nel sistema slave è definita struttura identica "DATATOMASTER" *)
101         (* che è allocata a DB100.0 quindi l'indirizzo di lettura è 40000. *)
102
103         0:
```

```
104         IF NOT(MMdb.Enable) THEN
105             MMdb.Address:=40000; (* Variable address *)
106             MMdb.Buffer:=ADR(FromSl[MIDx]); (* Buffer address *)
107             MMdb.Points:=SIZEOF(FromSl[0])/2; (* Number of points *)
108             MMdb.Node:=MIDx+1; (* Modbus node *)
109             MMdb.FCode:=16#03; (* Modbus function *)
110             MMdb.Enable:=TRUE; (* FB enable *)
111         END_IF;
112
113         (* ---------------------------------------------------------------- *)
114         (* Eseguo scrittura struttura "DATATOSLAVE" verso nodo slave. *)
115         (* Nel sistema slave è definita struttura identica "DATAFROMMASTER" *)
116         (* che è allocata a DB100.16 quindi l'indirizzo di lettura è 40008. *)
117
118         1:
119         IF NOT(MMdb.Enable) THEN
120             MMdb.Address:=40008; (* Variable address *)
121             MMdb.Buffer:=ADR(ToSl[MIDx]); (* Buffer address *)
122             MMdb.Points:=SIZEOF(ToSl[0])/2; (* Number of points *)
123             MMdb.Node:=MIDx+1; (* Modbus node *)
124             MMdb.FCode:=16#10; (* Modbus function *)
125             MMdb.Enable:=TRUE; (* FB enable *)
126         END_IF;
127
128         (* ---------------------------------------------------------------- *)
129         (* Eseguo incremento nodo PLC slave. *)
130
131         2:
132         MIDx:=MIDx+1;(* Modbus index *)
133         IF (MIDx >= 16) THEN
134             MIDx:=0; (* Modbus index *)
135             NErCheck:=NErCheck+1; (* Node error check *)
136             IF (NErCheck >= 16) THEN NErCheck:=0; END_IF;
137         END_IF;
138
139         (* Controllo se nodo in errore. Per evitare di perdere tempo se *)
140         (* nodo in errore passo a nodo successivo. Controllo un nodo in *)
141         (* errore ogni loop completo di tutti i nodi per verificare se nodo *)
142         (* riprende operatività. *)
143
144         IF (MdbEr[MIDx] AND (MIDx <> NErCheck)) THEN RETURN; END_IF;
145         CaseNr:=0; (* Case gestione *)
146     END_CASE;
147
148 (* [End of file] *)
149
150
```

FUNCTION_BLOCK ModbusMaster

(SFR054B230) Manages the modbus master communication
ENCRYPTED CODE

```
VAR_INPUT
Enable : BOOL;
SpyOn : BOOL; (* Spy active *)
File : FILEP; (* Terminal I/O pointer *)
Type : USINT; (* Modbus type *)
Node : USINT; (* Node number *)
FCode : USINT; (* Function code *)
Address : UINT; (* Start address *)
Points : UDINT; (* Number of points *)
Buffer : @USINT; (* Address of data buffer *)
IFTime : UDINT; (* Interframe time (uS) *)
Timeout : UINT; (* Timeout time (mS) *)
Delay : UINT; (* Delay time (mS) *)
END_VAR

VAR_OUTPUT
Done : BOOL; (* Command done *)
Ok : BOOL := FALSE; (* Execution Ok *)
Fault : BOOL; (* Command fault *)
Errors : UDINT; (* Error counter *)
END_VAR
```

1

	Project : Master	
	FUNCTION BLOCK : ModbusMaster	
	Release : ClimateMng	Ver :1.00
	Author :	Date:13/05/2015
	Note :	Page:1 of 1

FUNCTION_BLOCK ModbusTCPGateway

(eLLabNetworkLib_A000) Modbus TCP gateway
ENCRYPTED CODE

```
VAR_INPUT
Enable : BOOL; (* FB enable *)
SpyOn : BOOL; (* Spy active *)
FpTCP : FILEP; (* File pointer (Modbus TCP) *)
FpRTU : FILEP; (* File pointer (Modbus RTU) *)
IFTime : UDINT; (* Interframe time (uS) *)
Timeout : UDINT; (* Timeout time (mS) *)
END_VAR

VAR_OUTPUT
Enabled : BOOL; (* FB enabled *)
Fault : BOOL; (* FB fault *)
END_VAR
```

1

	Project : Master	
	FUNCTION BLOCK : ModbusTCPGateway	
	Release : ClimateMng	Ver :1.00
	Author :	Date:13/05/2015
	Note :	Page:1 of 1