```
VAR_GLOBAL
Slaves : ARRAY[ 0..1 ] OF MODBUSDATA; (* Modbus slave data *)
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 : DataStreamExch; (* Data stream exchange *)
MdbENr : ARRAY[ 0..1 ] OF UDINT; (* Modbus error number *)
END_VAR
```

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

    IF (SysFirstLoop) THEN

        (* Apro ed inizializzo porta seriale. *)

        Fp[0]:=Sysfopen('COM2', '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.IFTime:=286; (* Tempo tra frames *)
        MMdb.Timeout:=100; (* Communication timeout *)
        MMdb.Delay:=10; (* Communication delay *)

        (* Inizializzo FB scambio dati tra socket e seriale. *)

        DExch.FpA:=Fp[0]; (* File pointer (Stream A) *)
        DExch.FpB:=Fp[1]; (* File pointer (Stream B) *)
        DExch.DBSize:=SIZEOF(SData); (* Data buffer size *)
        DExch.DDelay:=30; (* Data delay *)
    END_IF;

    (* ---------------------------------------------------------------------- *)
    (* GESTIONE CONTROLLO STATO SOCKET TCP                                    *)
    (* ---------------------------------------------------------------------- *)
    (* Forzo socket in condizione di listening. *)
    (* "LifeTm" piccolo, forza chiusura socket su mancanza comunicazione. *)
    (* "FlushTm" deve essere più grande del tempo di loop programma. *)
```

```
46
47      SktLsn.File:=Fp[1]; (* Flusso dati stream *)
48      SktLsn.MyPort:=2000; (* Porta in ascolto su socket *)
49      SktLsn.LifeTm:=60; (* Tempo di vita socket (S) *)
50      SktLsn.FlushTm:=10; (* Tempo di flush socket (mS) *)
51      SktLsn.RxSize:=TO_UINT(SIZEOF(SData)); (* Dimensione buffer Rx dati socket *)
52      SktLsn.TxSize:=TO_UINT(SIZEOF(SData)); (* Dimensione buffer Tx dati socket *)
53      SktLsn(Enable:=TRUE); (* Forzo socket in listening *)
54
55      (* ---------------------------------------------------------------- *)
56      (* GESTIONE SCAMBIO DATI TRA SOCKET TCP E SERIALE                   *)
57      (* ---------------------------------------------------------------- *)
58      (* Se client è connesso al socket non viene più eseguita la comunicazione *)
59      (* modbus con i nodi slaves ora i dati arrivano dalla connessione TCP/IP. *)
60
61      DExch(Enable:=SktLsn.Connect); (* Data stream exchange *)
62      IF (SktLsn.Connect) THEN MIDx:=0; RETURN; END_IF;
63
64      (* ---------------------------------------------------------------- *)
65      (* GESTIONE COMUNICAZIONE MODBUS                                    *)
66      (* ---------------------------------------------------------------- *)
67      (* Gestione comunicazione modbus. *)
68
69      MMdb(); (* Modbus master communication *)
70
71      (* Su esecuzione lettura passo a gestire nuovo nodo, esco per propagare *)
72      (* la disabilitazione della FB di comunicazione modbus. *)
73
74      IF (MMdb.Done) THEN MMdb.Enable:=FALSE; MIDx:=MIDx+1; RETURN; END_IF;
75
76      (* Eseguo conteggio errori di comunicazione. *)
77
78      IF (MMdb.Fault) THEN MdbENr[MMdb.Node]:=MdbENr[MMdb.Node]+1; END_IF;
79
80      (* ---------------------------------------------------------------- *)
81      (* GESTIONE NODI MODBUS                                             *)
82      (* ---------------------------------------------------------------- *)
83      (* Gestione parametrizzazione nodi modbus. *)
84
85      CASE (MIDx) OF
86
87          (* ---------------------------------------------------------- *)
88          (* COMUNICAZIONE CON NODO SLAVE "0"                           *)
89          (* ---------------------------------------------------------- *)
90          (* Eseguo lettura struttura "Rd" dal nodo slave. *)
91
92          0:
93          IF NOT(MMdb.Enable) THEN
94              MMdb.Type:=2; (* Modbus type *)
95              MMdb.Address:=40000; (* Variable address *)
96              MMdb.Buffer:=ADR(Slaves[0].Rd); (* Buffer address *)
97              MMdb.Points:=SIZEOF(Slaves[0].Rd)/2; (* Number of points *)
98              MMdb.Node:=0; (* Modbus node *)
99              MMdb.FCode:=16#04; (* Modbus function *)
100             MMdb.Enable:=TRUE; (* FB enable *)
101         END_IF;
102
103         (* ---------------------------------------------------------- *)
104         (* Eseguo scrittura struttura "Wr" sul nodo slave. *)
105
```

```
106         1:
107         IF NOT(MMdb.Enable) THEN
108             MMdb.Type:=2; (* Modbus type *)
109             MMdb.Address:=TO_UINT(40000+SIZEOF(Slaves[0].Rd)/2); (* Variable address *)
110             MMdb.Buffer:=ADR(Slaves[0].Wr); (* Buffer address *)
111             MMdb.Points:=SIZEOF(Slaves[0].Wr)/2; (* Number of points *)
112             MMdb.Node:=0; (* Modbus node *)
113             MMdb.FCode:=16#10; (* Modbus function *)
114             MMdb.Enable:=TRUE; (* FB enable *)
115         END_IF;
116
117         (* ---------------------------------------------------------------- *)
118         (* COMUNICAZIONE CON NODO SLAVE "1"                                 *)
119         (* ---------------------------------------------------------------- *)
120         (* Eseguo lettura struttura "Rd" dal nodo slave. *)
121
122         2:
123         IF NOT(MMdb.Enable) THEN
124             MMdb.Type:=2; (* Modbus type *)
125             MMdb.Address:=40000; (* Variable address *)
126             MMdb.Buffer:=ADR(Slaves[1].Rd); (* Buffer address *)
127             MMdb.Points:=SIZEOF(Slaves[1].Rd)/2; (* Number of points *)
128             MMdb.Node:=1; (* Modbus node *)
129             MMdb.FCode:=16#04; (* Modbus function *)
130             MMdb.Enable:=TRUE; (* FB enable *)
131         END_IF;
132
133         (* ---------------------------------------------------------------- *)
134         (* Eseguo scrittura struttura "Wr" sul nodo slave. *)
135
136         3:
137         IF NOT(MMdb.Enable) THEN
138             MMdb.Type:=2; (* Modbus type *)
139             MMdb.Address:=TO_UINT(40000+SIZEOF(Slaves[1].Rd)/2); (* Variable address *)
140             MMdb.Buffer:=ADR(Slaves[1].Wr); (* Buffer address *)
141             MMdb.Points:=SIZEOF(Slaves[1].Wr)/2; (* Number of points *)
142             MMdb.Node:=1; (* Modbus node *)
143             MMdb.FCode:=16#10; (* Modbus function *)
144             MMdb.Enable:=TRUE; (* FB enable *)
145         END_IF;
146
147         (* ---------------------------------------------------------------- *)
148         (* Fine gestione nodi, reinizializzo. *)
149
150     ELSE
151         MIDx:=0; (* Modbus index *)
152     END_CASE;
153
154 (* [End of file] *)
155
156
```

FUNCTION_BLOCK DataStreamExch

(SFR054B230) Exchanges data between two I/O streams
ENCRYPTED CODE

```
VAR_INPUT
Enable : BOOL; (* FB enable *)
FpA : FILEP; (* File pointer (Stream A) *)
FpB : FILEP; (* File pointer (Stream B) *)
DBSize : UDINT; (* Data buffer size *)
DDelay : UDINT; (* Data delay *)
END_VAR

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

1

	Project : RS485Network	
	FUNCTION BLOCK : DataStreamExch	
	Release : ClimateMng	Ver :1.00
	Author :	Date:04/01/2014
	Note :	Page:1 of 1

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 : RS485Network	
	FUNCTION BLOCK : ModbusMaster	
	Release : ClimateMng	Ver :1.00
	Author :	Date:04/01/2014
	Note :	Page:1 of 1