PROGRAM SDMInterface

```
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 : UINT; (* Modbus error number *)
MdbCfg : ARRAY[ 0..9 ] OF MODBUSCONFIG; (* Modbus configuration *)
MdbValue : UDINT; (* Valore per modbus *)
SDM120 : SDM120REGISTERS; (* Valore registri SDM120 *)
END_VAR
```

```
(* ************************************************************************ *)
(* PROGRAM "SDMInterface"                                                   *)
(* ************************************************************************ *)
(* Questo programma gestisce la comunicazione modbus con il modulo SDM.     *)
(* ------------------------------------------------------------------------ *)
(* Eseguo inizializzazioni. *)

    IF (SysFirstLoop) THEN

        (* ---------------------------------------------------------------- *)
        (* CONFIGURAZIONE REGISTRI ANALIZZATORE ENERGIA                      *)
        (* ---------------------------------------------------------------- *)
        (* Lettura "Voltage". *)

        MdbCfg[0].Node:=1; (* Modbus node *)
        MdbCfg[0].FCode:=16#04; (* Modbus function *)
        MdbCfg[0].Address:=1; (* Register address *)
        MdbCfg[0].Buffer:=ADR(SDM120.Voltage); (* Buffer address *)

        (* Lettura "Current". *)

        MdbCfg[1].Node:=1; (* Modbus node *)
        MdbCfg[1].FCode:=16#04; (* Modbus function *)
        MdbCfg[1].Address:=17; (* Register address *)
        MdbCfg[1].Buffer:=ADR(SDM120.Current); (* Buffer address *)

        (* Lettura "Active power". *)

        MdbCfg[2].Node:=1; (* Modbus node *)
        MdbCfg[2].FCode:=16#04; (* Modbus function *)
        MdbCfg[2].Address:=13; (* Register address *)
        MdbCfg[2].Buffer:=ADR(SDM120.ActivePwr); (* Buffer address *)

        (* Lettura "Apparent power". *)

        MdbCfg[3].Node:=1; (* Modbus node *)
        MdbCfg[3].FCode:=16#04; (* Modbus function *)
        MdbCfg[3].Address:=19; (* Register address *)
        MdbCfg[3].Buffer:=ADR(SDM120.ApparentPwr); (* Buffer address *)

        (* Lettura "Reactive power". *)
```

```
43          MdbCfg[4].Node:=1; (* Modbus node *)
44          MdbCfg[4].FCode:=16#04; (* Modbus function *)
45          MdbCfg[4].Address:=25; (* Register address *)
46          MdbCfg[4].Buffer:=ADR(SDM120.ReactivePwr); (* Buffer address *)
47
48          (* Lettura "Power factor". *)
49
50          MdbCfg[5].Node:=1; (* Modbus node *)
51          MdbCfg[5].FCode:=16#04; (* Modbus function *)
52          MdbCfg[5].Address:=31; (* Register address *)
53          MdbCfg[5].Buffer:=ADR(SDM120.PwrFactor); (* Buffer address *)
54
55          (* Lettura "Frequency". *)
56
57          MdbCfg[6].Node:=1; (* Modbus node *)
58          MdbCfg[6].FCode:=16#04; (* Modbus function *)
59          MdbCfg[6].Address:=71; (* Register address *)
60          MdbCfg[6].Buffer:=ADR(SDM120.Frequency); (* Buffer address *)
61
62          (* Lettura "Import active energy". *)
63
64          MdbCfg[7].Node:=1; (* Modbus node *)
65          MdbCfg[7].FCode:=16#04; (* Modbus function *)
66          MdbCfg[7].Address:=73; (* Register address *)
67          MdbCfg[7].Buffer:=ADR(SDM120.IActiveEn); (* Buffer address *)
68
69          (* Lettura "Export active energy". *)
70
71          MdbCfg[8].Node:=1; (* Modbus node *)
72          MdbCfg[8].FCode:=16#04; (* Modbus function *)
73          MdbCfg[8].Address:=75; (* Register address *)
74          MdbCfg[8].Buffer:=ADR(SDM120.EActiveEn); (* Buffer address *)
75
76          (* Lettura "Total active energy". *)
77
78          MdbCfg[9].Node:=1; (* Modbus node *)
79          MdbCfg[9].FCode:=16#04; (* Modbus function *)
80          MdbCfg[9].Address:=343; (* Register address *)
81          MdbCfg[9].Buffer:=ADR(SDM120.TActiveEn); (* Buffer address *)
82
83          (* ------------------------------------------------------------------ *)
84          (* APERTURA STREAMS DI COMUNICAZIONE                                  *)
85          (* ------------------------------------------------------------------ *)
86          (* Apro ed inizializzo porta seriale. *)
87
88          Fp[0]:=Sysfopen('COM0', 'rw'); (* File pointer *)
89          ABf:=SysGetSerialMode(ADR(Sm), Fp[0]); (* Auxiliary buffer *)
90          Sm.Baudrate:=2400; (* Baud rate *)
91          Sm.Parity:='N'; (* Parity *)
92          Sm.DataBits:=8; (* Data bits *)
93          Sm.DTRManagement:=DTR_AUTO_WO_TIMES; (* DTR management *)
94          ABf:=SysSetSerialMode(ADR(Sm), Fp[0]); (* Auxiliary buffer *)
95
96          (* Eseguo apertura socket. *)
97
98          Fp[1]:=Sysfopen('TCPSKT', 'rw'); (* File pointer *)
99
100         (* ------------------------------------------------------------------ *)
101         (* PARAMETRIZZAZIONE FBs                                              *)
102         (* ------------------------------------------------------------------ *)
```

```
103         (* Inizializzo FB comunicazione modbus. *)
104         (* Tutti i comandi operano su doppio registro (16 bits). *)
105
106         MIDx:=0; (* Modbus index *)
107         MMdb.Type:=0; (* Modbus type *)
108         MMdb.File:=Fp[0]; (* File pointer *)
109         MMdb.Buffer:=ADR(MdbValue); (* Buffer address *)
110         MMdb.Points:=2; (* Number of points *)
111         MMdb.IFTime:=14000; (* Tempo tra frames *)
112         MMdb.Timeout:=1000; (* Communication timeout *)
113         MMdb.Delay:=10; (* Communication delay *)
114
115         (* Inizializzo FB scambio dati tra socket e seriale. *)
116
117         DExch.FpA:=Fp[0]; (* File pointer (Stream A) *)
118         DExch.FpB:=Fp[1]; (* File pointer (Stream B) *)
119         DExch.DBSize:=SIZEOF(SData); (* Data buffer size *)
120         DExch.DDelay:=30; (* Data delay *)
121     END_IF;
122
123     (* ------------------------------------------------------------------- *)
124     (* GESTIONE CONTROLLO STATO SOCKET TCP                                 *)
125     (* ------------------------------------------------------------------- *)
126     (* Forzo socket in condizione di listening. *)
127     (* "LifeTm" piccolo, forza chiusura socket su mancanza comunicazione. *)
128     (* "FlushTm" deve essere più grande del tempo di loop programma. *)
129
130     SktLsn.File:=Fp[1]; (* Flusso dati stream *)
131     SktLsn.MyPort:=2000; (* Porta in ascolto su socket *)
132     SktLsn.LifeTm:=60; (* Tempo di vita socket (S) *)
133     SktLsn.FlushTm:=10; (* Tempo di flush socket (mS) *)
134     SktLsn.RxSize:=TO_UINT(SIZEOF(SData)); (* Dimensione buffer Rx dati socket *)
135     SktLsn.TxSize:=TO_UINT(SIZEOF(SData)); (* Dimensione buffer Tx dati socket *)
136     SktLsn(Enable:=TRUE); (* Forzo socket in listening *)
137
138     (* ------------------------------------------------------------------- *)
139     (* GESTIONE SCAMBIO DATI TRA SOCKET TCP E SERIALE                      *)
140     (* ------------------------------------------------------------------- *)
141     (* Se client è connesso al socket non viene più eseguita la comunicazione *)
142     (* modbus con i nodi slaves ora i dati arrivano dalla connessione TCP/IP. *)
143
144     DExch(Enable:=SktLsn.Connect); (* Data stream exchange *)
145     IF (SktLsn.Connect) THEN MIDx:=0; RETURN; END_IF;
146
147     (* ------------------------------------------------------------------- *)
148     (* GESTIONE COMUNICAZIONE MODBUS                                       *)
149     (* ------------------------------------------------------------------- *)
150     (* Eseguo controllo e conteggio errori di comunicazione. *)
151
152     IF (MMdb.Fault) THEN MdbENr:=MdbENr+1; END_IF;
153
154     (* Fine comando lettura eseguo swap valore letto. *)
155
156     IF (MMdb.Ok) THEN MdbCfg[MIDx].@Buffer:=ROL(MdbValue, 16); END_IF;
157
158     (* Fine esecuzione comando Modbus passo a gestire nuovo nodo. *)
159     (* Eseguo incremento indice tabella configurazione Modbus. *)
160
161     IF (MMdb.Done) THEN
162         MIDx:=MIDx+1; (* Modbus index *)
```

```
163         IF (MIDx >= (SIZEOF(MdbCfg)/SIZEOF(MdbCfg[0]))) THEN MIDx:=0; END_IF;
164
165         (* Eseguo reset abilitazione FB per eseguire nuovo comando. *)
166
167         MMdb.Enable:=FALSE; (* FB enable *)
168     END_IF;
169
170     (* Gestione comunicazione modbus. *)
171
172     MMdb(); (* Modbus master communication *)
173
174     (* Eseguo parametrizzazione FB modbus. *)
175
176     IF NOT(MMdb.Enable) THEN
177         MMdb.Enable:=TRUE; (* FB enable *)
178         MMdb.Node:=MdbCfg[MIDx].Node; (* Modbus node *)
179         MMdb.FCode:=MdbCfg[MIDx].FCode; (* Modbus function *)
180         MMdb.Address:=MdbCfg[MIDx].Address; (* Variable address *)
181     END_IF;
182
183 (* [End of file] *)
184
185
```

FUNCTION_BLOCK SetSMode

(ePLCUtyLib_C000) Set the serial communication mode
ENCRYPTED CODE

```
VAR_INPUT
Fp : FILEP;
Baudrate : UDINT; (* Baudrate *)
Parity : STRING[ 1 ]; (* Parity type *)
DataBits : USINT; (* Nr of data bits *)
StopBits : USINT; (* Nr of stop bits *)
DTRManagement : USINT; (* DTR management type *)
DTRComplement : BOOL; (* Complement the DTR signal *)
EchoFlush : BOOL; (* Flush the echo *)
DTROnTime : UINT; (* DTR On wait time *)
DTROffTime : UINT; (* DTR Off wait time *)
END_VAR

VAR_OUTPUT
Done : BOOL; (* Execution done *)
Fault : BOOL; (* Execution fault *)
END_VAR
```

1

	Project : SDM120AllRegisters	
	FUNCTION BLOCK : SetSMode	
	Release : SDM120	Ver :1.00
	Author :	Date:25/06/2015
	Note :	Page:1 of 1

FUNCTION_BLOCK ModbusMaster

(ePLCUtyLib_C000) Manages the modbus master communication
ENCRYPTED CODE

```
VAR_INPUT
Enable : BOOL; (* FB enable *)
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 : SDM120AllRegisters	
	FUNCTION BLOCK : ModbusMaster	
	Release : SDM120	Ver :1.00
	Author :	Date:25/06/2015
	Note :	Page:1 of 1

FUNCTION_BLOCK DataStreamExch

(ePLCUtyLib_C000) 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 : SDM120AllRegisters	
	FUNCTION BLOCK : DataStreamExch	
	Release : SDM120	Ver :1.00
	Author :	Date:25/06/2015
	Note :	Page:1 of 1