```
VAR_GLOBAL
WiegandD0 AT %IX255.0 : BOOL; (* Wiegand (D0) *)
WiegandD1 AT %IX255.1 : BOOL; (* Wiegand (D1) *)
DoorUlck AT %QX255.0 : BOOL; (* Comando apertura porta *)
Do01CPU AT %QX255.1 : BOOL; (* Do01 modulo CPU *)
LInputs AT %MW100.16 : UINT; (* Logic inputs (Web page) *)
DWiegand : DWORD; (* Wiegand data *)
BWiegand : USINT; (* Wiegand bits *)
OkWiegand : BOOL; (* Wiegand data Ok *)
LOutputs AT %MW100.18 : UINT; (* Logic outputs (Web page) *)
TAGID AT %MD100.32 : UDINT; (* TAG ID (Web page) *)
TAGs AT %MX100.2304 : ARRAY[ 0..15 ] OF TAGSETTINGS; (* TAG settings *)
WrongTCtr AT %MX100.20 : USINT; (* Wrong TAG counter (Web page) *)
Log : LogWrite; (* Log write instance *)
LockTime AT %MX100.21 : USINT; (* LockTime (Web page) *)
END_VAR
```

```
VAR
CaseNr : USINT; (* Program case *)
WDecoder : Wiegand26Dcd; (* Wiegand decoder *)
WKeyboard : WiegandKbd; (* Wiegand keyboard *)
TAGIDx : USINT; (* TAG index *)
TimeBf : UDINT; (* Time buffer (uS) *)
i : INT; (* Auxiliary counter *)
TAGRd : BOOL; (* TAG readed *)
END_VAR
```

```
(* ********************************************************************** *)
(* PROGRAM "Doorkeeper"                                                   *)
(* ********************************************************************** *)
(* Questo programma gestisce il comando di apertura di una porta in base ad   *)
(* un codice impostato da tastiera o letto da TAG RFID.                   *)
(* ---------------------------------------------------------------------- *)

    (* ---------------------------------------------------------------------- *)
    (* INIZIALIZZAZIONI                                                       *)
    (* ---------------------------------------------------------------------- *)
    (* Eseguo attesa spionaggio libero. *)

    IF NOT(SysSpyData(0, 0, NULL, NULL))THEN RETURN; END_IF;

    (* ---------------------------------------------------------------------- *)
    (* INIZIALIZZAZIONI                                                       *)
    (* ---------------------------------------------------------------------- *)
    (* Attesa codice Wiegand da RFID. *)

    IF (OkWiegand) THEN
        OkWiegand:=FALSE; (* Wiegand data Ok *)

        (* Non controllo lunghezza codice Wiegand, in questo modo se arriva *)
        (* codice diverso da tasto abortisco eventuale inputazione in corso. *)

        WKeyboard(Enable:=TRUE, WBits:=DWiegand); (* Gestione tastiera *)
        IF (WKeyboard.EKeyPsd) THEN
            TAGRd:=TRUE; (* TAG readed *)
            TAGID:=WKeyboard.Value; (* TAG ID to web page *)

            i:=SysVarsnprintf(ADR(Log.Text), SIZEOF(Log.Text), 'Codice da tastiera:%d', UDINT_TYPE, ADR
(TAGID));
            i:=SysSpyData(SPY_ASCII, 16#00000001, ADR('Lg'), ADR(Log.Text));
            Log(); (* Salvo log *)
        END_IF;

        (* Se codice Wiegand a 26 bits acquisisco ID TAG. *)

        IF (BWiegand = 26) THEN
            WDecoder(Enable:=TRUE, WBits:=DWiegand); (* Decodifica codice TAG *)
            IF (WDecoder.CodeOk) THEN
                TAGRd:=TRUE; (* TAG readed *)
                TAGID:=TO_UDINT(WDecoder.Facility)*16#10000; (* TAG ID to web page *)
                TAGID:=TAGID+WDecoder.IDNumber; (* TAG ID to web page *)

                i:=SysVarsnprintf(ADR(Log.Text), SIZEOF(Log.Text), 'Acquisito TAG:%d', UDINT_TYPE, ADR(
TAGID));
```

```
46                 i:=SysSpyData(SPY_ASCII, 16#00000001, ADR('Lg'), ADR(Log.Text));
47                 Log(); (* Salvo log *)
48             END_IF;
49         END_IF;
50     END_IF;
51
52     (* ---------------------------------------------------------------------- *)
53     (* GESTIONE CASES PROGRAMMA                                               *)
54     (* ---------------------------------------------------------------------- *)
55     (* Cases di gestione programma. *)
56
57     CASE (CaseNr) OF
58
59         (* ---------------------------------------------------------------- *)
60         (* Attesa acquiszione TAG ID e ricerca codice in memoria. *)
61
62         0:
63         IF NOT(TAGRd) THEN RETURN; END_IF;
64         TAGRd:=FALSE; (* TAG readed *)
65         FOR TAGIDx:=0 TO (SIZEOF(TAGs)/SIZEOF(TAGs[0])) DO
66             IF (TAGs[TAGIDx].ID = TAGID) THEN CaseNr:=10; RETURN; END_IF;
67         END_FOR;
68
69         (* TAG non trovato incremento contatore errori. *)
70
71         WrongTCtr:=WrongTCtr+1; (* Wrong TAG counter *)
72         i:=SysVarsnprintf(ADR(Log.Text), SIZEOF(Log.Text), 'Errore TAG Nr:%d', USINT_TYPE, ADR(WrongTCt
   r));
73         i:=SysSpyData(SPY_ASCII, 16#00000001, ADR('Lg'), ADR(Log.Text));
74         Log(); (* Salvo log *)
75
76         (* Eseguo controllo se raggiunto limite errori consecutivi. *)
77
78         IF (WrongTCtr < 3) THEN RETURN; END_IF;
79         CaseNr:=CaseNr+1; (* Program case *)
80
81         (* ---------------------------------------------------------------- *)
82         (* Blocco apertura per troppi errori. *)
83
84         1:
85         Log(Text:='Blocco apertura'); (* Salvo log *)
86         i:=SysSpyData(SPY_ASCII, 16#00000001, ADR('Lg'), ADR(Log.Text));
87         TimeBf:=SysGetSysTime(TRUE); (* Time buffer (uS) *)
88         LockTime:=60; (* LockTime (Web page) *)
89         CaseNr:=CaseNr+1; (* Program case *)
90
91         (* ---------------------------------------------------------------- *)
92         (* Se continuo a leggere TAGs reinizializzo tempo. *)
93
94         2:
95         IF (TAGRd) THEN TAGRd:=FALSE; LockTime:=60; END_IF;
96
97         (* Temporizzazione blocco apertura. *)
98
99         IF ((SysGetSysTime(TRUE)-TimeBf) < 1000000) THEN RETURN; END_IF;
100        TimeBf:=SysGetSysTime(TRUE); (* Time buffer (uS) *)
101        LockTime:=LockTime-1; (* LockTime (Web page) *)
102        IF (LockTime > 0) THEN RETURN; END_IF;
103
104        (* Fine blocco apertura. *)
```

```
105
106         Log(Text:='Fine blocco'); (* Salvo log *)
107         i:=SysSpyData(SPY_ASCII, 16#00000001, ADR('Lg'), ADR(Log.Text));
108         WrongTCtr:=0; (* Wrong TAG counter *)
109         CaseNr:=0; (* Program case *)
110
111         (* ------------------------------------------------------------------ *)
112         (* APERTURA PORTA                                                     *)
113         (* ------------------------------------------------------------------ *)
114         (* TAG trovato, eseguo log. *)
115
116         10:
117         i:=SysVarsnprintf(ADR(Log.Text), SIZEOF(Log.Text), 'Apertura da:%s', STRING_TYPE, ADR(TAGs[TAGI
    Dx].User));
118         i:=SysSpyData(SPY_ASCII, 16#00000001, ADR('Lg'), ADR(Log.Text));
119         Log(); (* Salvo log *)
120
121         (* Eseguo comando apertura porta. *)
122
123         DoorUlck:=TRUE; (* Comando apertura porta *)
124         WrongTCtr:=0; (* Wrong TAG counter *)
125         TimeBf:=SysGetSysTime(TRUE); (* Time buffer (uS) *)
126         CaseNr:=CaseNr+1; (* Program case *)
127
128         (* ------------------------------------------------------------------ *)
129         (* Ricerca TAG in memoria. *)
130
131         11:
132         IF ((SysGetSysTime(TRUE)-TimeBf) < 1000000) THEN RETURN; END_IF;
133         DoorUlck:=FALSE; (* Comando apertura porta *)
134         CaseNr:=0; (* Program case *)
135     END_CASE;
136
137 (* [End of file] *)
138
139
```

# PROGRAM WiegandAcq

```
VAR
LInp : SysGetPhrDI; (* Acquisizione ingressi logici *)
Data : ARRAY[ 0..1 ] OF BOOL; (* Appoggio ingressi dato *)
BTime : UDINT; (* Buffer time (uS) *)
DStrobe : BOOL; (* Data strobe *)
Value : DWORD; (* Digital input value *)
END_VAR
```

```
(* ********************************************************************** *)
(* PROGRAM "WiegandAcq"                                                   *)
(* ********************************************************************** *)
(* Eseguo acquisizione bus Wiegand connesso agli inputs del modulo CPU.   *)
(* Per collegare il lettore dell'SH4 usare un cavo di rete.               *)
(* Marrone/Bianco: +12Vdc                                                 *)
(* Blu:            GND                                                    *)
(* Verde:          Wiegand D0                                             *)
(* Marrone:        Wiegand D1                                             *)
(* Arancio/Bianco: LED verde (Mettendolo a massa il LED diventa verde)    *)
(* Arancio:        Buzzer (Mettendolo a massa il buzzer suona)            *)
(* ---------------------------------------------------------------------- *)

    (* ---------------------------------------------------------------------- *)
    (* INITIALIZATION                                                         *)
    (* ---------------------------------------------------------------------- *)
    (* Program initialization. *)

    IF (SysFirstLoop) THEN
        DWiegand:=SysSetTaskLpTime(TaskID:=ID_TASK_FAST, Time:=150);
        LInp.Address:=255; (* Module address *)
        LInp.Mode:=DI_I_8_LL; (* Acquisition mode *)
    END_IF;

    (* ---------------------------------------------------------------------- *)
    (* WIEGAND DATA ACQUISITION                                               *)
    (* ---------------------------------------------------------------------- *)
    (* Se non variazione ingressi per tempo definito bus a riposo. *)

    IF (DStrobe) THEN
        IF ((SysGetSysTime(TRUE)-BTime) > 100000) THEN
            DStrobe:=FALSE; (* Data strobe *)
            OkWiegand:=TRUE; (* Wiegand data Ok *)
        END_IF;
    END_IF;

    (* Digital input acquisistion. *)

    LInp(); (* Acquisizione ingressi logici *)

    (* Controllo se variazione ingressi. *)

    IF (LInp.Value = Value) THEN RETURN; END_IF;
    Value:=LInp.Value; (* Digital input value *)
    BTime:=SysGetSysTime(TRUE); (* Buffer time (uS) *)

    (* Se variazione ingressi dopo bus a riposo start dati. *)

    IF NOT(DStrobe) THEN
```

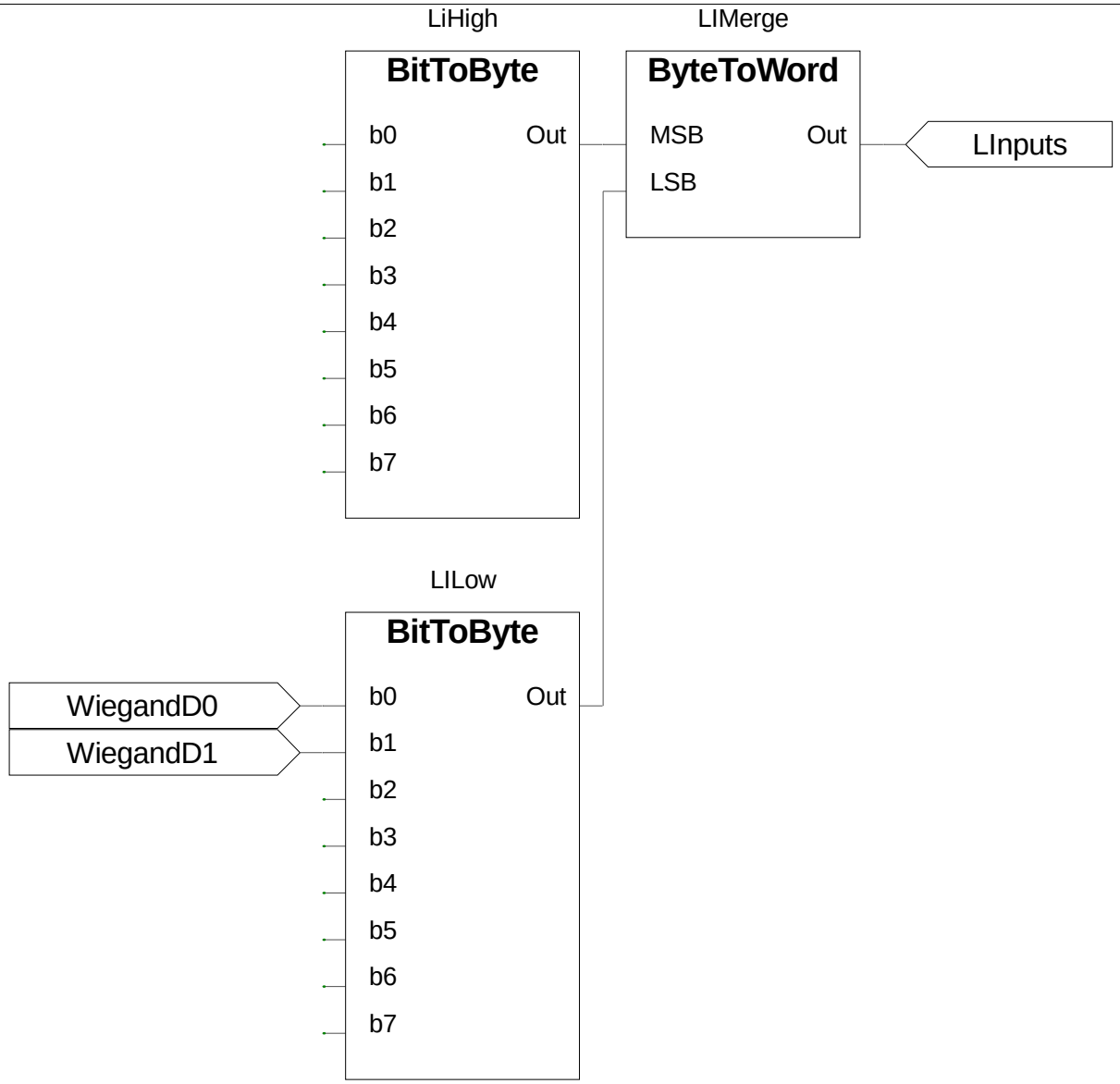
```
50          DStrobe:=TRUE; (* Data strobe *)
51          BWiegand:=0; (* Wiegand bits *)
52          DWiegand:=0; (* Wiegand data *)
53      END_IF;
54
55      (* Gestione "D0" ogni transizione è un bit con valore "0". *)
56
57      IF (TO_BOOL(LInp.Value AND 16#01) <> Data[0]) THEN
58          Data[0]:=NOT(Data[0]); (* Appoggio ingresso "D0" *)
59          IF NOT(Data[0]) THEN DWiegand:=(DWiegand*2); BWiegand:=BWiegand+1; END_IF;
60      END_IF;
61
62      (* Gestione "D1" ogni transizione è un bit con valore "1". *)
63
64      IF (TO_BOOL(LInp.Value AND 16#02) <> Data[1]) THEN
65          Data[1]:=NOT(Data[1]); (* Appoggio ingresso "D1" *)
66          IF NOT(Data[1]) THEN DWiegand:=(DWiegand*2)+16#00000001; BWiegand:=BWiegand+1; END_IF;
67      END_IF;
68
69  (* [End of file] *)
70
```

```
(* ********************************************************************** *)
(* PROGRAM "StartUp"                                                      *)
(* ********************************************************************** *)
(* Gestione inizializzazioni.                                             *)
(* ---------------------------------------------------------------------- *)

    (* ---------------------------------------------------------------------- *)
    (* ESEGUO IMPOSTAZIONI FB DI LOG                                          *)
    (* ---------------------------------------------------------------------- *)
    (* ESeguo impostazioni FB di log. *)

    Log.Folder:='Storage/'; (* Cartella files di log *)
    Log.NrOfLogs:=3; (* Numero di mesi in log *)

(* [End of file] *)
```

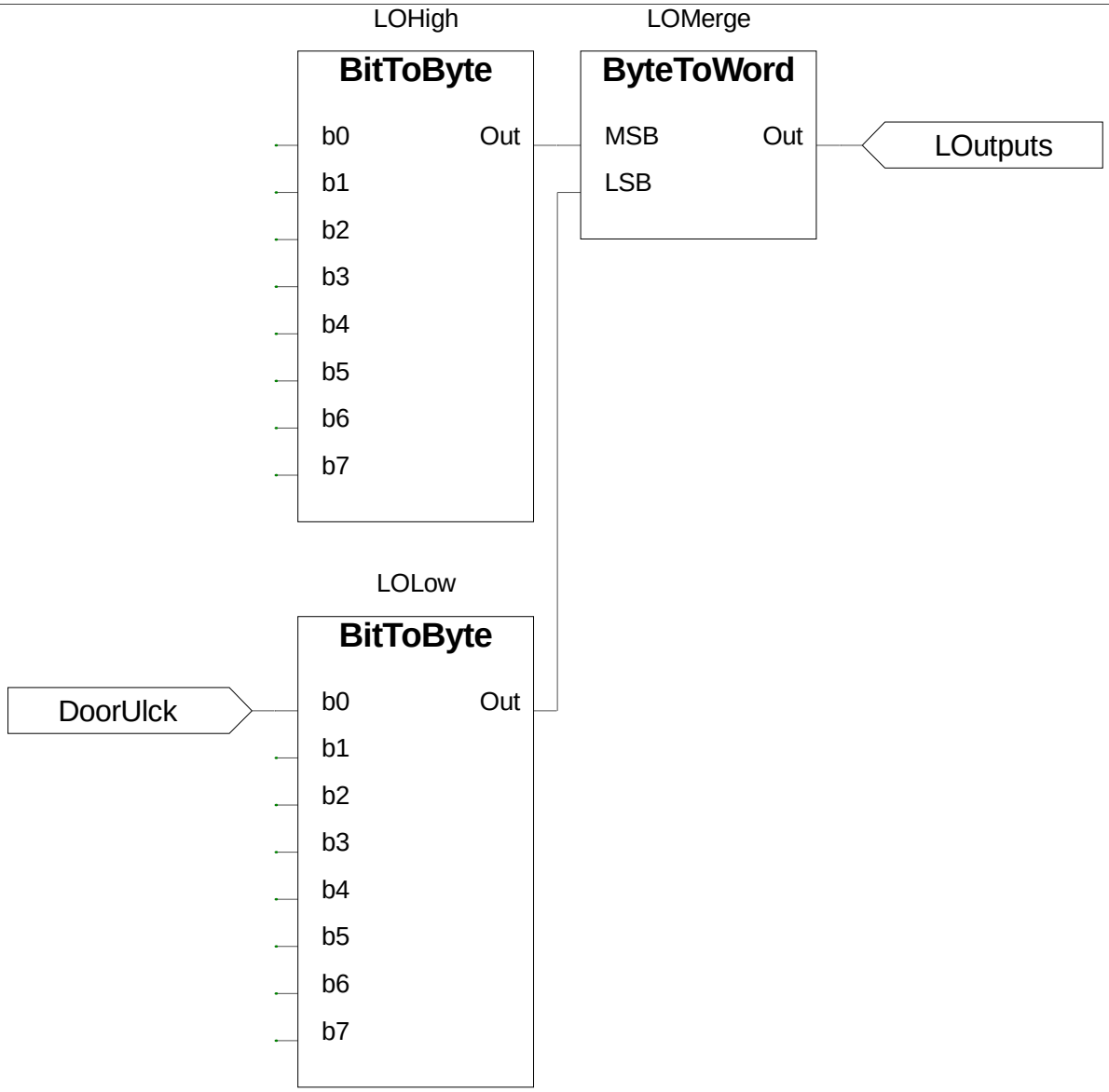
PROGRAM WebIOStatus

```
VAR
LILow : BitToByte; (* Input merge (Low) *)
LiHigh : BitToByte; (* Input merge (High) *)
LIMerge : ByteToWord; (* Input merge *)
LOLow : BitToByte; (* Output merge (Low) *)
LOHigh : BitToByte; (* Output merge (High) *)
LOMerge : ByteToWord; (* Output merge *)
END_VAR
```

1




| | | |
|---|---|---|
| | Project : eDoorkeeper | |
| | PROGRAM : WebIOStatus | |
| | Release : Xtarget | Ver :1.00 |
| | Author : | Date:19/01/2017 |
| | Note : | Page:1 of 2 |




2

| | | |
|---|---|---|
| | Project : eDoorkeeper | |
| | PROGRAM : WebIOStatus | |
| | Release : Xtarget | Ver :1.00 |
| | Author : | Date:19/01/2017 |
| | Note : | Page:2 of 2 |

FUNCTION_BLOCK Wiegand26Dcd

(eLLabUtyLib_C050) Decodes the Wiegand 26 bits code
ENCRYPTED CODE

```
VAR_INPUT
Enable : BOOL; (* FB enable *)
WBits : DWORD; (* Wiegand bits code *)
END_VAR

VAR_OUTPUT
CodeOk : BOOL; (* Code correctly decoded *)
Fault : BOOL; (* Execution fault *)
Facility : USINT; (* Facility code *)
IDNumber : UINT; (* TAG ID number *)
END_VAR
```

1

| | | |
|---|---|---|
| | Project : eDoorkeeper | |
| | FUNCTION BLOCK : Wiegand26Dcd | |
| | Release : Xtarget | Ver :1.00 |
| | Author : | Date:19/01/2017 |
| | Note : | Page:1 of 1 |

# FUNCTION_BLOCK WiegandKbd

```
VAR_INPUT
Enable : BOOL; (* FB enable *)
WBits : DWORD; (* Wiegand bits code *)
END_VAR

VAR_OUTPUT
EKeyPsd : BOOL; (* Enter key pressed *)
Digits : USINT; (* Numero digits *)
Value : UDINT; (* Valore inputato *)
END_VAR

VAR
TimeBf : UDINT; (* Time buffer (uS) *)
END_VAR
```

```
(* ********************************************************************** *)
(* BLOCCO FUNZIONE "WiegandKbd"                                           *)
(* ********************************************************************** *)
(* Eseguo acquisizione tastiera.                                          *)
(* ---------------------------------------------------------------------- *)

    (* ---------------------------------------------------------------- *)
    (* GESTIONE STROBE DATI                                             *)
    (* ---------------------------------------------------------------- *)
    (* Reset dati se entro dopo il termine di una inputazione. *)

    IF (EKeyPsd) THEN Digits:=0; Value:=0; EKeyPsd:=FALSE; END_IF;

    (* Eseguo gestione abilitazione. *)

    IF NOT(Enable) THEN
        Digits:=0; (* Numero digits *)
        Value:=0; (* Valore inputato *)
        RETURN;
    END_IF;

    (* Eseguo controllo se codice a 8 bits. Se arriva un codice errato viene *)
    (* abortita l'eventuale inputazione in corso. *)

    IF ((WBits AND 16#FFFFFF00) <> 0) THEN
        Digits:=0; (* Numero digits *)
        Value:=0; (* Valore inputato *)
        RETURN;
    END_IF;

    (* Controllo se timeout inputazione. *)

    IF ((SysGetSysTime(TRUE)-TimeBf) > 5000000) THEN Digits:=0; Value:=0; END_IF;
    TimeBf:=SysGetSysTime(TRUE); (* Time buffer (uS) *)

    (* ---------------------------------------------------------------- *)
    (* GESTIONE TASTIERA                                                *)
    (* ---------------------------------------------------------------- *)
    (* Eseguo data entry. *)

    CASE (WBits) OF

```

```
43          (* ---------------------------------------------------------------- *)
44          (* Gestione tasti numerici. *)
45
46          240: Value:=(Value*10)+0; Digits:=Digits+1;
47          225: Value:=(Value*10)+1; Digits:=Digits+1;
48          210: Value:=(Value*10)+2; Digits:=Digits+1;
49          195: Value:=(Value*10)+3; Digits:=Digits+1;
50          180: Value:=(Value*10)+4; Digits:=Digits+1;
51          165: Value:=(Value*10)+5; Digits:=Digits+1;
52          150: Value:=(Value*10)+6; Digits:=Digits+1;
53          135: Value:=(Value*10)+7; Digits:=Digits+1;
54          120: Value:=(Value*10)+8; Digits:=Digits+1;
55          105: Value:=(Value*10)+9; Digits:=Digits+1;
56
57          (* ---------------------------------------------------------------- *)
58          (* Gestione tasto ESC. *)
59
60          90: Digits:=0; Value:=0;
61
62          (* ---------------------------------------------------------------- *)
63          (* Gestione tasto ENT. *)
64
65          75: EKeyPsd:=TRUE; (* Enter key pressed *)
66      END_CASE;
67
68  (* [End of file] *)
69
```

FUNCTION_BLOCK BitToByte

(eLLabUtyLib_C050) Merges 8 BOOL variables into a BYTE
ENCRYPTED CODE

```
VAR_INPUT
b0 : BOOL;
b1 : BOOL;
b2 : BOOL;
b3 : BOOL;
b4 : BOOL;
b5 : BOOL;
b6 : BOOL;
b7 : BOOL;
END_VAR

VAR_OUTPUT
Out : BYTE; (* Function result *)
END_VAR
```

1

| | | |
|---|---|---|
| | Project : eDoorkeeper | |
| | FUNCTION BLOCK : BitToByte | |
| | Release : Xtarget | Ver :1.00 |
| | Author : | Date:19/01/2017 |
| | Note : | Page:1 of 1 |

FUNCTION_BLOCK ByteToWord

(eLLabUtyLib_C050) Merges 2 BYTE variables into a WORD
ENCRYPTED CODE

```
VAR_INPUT
MSB : BYTE; (* MSB Value *)
LSB : BYTE; (* LSB Value *)
END_VAR

VAR_OUTPUT
Out : WORD; (* Function result *)
END_VAR
```

1

| | | |
|---|---|---|
| | Project : eDoorkeeper | |
| | FUNCTION BLOCK : ByteToWord | |
| | Release : Xtarget | Ver :1.00 |
| | Author : | Date:19/01/2017 |
| | Note : | Page:1 of 1 |

# FUNCTION_BLOCK LogWrite

```
VAR_INPUT
NrOfLogs : USINT; (* Number of month in log *)
Folder : STRING[ 16 ]; (* Folder name *)
Text : STRING[ 64 ]; (* Pointer to log text *)
END_VAR

VAR_EXTERNAL
SysDateTime : UDINT; (* System Date/Time epoch *)
END_VAR

VAR
Pulse : BOOL; (* Time base pulse *)
NrOfLog : USINT; (* Number of log stored *)
i : INT; (* Auxiliary counter *)
LogRecord : STRING[ 64 ]; (* Log record *)
Fp : FILEP; (* File pointer *)
DateTime : SysETimeToDate; (* Date/Time conversion *)
TimeBf : UDINT; (* Time buffer (uS) *)
Filename : STRING[ 32 ]; (* Nome file *)
LogMonth : USINT; (* Mese in log *)
j : INT; (* Auxiliary counter *)
END_VAR
```

```
 1 (* ************************************************************************** *)
 2 (* FUNCTION BLOCK "LogWrite"                                                  *)
 3 (* ************************************************************************** *)
 4 (* Questo blocco funzione esegue la scrittura di un record di log in un file  *)
 5 (* il nome del file viene composto utilizzando il nome del mese. I files più  *)
 6 (* vecchi vengono eliminati, è possibile definire quanti files mantenere.     *)
 7 (* Il FB può essere eseguito come una funzione.                               *)
 8 (*                                                                            *)
 9 (* Il record scritto è del tipo: "31/01/2017;09:25:58;Record di log"          *)
10 (* -------------------------------------------------------------------------- *)
11
12     (* -------------------------------------------------------------------- *)
13     (* GESTIONE DATA                                                        *)
14     (* -------------------------------------------------------------------- *)
15     (* Gestione data/ora attuali, il valore dei mesi è espresso da 1 a 12. *)
16
17     DateTime(EpochTime:=SysDateTime); (* Data ed ora attuali *)
18
19     (* -------------------------------------------------------------------- *)
20     (* MEMORIZZAZIONE LOG MENSILE                                           *)
21     (* -------------------------------------------------------------------- *)
22     (* Controllo se mese attuale diverso da mese in log. *)
23
24     IF (DateTime.Month <> LogMonth) THEN
25         LogMonth:=DateTime.Month; (* Mese in log *)
26
27         (* Eseguo cancellazione records di log più vecchi. *)
28
29         FOR i:=TO_INT(LogMonth)+1 TO TO_INT(LogMonth+(12-NrOfLogs)) DO
30             IF (i > 12) THEN j:=i-12; ELSE j:=i; END_IF;
31
32             CASE (j) OF
33                 1: Filename:='Gennaio'; (* Nome file *)
34                 2: Filename:='Febbraio'; (* Nome file *)
```

```
                3: Filename:='Marzo'; (* Nome file *)
                4: Filename:='Aprile'; (* Nome file *)
                5: Filename:='Maggio'; (* Nome file *)
                6: Filename:='Giugno'; (* Nome file *)
                7: Filename:='Luglio'; (* Nome file *)
                8: Filename:='Agosto'; (* Nome file *)
                9: Filename:='Settembre'; (* Nome file *)
                10: Filename:='Ottobre'; (* Nome file *)
                11: Filename:='Novembre'; (* Nome file *)
                12: Filename:='Dicembre'; (* Nome file *)
            END_CASE;

            Filename:=CONCAT(Folder, Filename); (* Aggiungo definizione cartella *)
            Filename:=CONCAT(Filename,'.txt'); (* Aggiungo estensione file *)
            j:=Sysremove(Filename); (* Eseguo cancellazione file *)
        END_FOR;
    END_IF;

    (* ---------------------------------------------------------------------- *)
    (* DEFINIZIONE FILE DI LOG                                                *)
    (* ---------------------------------------------------------------------- *)
    (* Creo nome del file di log. *)

    CASE (LogMonth) OF
        1: Filename:='Gennaio'; (* Nome file *)
        2: Filename:='Febbraio'; (* Nome file *)
        3: Filename:='Marzo'; (* Nome file *)
        4: Filename:='Aprile'; (* Nome file *)
        5: Filename:='Maggio'; (* Nome file *)
        6: Filename:='Giugno'; (* Nome file *)
        7: Filename:='Luglio'; (* Nome file *)
        8: Filename:='Agosto'; (* Nome file *)
        9: Filename:='Settembre'; (* Nome file *)
        10: Filename:='Ottobre'; (* Nome file *)
        11: Filename:='Novembre'; (* Nome file *)
        12: Filename:='Dicembre'; (* Nome file *)
    END_CASE;

    (* Open the file in "append" mode. *)

    Filename:=CONCAT(Folder, Filename); (* Aggiungo definizione cartella *)
    Filename:=CONCAT(Filename,'.txt'); (* Aggiungo estensione file *)
    Fp:=Sysfopen(Filename, 'a'); (* File pointer *)
    IF (Fp = NULL) THEN RETURN; END_IF;

    (* ---------------------------------------------------------------------- *)
    (* WRITE THE LOG                                                          *)
    (* ---------------------------------------------------------------------- *)
    (* Eseguo scrittura in append sul file del record di log. *)

    i:=SysVarsnprintf(ADR(LogRecord), SIZEOF(LogRecord), '%02d/', USINT_TYPE, ADR(DateTime.Day));
    i:=SysLWVarsnprintf(ADR(LogRecord), SIZEOF(LogRecord), '%02d/', USINT_TYPE, ADR(DateTime.Month));
    i:=SysLWVarsnprintf(ADR(LogRecord), SIZEOF(LogRecord), '%04d;', UINT_TYPE, ADR(DateTime.Year));
    i:=SysLWVarsnprintf(ADR(LogRecord), SIZEOF(LogRecord), '%02d:', USINT_TYPE, ADR(DateTime.Hour));
    i:=SysLWVarsnprintf(ADR(LogRecord), SIZEOF(LogRecord), '%02d:', USINT_TYPE, ADR(DateTime.Minute));
    i:=SysLWVarsnprintf(ADR(LogRecord), SIZEOF(LogRecord), '%02d;', USINT_TYPE, ADR(DateTime.Second));
    i:=SysLWVarsnprintf(ADR(LogRecord), SIZEOF(LogRecord), '%s$r$n', STRING_TYPE, ADR(Text));

    (* Eseguo scrittura record nel file. *)

```

```
95     i:=Sysfwrite(ADR(LogRecord), TO_INT(LEN(LogRecord)), 1, Fp); (* Write to file *)
96     i:=Sysfclose(Fp); (* Close file *)
97
98 (* [End of file] *)
99
```