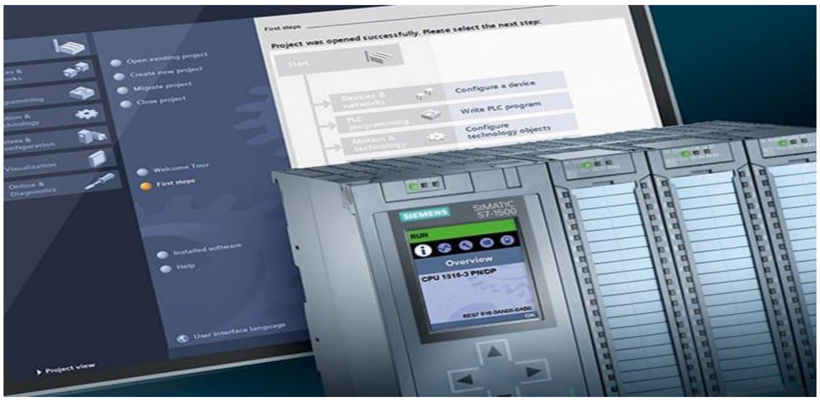
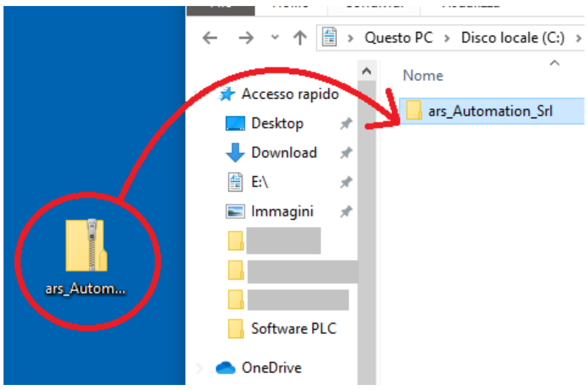
SIEMENS FLEXIBOWL PLUGIN

This plugin was created with the idea of communicating quickly and safely with the FlexiBowl® throuth the Siemens PLCs S7-1200 and S7-1500 series using KOP/SCL instructions language.
Whit the plugin developed in TIA Portal V15 V15.1 V16 V17 doesn't required additional licenses and it was develop for comunicate throuth the ethernet board integrated in PLCs Siemens.
Requited firmware 4.0 version or higher for PLC S7-1200
Requited firmware 2.0 version or higher for PLC S7-1500
Requited firmware 2.0 version or higher for PLC ET200SP

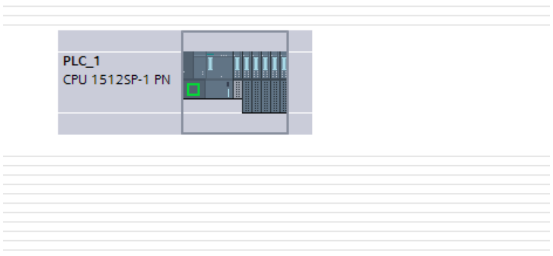


STEP 1:

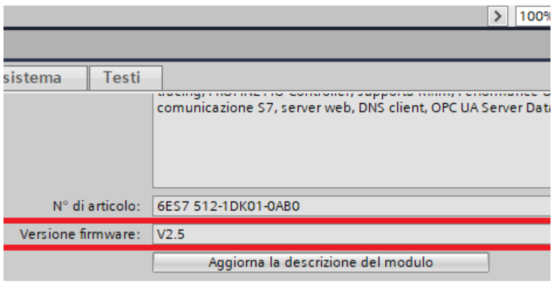


Unpackage the "ars_Automation_Srl" Library in a desired folder

STEP 2:

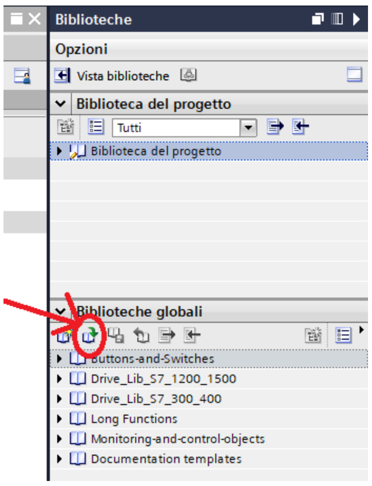


Make sure in the hardware configuration of the TIA Portal that in the CPU firmware version it is greater than or equal to 2.0 for the S7-1500 or ET200 SP PLCs or greater than or equal to 4.0 for the S7-1200 PLCs


100%
sistema
Testi
comunicazione S7, server web, DNS client, OPC UA Server Dat
Aggiorna la descrizione del modulo

N° di articolo: 6ES7 512-1DK01-0AB0
Versione firmware: V2.5

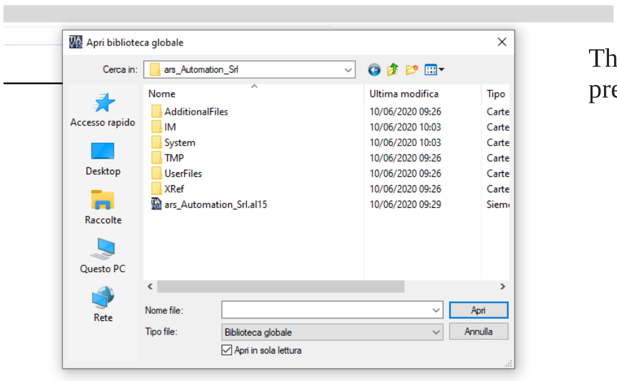
STEP 3:



Click on the "Open global library" icon inside the section library in the TIA portal.

TIA Portal

STEP 4:


Apri biblioteca globale

Cerca in: ars_Automation_Srl

Nome	Ultima modifica	Tipo
AdditionalFiles	10/06/2020 09:26	Carte
IM	10/06/2020 10:03	Carte
System	10/06/2020 10:03	Carte
TMP	10/06/2020 09:26	Carte
UserFiles	10/06/2020 09:26	Carte
XRef	10/06/2020 09:26	Carte
ars_Automation_Srl.al15	10/06/2020 09:29	Siem

Nome file:

Tipo file: Biblioteca globale

[✓] Apri in sola lettura

Then select the library file within the previously unzipped folder.

STEP 5:

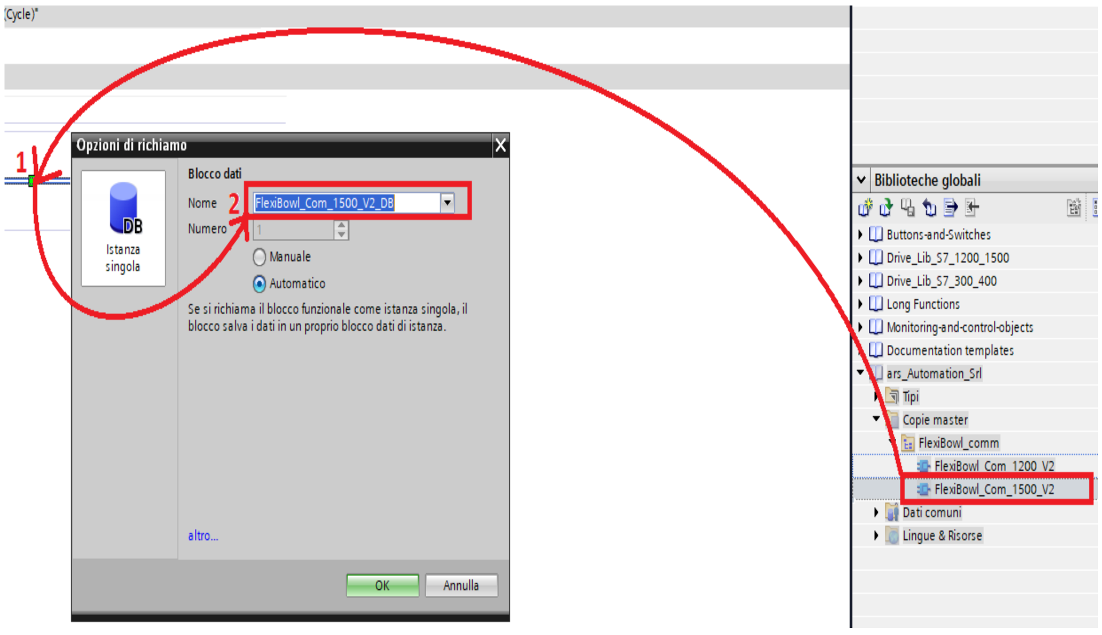
Drag the communication block now available in the library into the software; select the block according to the CPU you are using:

• FlexiBowl_com_1200_V2 for S7-1200;

• FlexiBowl_com_1500_V2 for S7-1500 or ET200 SP;

Once the block has been inserted into the software, you will be asked to give an instance DB. Multi-instance blocks can also be created.

NB: It is recommended to use only one block for each FlexiBowl you intend to communicate and manage the requests to be sent upstream of the block, using the "Done" and "Busy" outputs of the block


Opzioni di richiamo

Istanza singola

Blocco dati

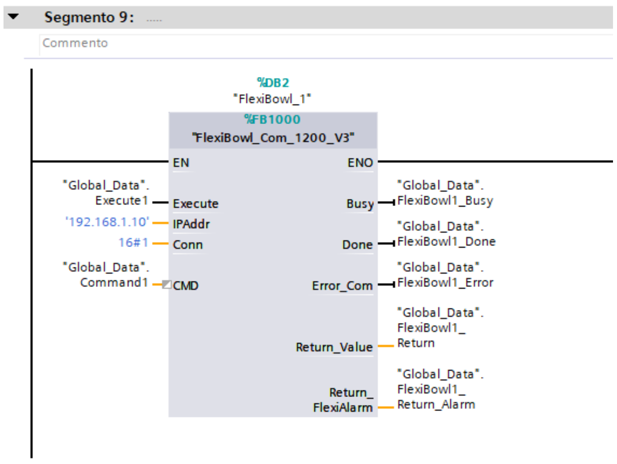
Nome: FlexiBowl_Com_1500_V2_DB

Numero: 1

Manuale [✓ Automatico]

Se si richiama il blocco funzionale come istanza singola, il blocco salva i dati in un proprio blocco dati di istanza.

STEP 6:



Now populate the block inputs and outputs according to your needs:

TAG	Type	Description
EN	IN:Bool	Block Enable. It must always be enabled because inside the block there are instructions that keeping the TCP connection active
Execute	IN:Bool	Send the present command on the "CMD" input. The block considerate this input into only for one scan of PLC cycle and if communication is established.
IPAddr	IN:String	IP address of the FlexiBowl. Specify the IP address of the Flexibowl. The length of the string is constrained, therefore it is necessary to fill all 4 triads with three numbers eg: 192.168.0.161 → '192.168.000.161' 10.100.120.5 → '010.100.120.005' The TCP communication instructions are high level and managed by the PLC ethernet soket therefore even if the FlexiBowl is in another subnet and there is a reference gateway set in the hardware configuration of the PLC it is possible to directly specify the final FlexiBowl IP address.

STEP 6:

TAG	Type	Description
CMD	IN:String	Command in string format to be sent to the flexiBowl (see table below)
Conn	IN: CONN_OUC	Connection ID, this parameter must be hexadecimal ex: “16#01” and must be unique and including between 1 and the maximum number of TCP connection supported by the PLC ex 32
ENO	OUT:Bool	Block execution successful without any software error
Busy	OUT:Bool	QX execution in progress by FlexiBowl, therefore no other command can be sent.
Done	OUT:Bool	Last command execution successful. Both on the first scan cycle of the PLC and for each communication error even if a command is not being executed, this bit is set to FALSE. Otherwise it always remains at TRUE following a successfully completed instruction until the next request.
Error	OUT:Bool	This bit is set to TRUE for each communication error and reset each time communication is restored.
Return	OUT:String	Return string from FlexyBowl. In the case of execution of a QX the '**Mooving**' value will be returned until completion. Once the QX instruction has been completed, the value of the last status request to the FlexiBowl will be found until the next execution. In case of a communication error, the value ‘**Communication Error. Check state on Nettwork&Device**’ will be returned.
Return_ FlexiAlarm	OUT:Word	Word value holding eventual error code readed from the FlexiBowl driver. See the next table.

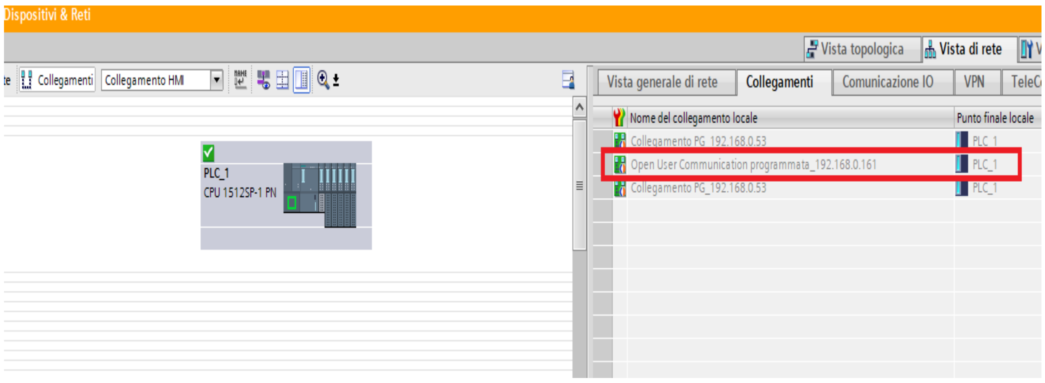
COMMAND LIST:

Command	Description
QX2	Move
QX3	Move-Flip
QX4	Move-Flip-Blow
QX5	Move-Blow
QX6	Shake
QX7	Light on
QX8	Light off
QX9	Flip
QX10	Blow
QX11	Quick Emptying Option
QX12	Reset Alarm

Alarm list:

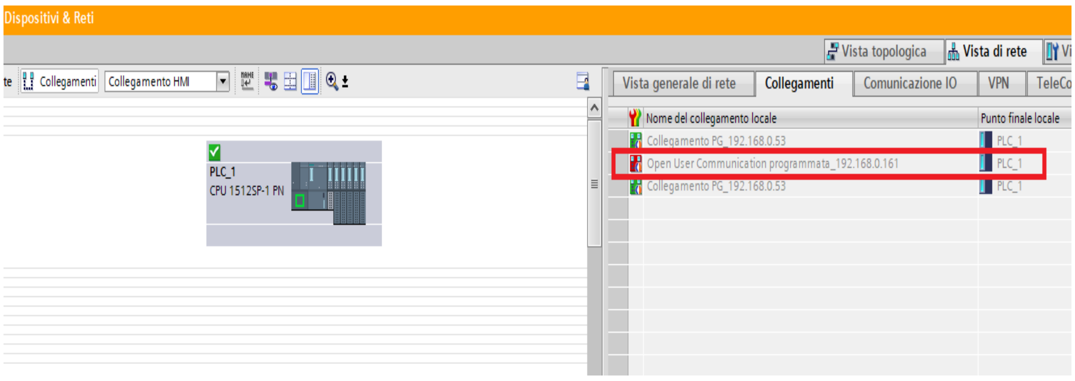
Hex Value	Descrizione
16#0001	Position Limit
16#0002	CCW Limit
16#0004	CW Limit
16#0008	Over Temp
16#0010	Excess Regen
16#0020	Over Voltage
16#0040	Under Voltage
16#0080	Over Current
16#0100	Open Motor Winding
16#0200	Bad Encoder
16#0400	Comm Error
16#0800	Bad Flash
16#1000	No Move
16#2000	Motor Resistance Out of Range
16#4000	Blank Q Segment
16#8000	(not used)

When the block has managed to open the communication to the FlexiBowl, it is possible to check online for the presence of the connection icon established in the connections section within the network configuration page of TIA Portal:



Nome del collegamento locale	Punto finale locale
Collegamento PG_192.168.0.53	PLC_1
Open User Communication programmata_192.168.0.161	PLC_1
Collegamento PG_192.168.0.53	PLC_1

Otherwise, when there is a communication error with the device, the situation is like follows:



Nome del collegamento locale	Punto finale locale
Collegamento PG_192.168.0.53	PLC_1
Open User Communication programmata_192.168.0.161	PLC_1
Collegamento PG_192.168.0.53	PLC_1

SCRIPT:

```
// Set the base parameter
// Convert the IP address into array of byte
#l := 1;
FOR #i := 1 TO 15 DO
    IF (#IPAddr[#i] <> '.') AND (#IPAddr[#i] <> '$00') THEN
        #string_app1 := CONCAT(IN1 := #string_app1, IN2 := #IPAddr[#i]);
    END_IF;
    IF (#IPAddr[#i] = '.') OR (#IPAddr[#i] = '$00') OR (#i = 15) THEN
        #Skt_Param_Snd.RemoteAddress.ADDR[#l] := UINT_TO_BYTE(IN := STRING_TO_UINT(#string_app1));
        #string_app1 := '';
        #l := #l + 1;
    END_IF;
    IF (#IPAddr[#i] = '$00') THEN
        EXIT;
    END_IF;
END_FOR;

// Set the IP address into array of byte
#Skt_Param_Snd.ConnectionType := 11;                    // 11=TCP
#Skt_Param_Snd.ActiveEstablished := TRUE;               // Stabilite active connection from the PLC
#Skt_Param_Snd.RemotePort := 7776;                      // FlexiBowl TCP listening port
#Skt_Param_Snd.ID := #Conn;                             // ID for connction parameter
#Skt_Param_Snd.LocalPort := 0;                          // Local port 0=Dinamic

#KeepAlive_Timer(IN := ((#SktTCPSend_instance.s_TDIAG_Status.State = 16#04) AND (NOT #Execute) AND (NOT
#InsideBusy) AND (NOT #ReadContSts)),
                 PT := T#20S);

IF #KeepAlive_Timer.Q THEN
    #ReadContSts := TRUE;
    #KeepAlive_Flag := TRUE;
END_IF;

// ####################################################################################################
#Exe_trig(CLK := #Execute);                             // Execution trigger

// Inizialize data
IF #Exe_trig.Q  AND (#SktTCPSendinstance.sTDIAGStatus.State = 16#04) THEN
    #InsideBusy := TRUE;
    #Done := FALSE;
    #ReadContSts := FALSE;
    #ReadAlarm := FALSE;
    #Error_Com := FALSE;
    #KeepAlive_Flag := FALSE;

    FOR #i := 0 TO 20 DO
        #SendSocketDat[#i] := 16#00;
    END_FOR;
    #SendSocketDat[1] := 16#07;

    FOR #i := 0 TO 51 DO
        #RcvSocketDat[#i] := 16#00;
    END_FOR;

    // --------------------------------------------
    // Arrange the message to send
    // Headre   16#00 16#07
    // Message  ascii to hex message
    // Footer   16#0D

    FOR #i := 0 TO 19 DO
        #SendSocketDat[#i + 2] := #CMD[#i + 1];
        IF #SendSocketDat[#i + 2] = 16#00 THEN
            #SendSocketDat[#i + 2] := 16#0D;
            EXIT;
        END_IF;
    END_FOR;
// --------------------------------------------
END_IF;
```

SCRIPT:

```
#ReadAlm_trig(CLK := (NOT #ReadContSts));                                              // Execution read alarme trigger

// If ReadContSts funtion active send 'SC' command
IF #ReadContSts THEN
   #SendSocketDat[0] := 16#00;
   #SendSocketDat[1] := 16#07;
   #SendSocketDat[2] := 'S';
   #SendSocketDat[3] := 'C';
   #SendSocketDat[4] := 16#0D;
   FOR #i := 5 TO 19 DO
      #SendSocketDat[#i] := 16#00;
   END_FOR;
END_IF;

// If ReadAlarm funtion active send 'AL' command
IF #ReadAlm_trig.Q THEN
   #SendSocketDat[0] := 16#00;
   #SendSocketDat[1] := 16#07;
   #SendSocketDat[2] := 'A';
   #SendSocketDat[3] := 'L';
   #SendSocketDat[4] := 16#0D;
   FOR #i := 5 TO 19 DO
      #SendSocketDat[#i] := 16#00;
   END_FOR;
   #ReadAlarm := TRUE;
END_IF;

// ##########################################################################################
// Send data Section
// Calculate Message Size
FOR #i := 2 TO 19 DO
   IF (#SendSocketDat[#i] = 16#0D) THEN
      #SndSize:=INT_TO_UDINT(#i);
      EXIT;
   END_IF;
END_FOR;

#Snd_Trig(CLK := #SktTCPRcv_instance.DONE OR #KeepAlive_Timer.Q);                 // Execution trigger
#SktTCPSend_instance(REQ := ((#Exe_trig.Q OR (#Snd_Trig.Q AND #ReadContSts) OR #ReadAlm_trig.Q)),
                CONT := TRUE,
                CONNECT := #Skt_Param_Snd,
                DATA := #SendSocketDat,
                LEN := #SndSize + 1);

// ##########################################################################################
// Receive data Section
#SktTCPRcv_instance(EN_R := TRUE,
               CONT := TRUE,
               CONNECT := #Skt_Param_Snd,
               DATA := #RcvSocketDat,
               LEN := 0,
               ADHOC := TRUE);

// Decoding the message from the FlaxiBowl
IF #SktTCPRcv_instance.DONE AND (NOT #ReadAlarm) THEN
   #Return_Value := '';
   FOR #i := 2 TO 52 DO
      IF (#RcvSocketDat[#i] = 16#0D) THEN
         EXIT;
      END_IF;
      #Return_Value[#i - 1] := BYTE_TO_CHAR(#RcvSocketDat[#i]);
   END_FOR;
```

SCRIPT:

```
    // If received data from FlaxiBowl (SC= 4 xxx) and ReadContSts funtion active, write 'QX Running' on Return_Value
variable elseif received data from FlaxiBowl (SC = xx 1x) and ReadContSts funtion active, write 'Mooving'
    //                                           23   4 5678
    IF (#RcvSocketDat[7] = 16#31) AND #ReadContSts THEN
        #Return_Value := 'Mooving';
    END_IF;
    IF (#RcvSocketDat[5] = 16#34) AND #ReadContSts THEN
        #Return_Value := 'QX Running';
    END_IF;
END_IF;

// If the check FlexiBowl alarm it's done then set Done and reset Busy and terminate the function
IF #SktTCPRcv_instance.DONE AND #ReadAlarm THEN
    #h := 1;
    FOR #i := 5 TO 8 DO
        #string_app2[#h] := #RcvSocketDat[#i];
        #h := #h + 1;
    END_FOR;
    #Return_FlexiAlarm := STRING_TO_UINT(IN := #string_app2);
    #ReadAlarm := FALSE;
    #InsideBusy := FALSE;
    #Done := TRUE;
END_IF;

// If 'QX' command sent active ReadContSts funtion
IF #SktTCPRcv_instance.DONE AND (#SktTCPRcv_instance.RCVD_LEN <> 0) AND (#RcvSocketDat[2] = 16#25) AND
(NOT #ReadContSts) THEN
    #ReadContSts := TRUE;
END_IF;

// If received data from FlaxiBowl and not ReadContSts funtion active, set Done and reset Busy
IF #SktTCPRcv_instance.DONE AND (#SktTCPRcv_instance.RCVD_LEN <> 0) AND (NOT #ReadContSts) AND (NOT
#ReadAlarm) THEN
    #InsideBusy := FALSE;
    #Done := TRUE;
END_IF;

// If received data from FlaxiBowl (SC= 0 001) and ReadContSts funtion active, set Done and reset Busy and terminate the
function                                  234 5 678
IF #SktTCPRcv_instance.DONE AND (#RcvSocketDat[7] = 16#30) AND #ReadContSts THEN
    #ReadContSts := FALSE;
END_IF;

IF #KeepAlive_Flag AND (#SktTCPRcv_instance.RCVD_LEN <> 0) THEN
    #KeepAlive_Flag := FALSE;
    #Error_Com := FALSE;
END_IF;

// #####################################################################################################
// TimeOut and error management
#Timeout_Timer(IN := (((NOT #ReadContSts) AND #InsideBusy) OR (#ReadContSts AND (NOT
#SktTCPRcv_instance.DONE)) OR #KeepAlive_Flag OR #ReadAlarm),
               PT := T#2S);

// Generate error
//
IF #Timeout_Timer.Q THEN
    #ReadContSts := FALSE;
    #ReadAlarm := FALSE;
    #KeepAlive_Flag := FALSE;
    #InsideBusy := FALSE;
    #Done := FALSE;
    #Error_Com := TRUE;
    #Return_Value := 'Communication Error. Check the flaxiBowl Con';
END_IF;

#Busy := #InsideBusy;
```