

# OMRON FLEXIBOWL PLUGIN



This plugin was created with the idea of communicating quickly and safely with the FlexiBowl® through the Omron PLC N series with the ethernet on board using LD/SC instructions language.

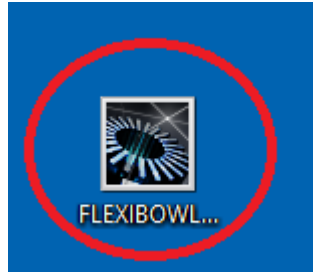
Whit the plugin developed in Sysmac Studio doesn't required additional licenses and it was develop for communicate through the ethernet board integrated in PLCs Omron.

Required firmware 1.16 version or higher

# FlexiBowl®

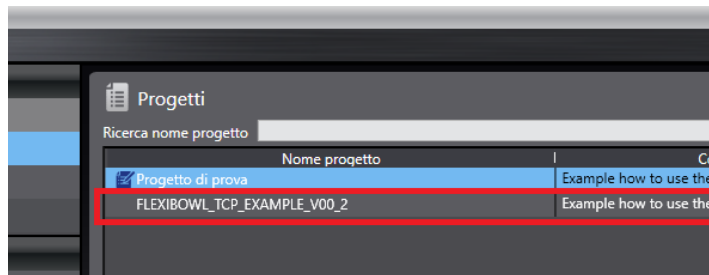


STEP 1:



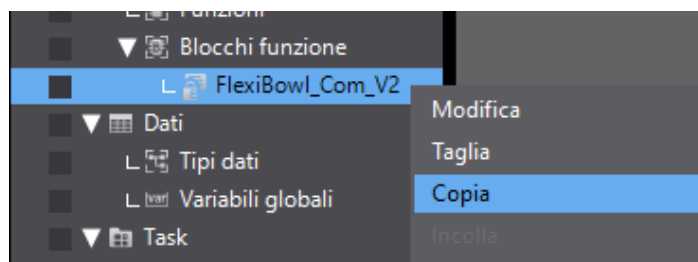
Unpackage the example project by double click on the file “FLEXIBOWL\_TCP\_EXAMPLE\_V00\_2”

STEP 2:



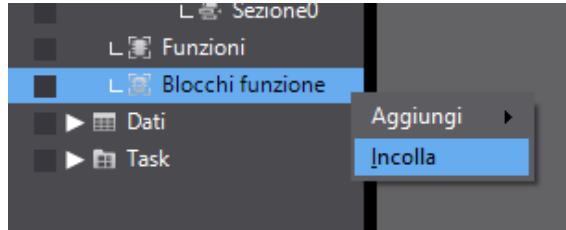
Now open the uncompressed project on the “Open project” section on Sysmac Studio

STEP 3:



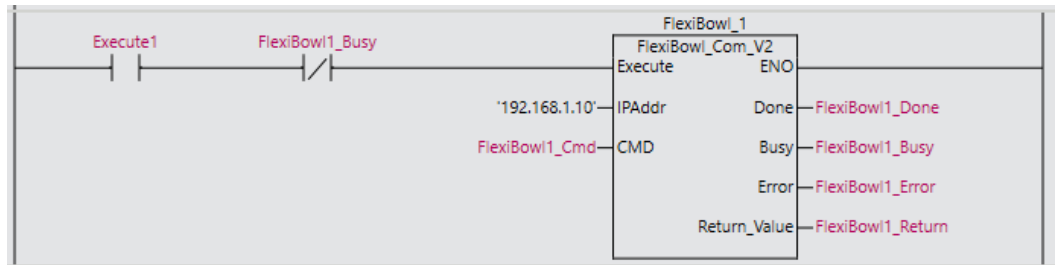
Then copy the function block “FlexiBowl\_Com\_V2”

STEP 4:



Right click in your project on the “Function block” section in the project’s tree then click paste

STEP 5:



Insert now the function block inside your software then populate the block inputs and outputs according to your needs:

| TAG     | Type      | Description  |
|---------|-----------|--|
| Execute | IN:Bool   | Send the present command on the "CMD" input.<br>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.<br>Specify the IP address of the Flexibowl. Es: '192.168.0.161'<br><br>The TCP communication instructions are high level and managed by the PLC ethernet socket 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. |
| CMD     | IN:String | Command in string format to be sent to the flexiBowl (see table below)   |

STEP 5:

| TAG    | Type       | Description   |
|--------|------------|---|
| 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 the flaxiBowl Con'. |

COMMAND  
LIST:

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

## SCRIPT:

```

// If socket closed then try to open socket
Time_Open(In:= (NOT Socket_Opened) AND (NOT Time_Open.Q),PT:=T#1S);
Open_trig(Clk:= Time_Open.Q);

SkTCPCConnect_instance(                               //Request a connection.
    Execute:=Open_trig.Q,
    SrcTcpPort:=UINT#0,                               // Local TCP port number: Automatically assigned.
    DstAdr:= IPAddr,                                 // Remote IP address
    DstTcpPort:=UINT#7776,                           // Destination TCP port number
    Socket=>WkSocket);                                // Socket

IF SkTCPCConnect_instance.Done THEN                  // If the socket is open
    Socket_Opened:=TRUE;
    Error := FALSE;
END_IF;

KeepAlive_Timer(In:= (Socket_Opened AND (NOT Execute) AND (NOT Busy) AND (NOT
KeepAlive_Timer.Q)),PT:=T#2S);
IF KeepAlive_Timer.Q THEN
    ReadContSts:=TRUE;
    KeepAlive_Flag:=TRUE;
END_IF;

// #####
// Initalize data for normal
Exe_trig(CLK := Execute); // Execution trigger

IF Exe_trig.Q AND Socket_Opened THEN
    Busy := TRUE;
    Done := FALSE;
    ReadContSts := FALSE;
    Error := 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;
END_IF;

```

## SCRIPT:

```

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

ToAryByte(In:=CMD, Order:=_LOW_HIGH, AryOut:=SendSocketDat[2]); // Convert text command into array byte
FOR i := 2 TO 19 DO
  IF SendSocketDat[i] = 16#00 THEN
    SendSocketDat[i] := 16#0D;
  EXIT;
  END_IF;
END_FOR;
// -----

// If ReadContSts funtion active send 'SC' command
IF ReadContSts THEN
  SendSocketDat[0] := 16#00;
  SendSocketDat[1] := 16#07;
  SendSocketDat[2] := 16#53;
  SendSocketDat[3] := 16#43;
  SendSocketDat[4] := 16#0D;
END_IF;

// #####
// Send data Section
SktTCPSend_instance(
  Execute:=(Exe_trig.Q OR (SktTCPRcv_instance.Done AND ReadContSts) OR KeepAlive_Timer.Q) AND
  Socket_Opened),
  Socket :=WkSocket, // Socket
  SendDat:=SendSocketDat[0], // Send data
  Size:=UINT#20); // Send data size

// #####
// Receive data Section
SktTCPRcv_instance(
  Execute:=SktTCPSend_instance.Done,
  Socket :=WkSocket, // Socket
  TimeOut:=UINT#0, // Timeout time
  Size :=UINT#52, // Receive data size
  RcvDat :=RcvSocketDat[0]); // Receive data

// Decoding the message from the FlaxiBowl
IF SktTCPRcv_instance.DONE AND Busy THEN
  Return_Value:=;
  FOR i := 2 TO 52 DO
    IF (RcvSocketDat[i] = 16#0D) THEN EXIT;
  END_IF;
  END_FOR;
  Return_Value:=AryToString(In:=RcvSocketDat[2], Size:=i-2);
END_IF;

```

## SCRIPT:

```

// If 'QX' command sent active ReadContSts funtion
IF (SendSocketDat[2] = 16#51) AND (SendSocketDat[3] = 16#58) 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.RcvSize <> 0) AND (NOT ReadContSts) THEN
  Busy := 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 (RcvSocketDat[5] = 16#30) AND ReadContSts THEN
  ReadContSts := FALSE;
  Busy := FALSE;
  Done := TRUE;
END_IF;

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

// If received data from FlaxiBowl (SC= 4 019) and ReadContSts funtion active, write 'Mooving' on Return_Value variable
//
234 5 678
IF (RcvSocketDat[5] = 16#34) AND ReadContSts THEN
  Return_Value := 'Mooving';
END_IF;

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

// Generate error
SktClose_instance(
  Execute:=Timeout_Timer.Q,
  Socket:=WkSocket);

IF SktClose_instance.DONE THEN
  Socket_Opened:=FALSE;
  ReadContSts := FALSE;
  KeepAlive_Flag:=FALSE;
  Busy := FALSE;
  Done := FALSE;
  Error := TRUE;
  Return_Value := 'Communication Error. Check the flaxiBowl Con';
END_IF;

```