Manual

FlexiBowl®

SCHNEIDER MODICON M241, M251 AND M262 PLUG-IN

ars | Feeding Industrial Robotics

INDEX

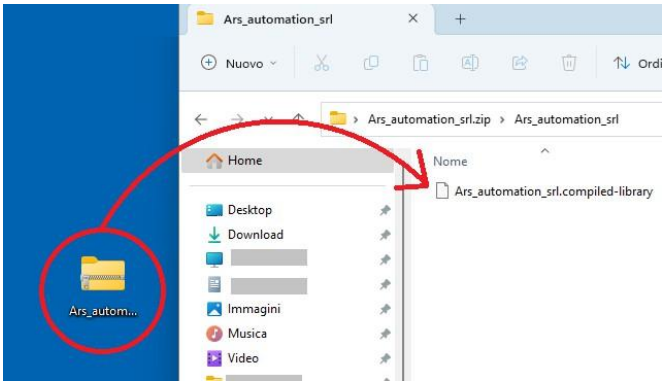
This Plugin was born with the idea of communicating quickly and securely with the FlexiBowl® via the Schneider Modicon M241, M251 and M262 Series PLCs, through the use of instructions in LD/ST language.

The Plugin developed in Machine Expert V2.1 does not require an additional license.

FlexiBowl® Plug-In

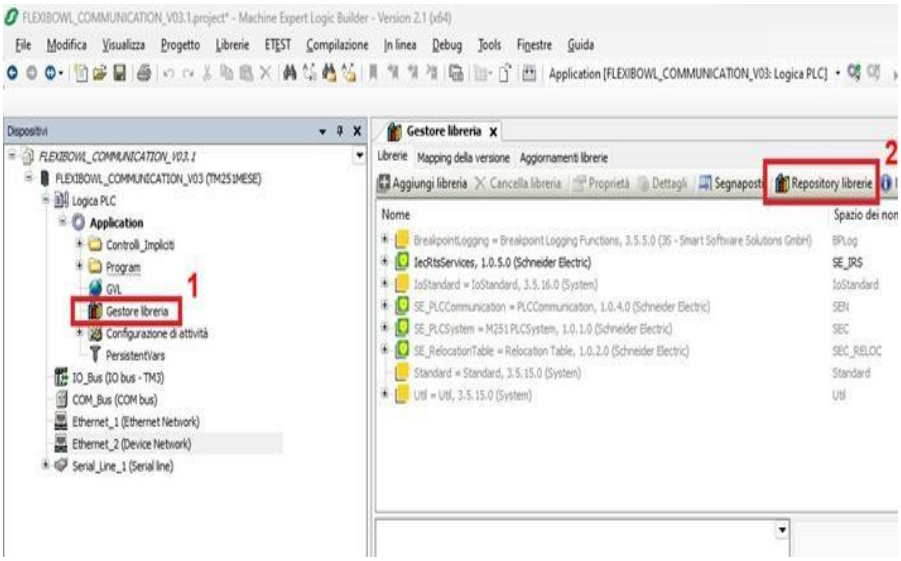
FlexiBowl by ars

STEP 1:



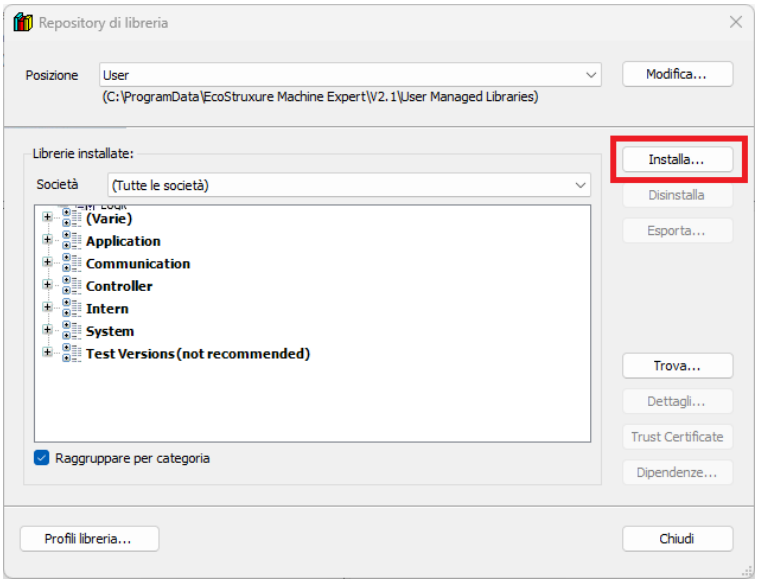
Extract the “Ars_automation_srl.compiled-library" into a desired folder.

STEP 2:



Open the "Library Repository" from your project

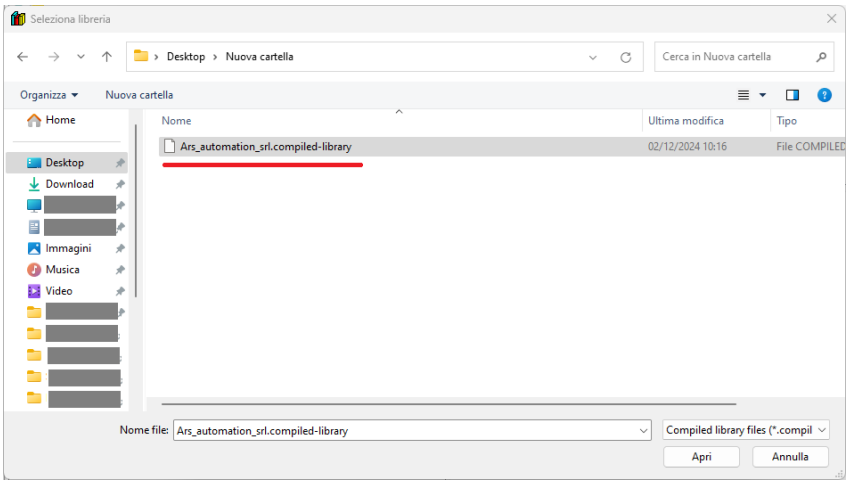
STEP 3:



Click on the install button

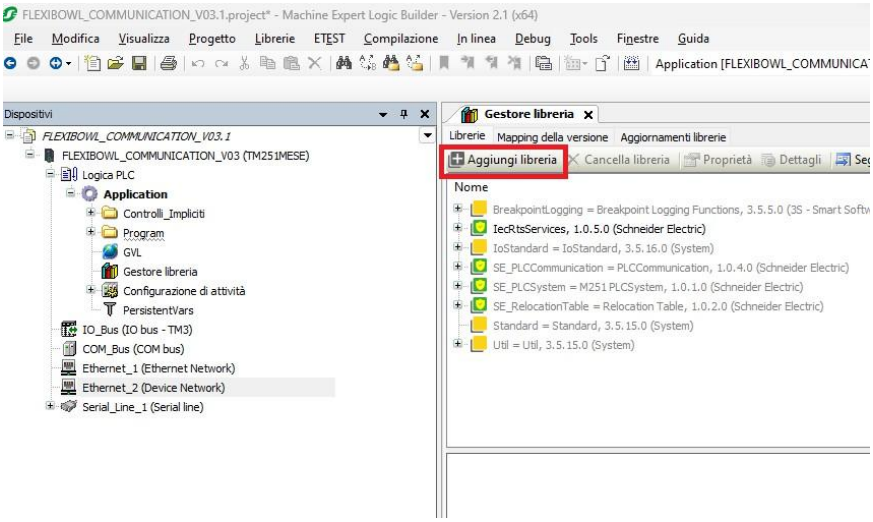
FlexiBowl by ars

STEP 4:



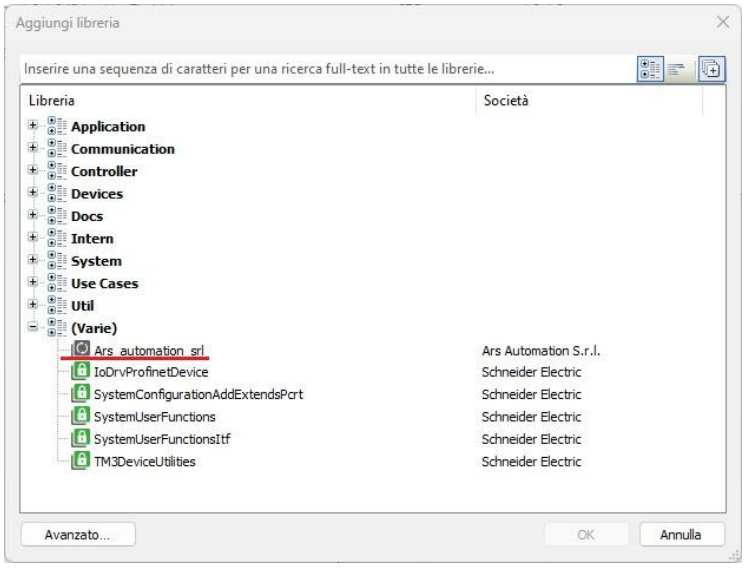
Select the file from the previously unpacked library and click on "Open". Then close the Library Repository.

STEP 5:



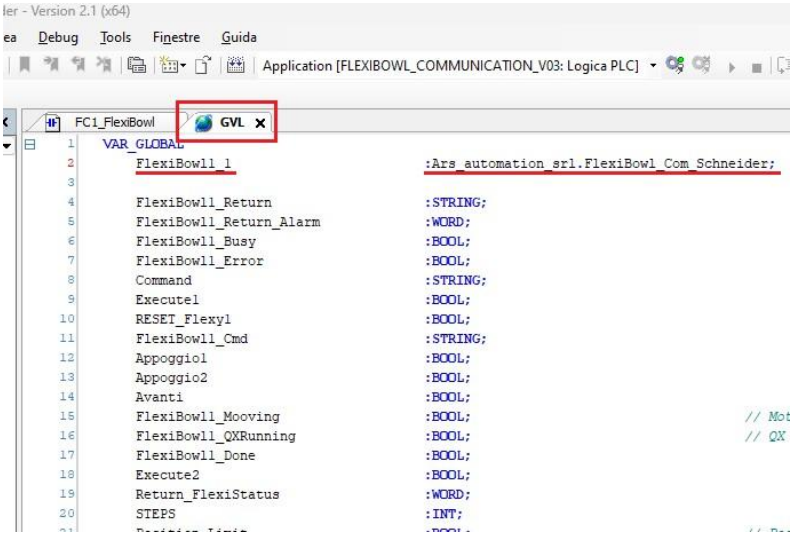
Still within your project, now click on "Add library"

STEP 6:



Expand the library tree under "(Miscellaneous)", then select "Ars_automation_srl" and click OK

STEP 7:



In global variables, declare an instance referring to the following type: "Ars_automation_srl. FlexiBowl_Com_Schneider";

STEP 8:

Populate the block inputs and outputs now according to your needs:

TAG	Type	Description
EN	IN:Bool	Enabling the block. It must always be enabled, because inside there are instructions that take care of keeping the TCP connection active
Execute	IN:Bool	Send the command present to the "CMD" input. The block only considers this input for a PLC scan cycle and if communication is established.
IPAddr	IN:String	FlexiBowl IP address. Specify the IP address of the Flexibowl. The length of the string is constrained. TCP communication instructions are at a high level and managed by the PLC's ethernet soket, so even if the FlexiBowl is located in another subnet If there is a reference gateway set up in the hardware configuration of the PLC, you can directly specify the final IP address of the FlexiBowl.

FlexiBowl by ars

STEP 8:

TAG	Type	Description
CMD	IN:String	Command in string format to be sent to the flexiBowl (see table below)
ENO	OUT:Bool	Successful block execution without any software error
Busy	OUT:Bool	QX execution by the FlexiBowl, so no other commands 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 you are not executing a command, this bit is set to FALSE. Otherwise it always remains at TRUE following a successfully completed statement 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	String back from the FlexiBowl. In the case of a QX execution, the value 'Moving' will be returned until it is completed. Once the QX statement is complete, you will find the value of the last status request to the FlexiBowl until the next run. In case of communication error, the value 'Communication Error. Check state on Network&Device '
Return_ FlexiAlarm	OUT:Word	Word file containing any error code read by the FlexiBowl driver. See the alarms table

FlexiBowl by ars

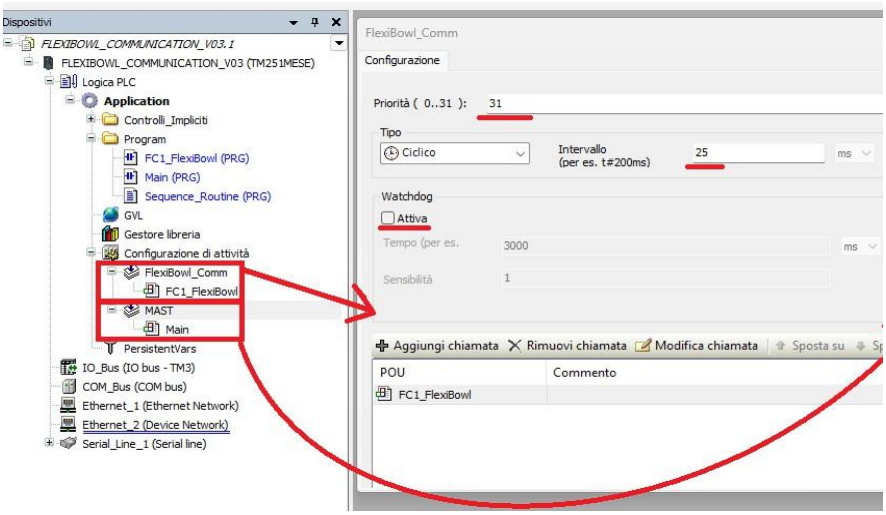
COMMAND LIST:

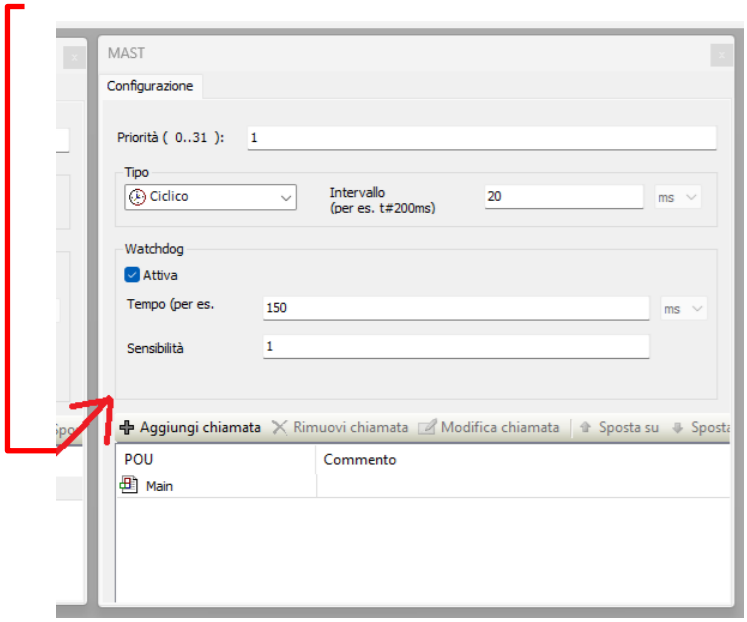
Command	Description
QX2	Move
QX3	Move-Flip
QX4	Move-Flip-Blow
QX5	Move-Blow
QX6	Shake
QX7	Light on
QX8	Light off
QX9	Flip
QX10	Blow
QX11	Quick Emptying Option
QX12	Reset Alarm

ALARM TABLE:

Hex Value	Description
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)

RECOMMENDATIONS



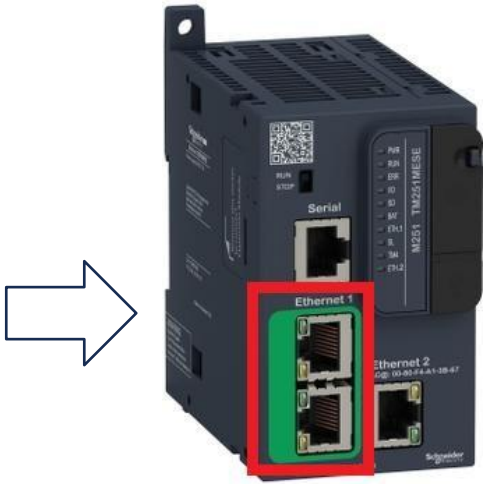


The "FlewiBowl_Com_Schneider" function block covered in this manual uses some Codesys system libraries that can have blocking effects on the execution of PLC tasks. Therefore it is strongly recommended to create a task totally dedicated to the routine that deals with communication with the FlexiBowl, with a "cyclical" execution time and minimum interval at "25 ms" with Watchdog "disabled" and the lowest possible priority, in the case above for example it has been set to "31".

This prevents unwanted shutdowns of the user program in the event of errors generated by the instructions used within the "FlewiBowl_Com_Schneider" block

RECOMMENDATIONS

If you are using a **M251MESE** PLC the Codesys communication libraries refer to the "**Ethernet 1**" port, i.e. the one in switch mode as shown in the photo. Therefore, the FlexiBowl must be connected on that port.



FlexiBowl by ars

SCRIPT:

```
// ###############################################################################################
// Set the base parameter
// Convert the IP address
syssocket.SysSockInetAddr(IPAddr,  ADR(_ipAddr.sin_addr));

// Set the IP address into array of byte
_ipAddr.sin_port := syssocket.SysSockHtons(7776);                    // FlexiBowl TCP listening port
_ipAddr.sin_family := SOCKET_AF_INET;                                // Codesys Identification number of address family


// If the connection TCP are close, then try to connect with the flexybowl
KeepAlive_Timer(IN := (((SokStatus = FLEXYSOCK_CONNECTED) OR (SokStatus = FLEXYSOCK_IDLE)) AND (NOT
Execute) AND (NOT InsideBusy) AND (NOT ReadContSts) AND (NOT ReadAlarm)),
              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 ((SokStatus = FLEXYSOCK_CONNECTED) OR (SokStatus = FLEXYSOCK_IDLE)) 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];
      IF SendSocketDat[i + 2] = 16#00 THEN
         SendSocketDat[i + 2] := 16#0D;
         EXIT;
      END_IF;
   END_FOR;
//
END_IF
```

FlexiBowl by ars

SCRIPT:

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

// If ReadContSts funtion active send 'SC' command
IF ReadContSts THEN
        _Char := 'SC';
    SendSocketDat[0] := 16#00;
    SendSocketDat[1] := 16#07;
    SendSocketDat[2] := _Char[0];
    SendSocketDat[3] := _Char[1];
    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
        _Char := 'AL';
    SendSocketDat[0] := 16#00;
    SendSocketDat[1] := 16#07;
    SendSocketDat[2] := _Char[0];
    SendSocketDat[3] := _Char[1];
    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_Trig1(CLK := (SokStatus = FLEXYSOCK_IDLE));            // Execution trigger 1
Snd_Trig2(CLK := (KeepAlive_Timer.Q));                     // Execution trigger 2

CASE SokStatus OF
// ############################## CREATING ##############################
        FLEXYSOCK_CLOSE:
                SokStatus := FLEXYSOCK_CONNECTING;

                syssocket.SysSockClose(SocketId);
                SocketId := syssocket.SysSockCreate(SOCKET_AF_INET, SOCKET_STREAM,
                        SOCKET_IPPROTO_TCP , ADR(SockCreateResult));

                SysSockIoctl(SocketId, SOCKET_FIONBIO, ADR(enable_option));

                IF (SocketId = RTS_INVALID_HANDLE) THEN
                        SokStatus := FLEXYSOCK_ERROR;
                ELSE
                        SokStatus := FLEXYSOCK_CONNECTING;
                END_IF
```

FlexiBowl® by ars

SCRIPT:

```
// ############################## CONNECTING ##############################
        FLEXYSOCK_CONNECTING:
                SockConnectResult := SysSockConnect(SocketId, ADR(_ipAddr), SIZEOF(_ipAddr));

            IF (SockConnectResult <> RTS_INVALID_HANDLE) THEN
                SokStatus := FLEXYSOCK_CONNECTED;
            ELSE
                SokStatus := FLEXYSOCK_ERROR;
            END_IF

// ############################## SENDING ##############################
FLEXYSOCK_CONNECTED, FLEXYSOCK_IDLE:
SockReceiveDone := FALSE;
IF ((Exe_trig.Q OR ((Snd_Trig1.Q OR Snd_Trig2.Q) AND ReadContSts) OR ReadAlarm)) THEN
        SockSendDone := FALSE;
        SentBytes := 0;
        SokStatus := FLEXYSOCK_SENDBUSY;
        SentBytes := syssocket.SysSockSend(SocketId, ADR(SendSocketDat), SndSize + 1, SOCKET_MSG_NONE,
ADR(SockSendResult));

        IF ((SentBytes > 0) AND (SockSendResult <> RTS_INVALID_HANDLE)) THEN
            SokStatus := FLEXYSOCK_RECEIVEREQ;
            SockSendDone := TRUE;
        END_IF;
        IF (SockSendResult = RTS_INVALID_HANDLE) THEN
            SokStatus := FLEXYSOCK_ERROR;
            SockSendDone := FALSE;
        END_IF
END_IF;

// ############################## RECEIVING ##############################
// Receive data Section
FLEXYSOCK_RECEIVEREQ:
        RcvBytes := 0;
        SokStatus := FLEXYSOCK_RECEIVBUSY;
        FOR i := 0 TO 51 DO
    RcvSocketDat[i] := 16#00;
  END_FOR;

        RcvBytes := syssocket.SysSockRecv(SocketId, ADR(RcvSocketDat), SIZEOF(RcvSocketDat),
SOCKET_MSG_NONE, ADR(iecRcvResult));

        IF ((RcvBytes > 0) AND (RcvSocketDat[0] = 16#00) AND (RcvSocketDat[1] = 16#07) AND (iecRcvResult <>
RTS_INVALID_HANDLE)) THEN
            SokStatus := FLEXYSOCK_IDLE;
            SockReceiveDone := TRUE;
        END_IF;
        IF (iecRcvResult = RTS_INVALID_HANDLE) THEN
            SokStatus := FLEXYSOCK_ERROR;
            SockReceiveDone := FALSE;
        END_IF;
END_CASE;
// ##########################################################################################

// Decoding the message from the FlaxiBowl
IF SockReceiveDone AND (NOT ReadAlarm) THEN
        FOR i := 0 TO 51 DO
            Return_Value[i] := 16#00;
        END_FOR;
  FOR i := 2 TO 52 DO
    IF (RcvSocketDat[i] = 16#0D) THEN
      EXIT;
    END_IF;
            Return_Value[i - 2] := RcvSocketDat[i];
  END_FOR;
```

FlexiBowl by ars

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'
   //                                          234 5 678
   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 SockReceiveDone 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(string_app2);   ReadAlarm :=
   FALSE;
   InsideBusy := FALSE;
   Done :=
TRUE;   END_IF;

// If 'QX' command sent active ReadContSts funtion
IF SockReceiveDone AND (RcvBytes <> 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 SockReceiveDone AND (RcvBytes <> 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 SockReceiveDone AND (RcvSocketDat[7] = 16#30) AND
   ReadContSts THEN   ReadContSts := FALSE;
END_IF;

IF KeepAlive_Flag AND (RcvBytes <> 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
SockReceiveDone)) OR   KeepAlive_Flag OR ReadAlarm),
          PT := T#5S);

// Generate error
IF Timeout_Timer.Q OR (SokStatus = FLEXYSOCK_ERROR)
   THEN   ReadContSts := FALSE;
   ReadAlarm := FALSE;
   KeepAlive_Flag :=
   FALSE;   InsideBusy :=
   FALSE;   Done := FALSE;
   Error_Com := TRUE;
   Return_Value := 'Communication Error. Check the
            flaxiBowl Con';
            syssocket.SysSockClose(SocketId);
            SokStatus := FLEXYSOCK_CLOSE;
END_IF;

Busy := InsideBusy;
```