MECADEMIC FLEXIBOWL PLUGIN

This Plugin was developed with the idea of communicating quickly and safely with FlexiBowl® through Mecademic robots by using instructions in C# or PHYTON.
The Plugin does not require any additional licence

Creating Your Own Application

To create your own application, you will need to choose a programming language, learn how SCL commands
and responses are encapsulated in UDP packets, and learn to use your programming language's interface to
the network.

UDP Packet Format

eSCL is based on MOONS''s Serial Command Language (SCL), an ASCII-based language with roots in RS-232
and RS-485 communication. eSCL drives support the full SCL and Q command sets, and utilize the speed and
reliability of Ethernet. Commands and responses are encapsulated in the payload of User Datagram Protocol
(UDP) packets, and are transmitted using standard Ethernet hardware and standard TCL/IP stacks.

Sending Commands to a Drive

An eSCL UDP packet consists of three parts, the header (binary 07), the SCL string (a sequence of ASCII encoded
characters) and the SCL terminator (ASCII carriage return, 13)
header SCL string <cr>
Example: Sending "RV"
• SCL Header = 07 (two bytes)
• R = ASCII 82
• V = ASCII 86
• <cr> (ASCII carriage return) = 13
header "RV" <cr>
0 7 82 86 13

Receiving Responses from a Drive

A typical response to "RV" would be "RV=103<cr>" which would be formatted as
header "RV=103" <cr>
0 7 82 86 61 49 48 51 13

C#.NET

The .NET languages are Microsoft's modern, object oriented Windows application building tools and
include robust Ethernet support. We present this example in C#.
Make sure your project includes this line, providing access to an Ethernet socket:

```
using System.Net.Sockets;
```

In your form header you must declare a UdpClient object and create an instance, which can be done in
the same line. The local port number is included in the "new UdpClient" call. This is the port number that will be
reserved on the PC for your application.

```
static UdpClient udpClient = new UdpClient(7777);
```

To open the connection, invoke the Connect method, specifying the drive's IP address and port number:

```
udpClient.Connect("192.168.0.130", 7775);
```

To send "RV" to the drive:

```
//create a string loaded with the SCL command
Byte[] SCLstring = Encoding.ASCII.GetBytes("RV");
// create a byte array that will be used for the actual
// transmission
Byte[] sendBytes = new Byte[SCLstring.Length + 3];
// insert opcode (07 is used for all SCL commands)
sendBytes[0] = 0;
sendBytes[1] = 7;
// copy string to the byte array
System.Array.Copy(SCLstring, 0, sendBytes, 2,
SCLstring.Length);
// insert terminator
sendBytes[sendBytes.Length - 1] = 13; // CR
// send it to the drive
udpClient.Send(sendBytes, sendBytes.Length);
```

Getting responses back from the drive in C# is a more complicated than VB6.
You have two choices: poll
for a response or create a callback function that will provide a true receive event.
Polling is easier to code but less efficient because you must either sit in a loop waiting for an expected
response or run a timer to periodically check for data coming in. Since the choice depends on your
programming style and the requirements of your application, we preset both techniques.

Polling for an incoming packet

The same UdpClient object that you use to send packets can be used to retrieve incoming responses from
the drive. The Available property will be greater than zero if a packet has been received. To retrieve a packet,
assign the Receive property to a Byte array. You must create an IPEndPoint object in order to use the Receive
property.

```
private void UDPpoll()
{
// you can call this from a timer event or a loop
if (udpClient.Available > 0) // is there a packet ready?
{

IPEndPoint RemoteIpEndPoint = new IPEndPoint(IPAddress.Any,
0);
try
{
      // Get the received packet. Receive method blocks
      // until a message returns on this socket from a
      // remote host,
      // so always check .Available to see if a packet is
      // ready.
      Byte[] receiveBytes = udpClient.Receive(ref
      RemoteIpEndPoint);
      // strip opcode
      Byte[] SCLstring = new byte[receiveBytes.Length - 2];
      for (int i = 0; i < SCLstring.Length; i++)
      SCLstring[i] = receiveBytes[i + 2];
      string returnData =
      Encoding.ASCII.GetString(SCLstring);
      AddToHistory(returnData);
      }
catch (Exception ex)
{
// put your error handler here
Console.WriteLine(ex.ToString());
}
}
}
```

EXAMPLE CLASS FLEXIBOWL PLUGIN:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.Net.Sockets;



namespace Mecademic_Plugin
{
    class Flexibowl_Plugin
    {
        UdpClient m_udpClient = new UdpClient(7777);

        public string Flexibowl(string ipAddress, string command)
        {
            string m_IpAddress = ipAddress;
            string m_command = command;

            string receiveString = "";
            string ReturnFlexibowl = "";
            int byteSent = 0;


            IPEndPoint ep = new IPEndPoint(0, 0);

            try
            {
                ep = new IPEndPoint(IPAddress.Parse(m_IpAddress), 7775);
                m_udpClient.Connect(ep);
                m_udpClient.Client.SendTimeout = 500;
                m_udpClient.Client.ReceiveTimeout = 500;
            }
            catch (Exception ex)
            {
                ReturnFlexibowl = ex.ToString();
                return ReturnFlexibowl;
            }

            string dataToSend = m_command.ToUpper();

            try
            {
                Byte[] SCLstring = Encoding.ASCII.GetBytes(dataToSend);
                Byte[] sendBytes = new Byte[SCLstring.Length + 3];
                sendBytes[0] = 0;
                sendBytes[1] = 7;
                System.Array.Copy(SCLstring, 0, sendBytes, 2,
SCLstring.Length);
                sendBytes[sendBytes.Length - 1] = 13; // CR
                byteSent = m_udpClient.Send(sendBytes, sendBytes.Length);
                var receivedData = m_udpClient.Receive(ref ep);
                receiveString = Encoding.ASCII.GetString(receivedData);
```

```
if ((receiveString.Contains("%")) && (dataToSend.Contains("Q")))
                {
                    bool moving = true;
                    while (moving == true)
                    {
                        SCLstring = Encoding.ASCII.GetBytes("RS");
                        sendBytes = new Byte[SCLstring.Length + 3];
                        sendBytes[0] = 0;
                        sendBytes[1] = 7;
                        System.Array.Copy(SCLstring, 0, sendBytes, 2,
SCLstring.Length);
                        sendBytes[sendBytes.Length - 1] = 13; // CR
                        byteSent = m_udpClient.Send(sendBytes,
sendBytes.Length);
                        receivedData = m_udpClient.Receive(ref ep);
                        receiveString =
Encoding.ASCII.GetString(receivedData);
                        if (receiveString.Contains("F"))
                            moving = true;
                        else
                            moving = false;
                        System.Threading.Thread.Sleep(20);
                    }
                    System.Threading.Thread.Sleep(100);
                    ReturnFlexibowl = "Done";
                    return ReturnFlexibowl;
                }
                else
                {
                    SCLstring = new Byte[receivedData.Length - 3];
                    System.Array.Copy(receivedData, 2, SCLstring, 0,
SCLstring.Length);
                    receiveString = Encoding.ASCII.GetString(SCLstring);
                    ReturnFlexibowl = receiveString;
                }
                return ReturnFlexibowl;

            }
            catch (Exception ex)
            {
                ReturnFlexibowl = ex.ToString();
                return ReturnFlexibowl;
            }
        }

        public void Flexibowl_Close()
        {
            m_udpClient.Dispose();
        }
    }
}
```

how to use the class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Mecademic_Plugin
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("////////////FLEXIBOWL PLUGIN EXAMPLE
MECADEMIC/////////////");

            string retunFlbString = "";

            Flexibowl_Plugin Fb = new Flexibowl_Plugin();

            //ALLARM
            retunFlbString = Fb.Flexibowl("192.168.0.161", "AL");
            Console.WriteLine(retunFlbString);

            //MOVE
            retunFlbString = Fb.Flexibowl("192.168.0.161", "QX2");
            Console.WriteLine(retunFlbString);

            //MOVE-FLIP
            retunFlbString = Fb.Flexibowl("192.168.0.161", "QX3");
            Console.WriteLine(retunFlbString);

            //CLOSE CONNECTION
            Fb.Flexibowl_Close();

            Console.ReadLine();
        }
    }
}
```

Command:

Action	Description
MOVE	Moves the feeder the current parameters.
MOVE-FLIP	Moves the feeder and activates Flip simultaneously
MOVE-BLOW-FLIP	Moves the feeder and activates Flip and blow simultaneously
MOVE-BLOW	Moves the feeder and activates Flip simultaneously
SHAKE	Shakes the feeder with the current parameters
LIGHT ON	Light on
LIGHT OFF	Light off
FLIP	Flip
BLOW	Blow
QUICK_EMPTING	Quick Emptying Option
RESET_ALARM	Reset Alarm and enable the motor

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

EXAMPLE CLASS FLEXIBOWL PLUGIN:

```
import socket
from time import sleep
import sys

def in_allarm(ip):
    assert type(ip) is str
    TCP_IP = ip
    TCP_PORT = 7776
    BUFFER_SIZE = 1024
    command ="AL"
    #invio il messaggio al flexibowl
    MESSAGE = chr(0)+chr(7)+command+chr(13)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        s.connect((TCP_IP, TCP_PORT))
        s.send(MESSAGE)
        data = s.recv(BUFFER_SIZE)
        print("Message send: " + MESSAGE)
        print("Message recive: " + data)
        sleep(0.1) # Time in seconds.
    except :
        print("Not Connected1")
        return false
    my_hexdata = data[5:None]
    print(my_hexdata)
    scale= 16 ## equals to hexadecimal
    num_of_bits = 16
    a=bin(int(my_hexdata, scale))[2:].zfill(num_of_bits)
    print(a)
    error_decimal=int(a, 2)
    if(a>0):
        #sono in eroore
        return (False)
        s.close()
    else:
        return(True)
        s.close()
```

```
def move_flb1(ip,command):
    assert type(command) is str
    assert type(ip) is str
    TCP_IP = ip
    TCP_PORT = 7776
    BUFFER_SIZE = 1024


    #decodifico il messaggio
    if(command=="MOVE"):
        command="QX2"
    elif (command=="MOVE FLIP"):
        command="QX3"
    elif (command=="MOVE BLOW FLIP"):
        command="QX4"
    elif (command=="MOVE BLOW"):
        command="QX5"
    elif (command=="SHAKE"):
        command="QX6"
    elif (command=="LIGHT ON"):
        command="QX7"
    elif (command=="LIGHT OFF"):
        command="QX8"
    elif (command=="FLIP"):
        command="QX10"
    elif (command=="BLOW"):
        command="QX9"
    elif (command=="QUICK EMPTY OPTION"):
        command="QX11"
    else:
        command="QX60"  #comando che non esiste

    #invio il messaggio al flexibowl
    MESSAGE = chr(0)+chr(7)+command+chr(13)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(2)

    try:
        s.connect((TCP_IP, TCP_PORT))
        s.send(MESSAGE)
        data = s.recv(BUFFER_SIZE)
        print("Message send: " + MESSAGE)
        print("Message recive: " + data)
        sleep(0.1) # Time in seconds.
        if("%" in data ):
            print("Wait move2")
            moving=1
            while True:
```

```
if(command=="QX11")or (command=="QX10")or(command=="QX4")or (command=="QX3")
:
                print("Wait busy ")
                sleep(0.1)
                MESSAGE = chr(0)+chr(7)+"IO"+chr(13)
                s.send(MESSAGE)
                data = s.recv(BUFFER_SIZE)
                print data
                moving=data[12:-1]
                print moving
                if int(moving) == 1:
                    sleep(0.1)
                    break
            else:
                MESSAGE = chr(0)+chr(7)+"SC"+chr(13)
                s.send(MESSAGE)
                data = s.recv(BUFFER_SIZE)
                moving=data[7:-2]

                if int(moving) == 0:
                    sleep(0.1)
                    break
            sleep(0.1) # Time in seconds.

        return(True)
    else:
        return(False)
    s.close()
except :
    print("Not Connected2")
    return False
```