User Guide for working with Connected Component Workbench TCP/IP

Micro820 PLC

Contents

1 Electrical connections

Two connections are required for the Micro 820 PLC:

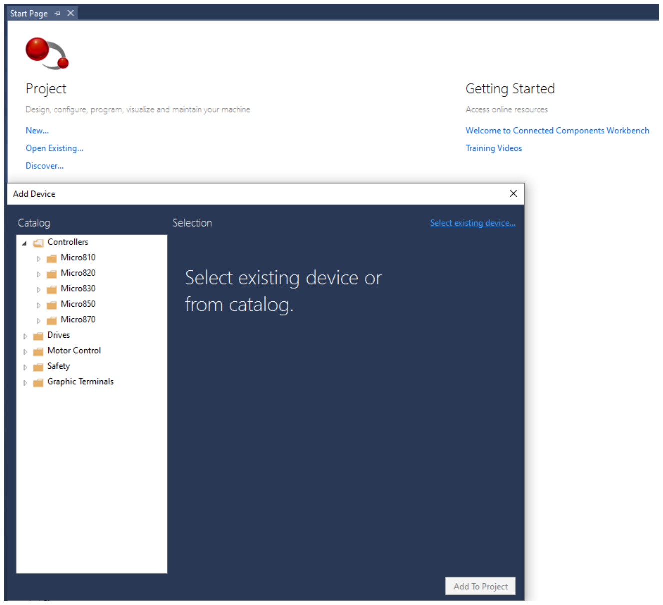
- a 24V power connection;
- a line of communication with the Meca500 and a computer via an Ethernet switch.

Ensure the above are properly connected before proceeding to make a connection between CCW, the PLC and the Meca500.

2 Software

Connected Component Workbench (CCW) is an application development software for a range of Rockwell PLCs. There is a free version called the standard edition and a developer edition that requires a paid license. Both editions are available on the Rockwell website[1].

When a new project is created in CCW a pop-up will appear where a controller must be selected as shown in Figure 1. Configure as required for your application.



Select existing device or from catalog.

Figure 1 – PLC and I/O configuration

[1] At the time of this writing the latest version is 12.00.00

After configuring your hardware, you will be greeted with a blank tree. In order to write logic for your application, right click on 'Programs' and add the desired language as highlighted in Figure 2.

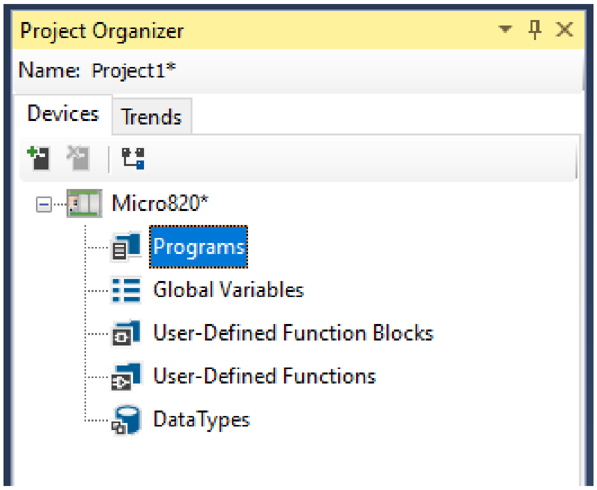


Figure 2 – Main Tree Side Bar

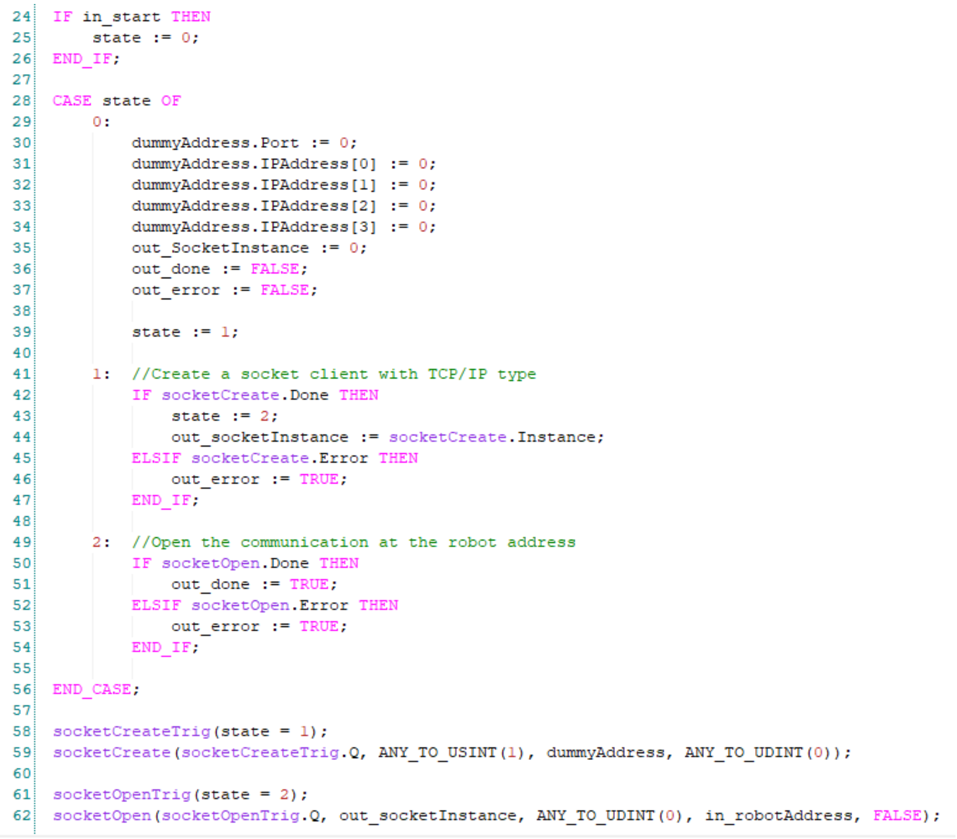
Once you are on your first program page you are ready to program TCP/IP communication with the Meca500.

3 Programming

This section will demonstrate different examples of commands that can be send to a Mecademic robot using CCW. The communication will be done using TCP/IP.

3.1 Connect, Activate and Home

Establishing a TCP/IP connection between the meca500 and the Micrologix PLC require some specific steps. The first would be to create and open a socket. The ‘Function Block’ ‘_MxConnect’ shown in Figure 3 demonstrate a typical way of doing this.

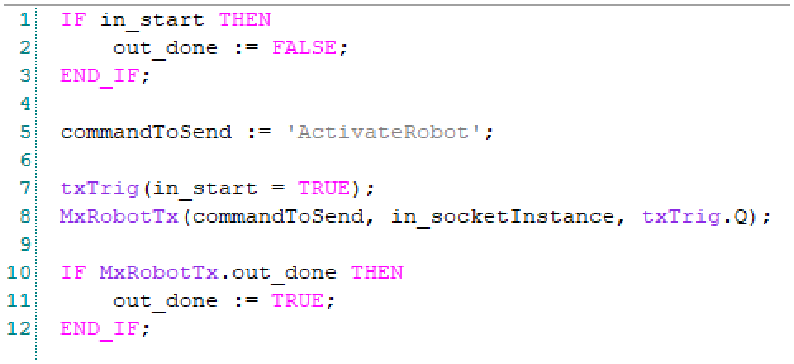


```
IF in_start THEN
    state := 0;
END_IF;

CASE state OF
    0:
        dummyAddress.Port := 0;
        dummyAddress.IPAddress[0] := 0;
        dummyAddress.IPAddress[1] := 0;
        dummyAddress.IPAddress[2] := 0;
        dummyAddress.IPAddress[3] := 0;
        out_SocketInstance := 0;
        out_done := FALSE;
        out_error := FALSE;

        state := 1;

    1:  //Create a socket client with TCP/IP type
        IF socketCreate.Done THEN
            state := 2;
            out_socketInstance := socketCreate.Instance;
        ELSIF socketCreate.Error THEN
            out_error := TRUE;
        END_IF;

    2:  //Open the communication at the robot address
        IF socketOpen.Done THEN
            out_done := TRUE;
        ELSIF socketOpen.Error THEN
            out_error := TRUE;
        END_IF;

END_CASE;

socketCreateTrig(state = 1);
socketCreate(socketCreateTrig.Q, ANY_TO_USINT(1), dummyAddress, ANY_TO_UDINT(0));

socketOpenTrig(state = 2);
socketOpen(socketOpenTrig.Q, out_socketInstance, ANY_TO_UDINT(0), in_robotAddress, FALSE);
```

Figure 3 – First Line for TCP/IP Communication

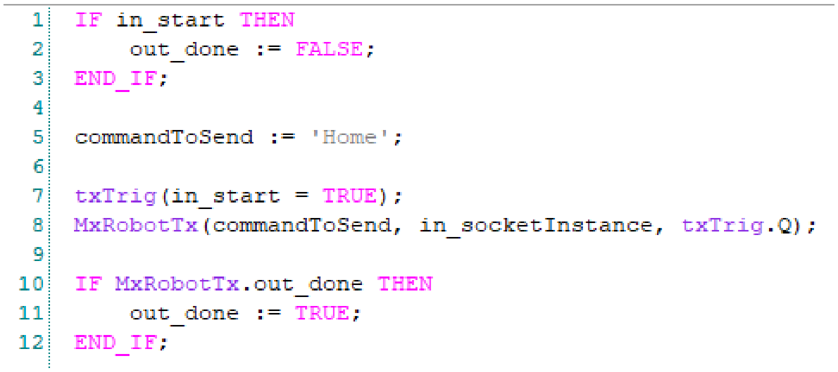
When the block is first called, all parameters are set to 0. Next, a communication socket is created. If the creation of the socket was successful, the socket is opened and if the opening was successful a “done” Boolean is returned to the main routine.

We now have a connection between the robot and the controller. Before sending move commands we need to activate and home the robot. We will do this by sending those commands with the help of the function blocks “_MxActivate” and “_MxHome” that are respectively shown in Figure 4 and Figure 5.



```
IF in_start THEN
    out_done := FALSE;
END_IF;

commandToSend := 'ActivateRobot';

txTrig(in_start = TRUE);
MxRobotTx(commandToSend, in_socketInstance, txTrig.Q);

IF MxRobotTx.out_done THEN
    out_done := TRUE;
END_IF;
```

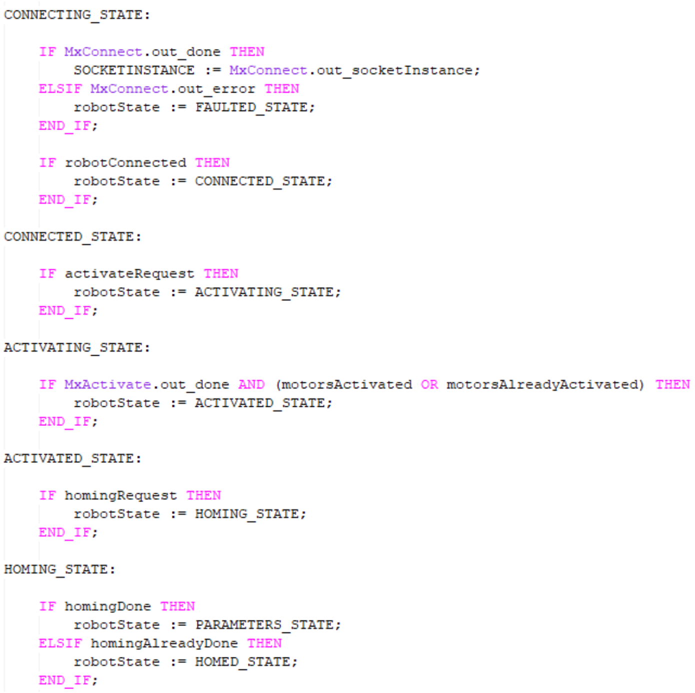
Figure 4 – “ActivateRobot” Command



```
IF in_start THEN
    out_done := FALSE;
END_IF;

commandToSend := 'Home';

txTrig(in_start = TRUE);
MxRobotTx(commandToSend, in_socketInstance, txTrig.Q);

IF MxRobotTx.out_done THEN
    out_done := TRUE;
END_IF;
```

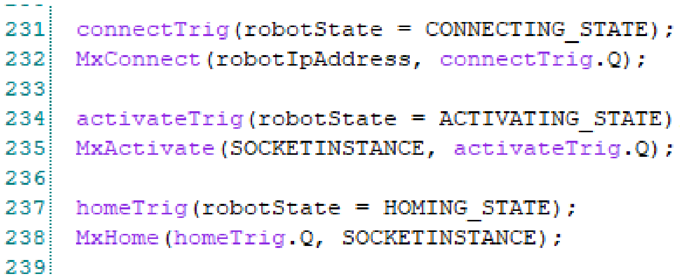
Figure 5 – “Home” Command

In Figure 6 and below, is the Main Routine where the logic of calling the previous functions are shown.



```
CONNECTING_STATE:

    IF MxConnect.out_done THEN
        SOCKETINSTANCE := MxConnect.out_socketInstance;
    ELSIF MxConnect.out_error THEN
        robotState := FAULTED_STATE;
    END_IF;

    IF robotConnected THEN
        robotState := CONNECTED_STATE;
    END_IF;

CONNECTED_STATE:

    IF activateRequest THEN
        robotState := ACTIVATING_STATE;
    END_IF;

ACTIVATING_STATE:

    IF MxActivate.out_done AND (motorsActivated OR motorsAlreadyActivated) THEN
        robotState := ACTIVATED_STATE;
    END_IF;

ACTIVATED_STATE:

    IF homingRequest THEN
        robotState := HOMING_STATE;
    END_IF;

HOMING_STATE:

    IF homingDone THEN
        robotState := PARAMETERS_STATE;
    ELSIF homingAlreadyDone THEN
        robotState := HOMED_STATE;
    END_IF;
```

Figure 6 – Main Routine Sequence



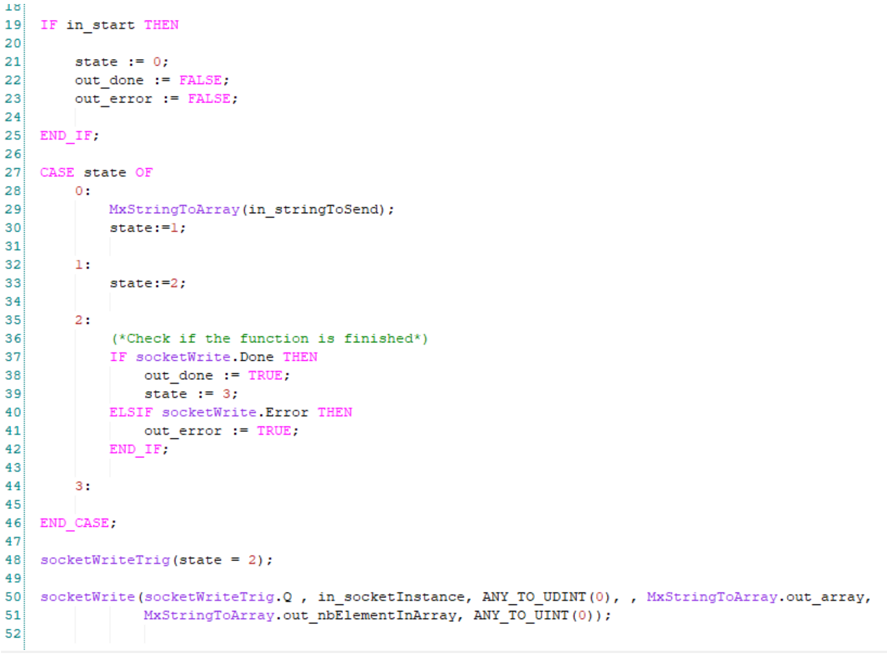
```
connectTrig(robotState = CONNECTING_STATE);
MxConnect(robotIpAddress, connectTrig.Q);

activateTrig(robotState = ACTIVATING_STATE);
MxActivate(SOCKETINSTANCE, activateTrig.Q);

homeTrig(robotState = HOMING_STATE);
MxHome(homeTrig.Q, SOCKETINSTANCE);

```

Figure 7 – Main Routine Function Blocks

At the time of writing, the management of the robot states are not yet finished in the "Mecademic_CCW_Demo_V0_16_0" project. However, the way these steps are managed is there and can be expanded in a future version.

3.2 Sending Commands

The Meca500 accepts commands in the form of ASCII strings. We can send those strings directly to the robot with the help of the "_MxRobotTx" function block shown in Figure 8.



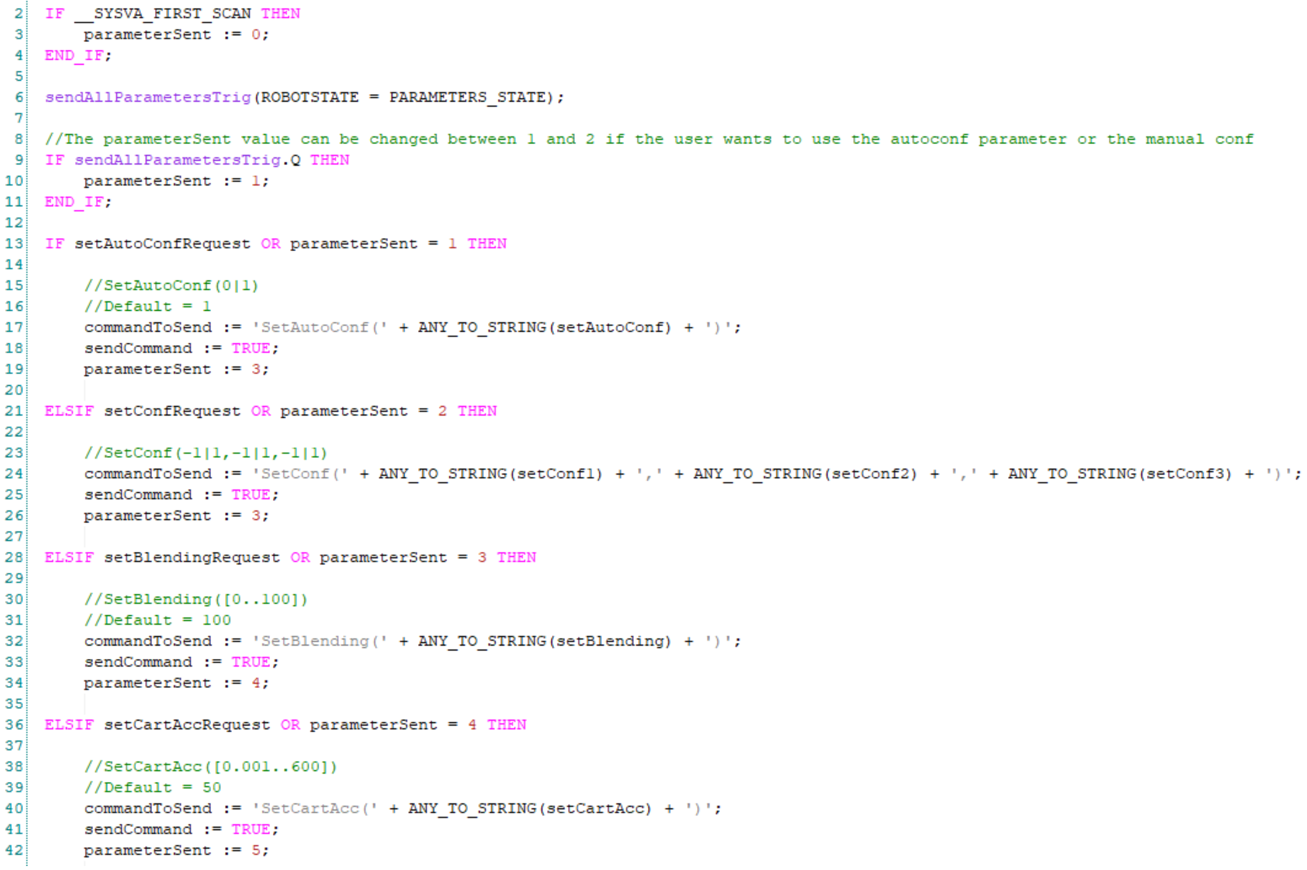
```

IF in_start THEN

    state := 0;
    out_done := FALSE;
    out_error := FALSE;

END_IF;

CASE state OF
    0:
        MxStringToArray(in_stringToSend);
        state:=1;

    1:
        state:=2;

    2:
        (*Check if the function is finished*)
        IF socketWrite.Done THEN
            out_done := TRUE;
            state := 3;
        ELSIF socketWrite.Error THEN
            out_error := TRUE;
        END_IF;

    3:

END_CASE;

socketWriteTrig(state = 2);

socketWrite(socketWriteTrig.Q , in_socketInstance, ANY_TO_UDINT(0), , MxStringToArray.out_array,
            MxStringToArray.out_nbElementInArray, ANY_TO_UINT(0));

```

Figure 8 – "_MxRobotTx" Function Block

To use this function block, a String to send must be set before using a rising edge trigger. When the block is called the "_MxStringArray" function is called, this will transform the string to an array that can then be send to the robot with the built function "SocketWrite".

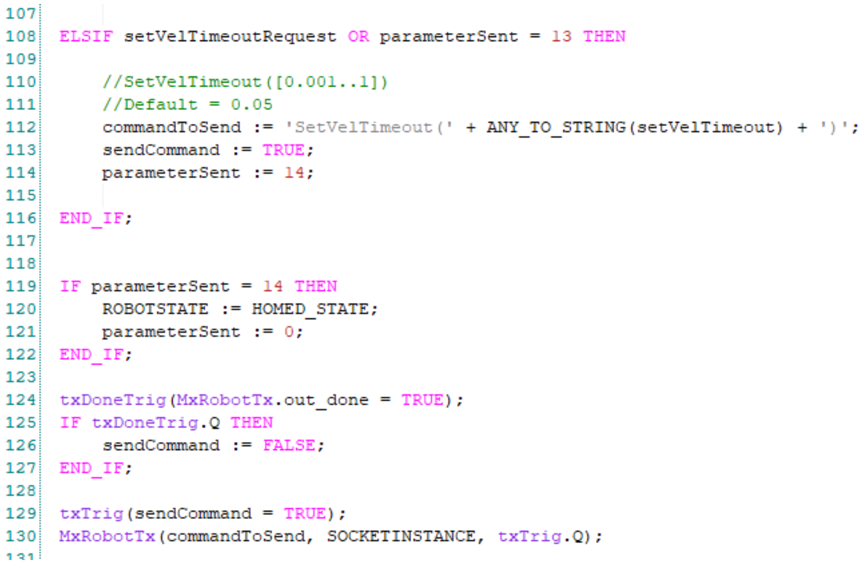
There are certain commands we want to send to the Meca500 for initialization. Such commands include 'ResetError', 'ActivateRobot', 'Home' as well as others to set reference frames, speeds and accelerations. In general, we only want to send these commands once, and we want to send them before any other commands at every power up.

Figure 9 and Figure 10 shows how the parameters are sent to the robot. To know if the robot needs to have all the parameters resent, we look at the response we get when we send the home command. This logic is shown in Figure 11.



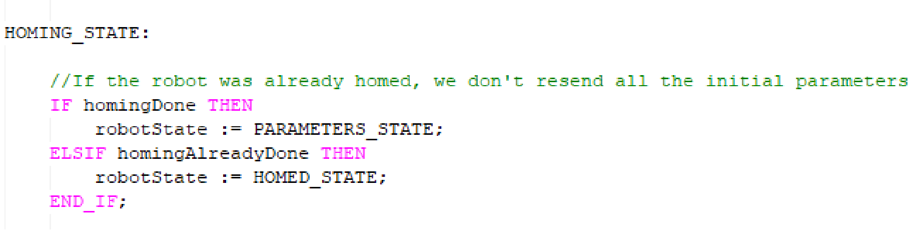
```
IF __SYSVA_FIRST_SCAN THEN
    parameterSent := 0;
END_IF;

sendAllParametersTrig(ROBOTSTATE = PARAMETERS_STATE);

//The parameterSent value can be changed between 1 and 2 if the user wants to use the autoconf parameter or the manual conf
IF sendAllParametersTrig.Q THEN
    parameterSent := 1;
END_IF;

IF setAutoConfRequest OR parameterSent = 1 THEN

    //SetAutoConf(0|1)
    //Default = 1
    commandToSend := 'SetAutoConf(' + ANY_TO_STRING(setAutoConf) + ')';
    sendCommand := TRUE;
    parameterSent := 3;

ELSIF setConfRequest OR parameterSent = 2 THEN

    //SetConf(-1|1,-1|1,-1|1)
    commandToSend := 'SetConf(' + ANY_TO_STRING(setConf1) + ',' + ANY_TO_STRING(setConf2) + ',' + ANY_TO_STRING(setConf3) + ')';
    sendCommand := TRUE;
    parameterSent := 3;

ELSIF setBlendingRequest OR parameterSent = 3 THEN

    //SetBlending([0..100])
    //Default = 100
    commandToSend := 'SetBlending(' + ANY_TO_STRING(setBlending) + ')';
    sendCommand := TRUE;
    parameterSent := 4;

ELSIF setCartAccRequest OR parameterSent = 4 THEN

    //SetCartAcc([0.001..600])
    //Default = 50
    commandToSend := 'SetCartAcc(' + ANY_TO_STRING(setCartAcc) + ')';
    sendCommand := TRUE;
    parameterSent := 5;
```

Figure 9 – Parameters program



```

ELSIF setVelTimeoutRequest OR parameterSent = 13 THEN

    //SetVelTimeout([0.001..1])
    //Default = 0.05
    commandToSend := 'SetVelTimeout(' + ANY_TO_STRING(setVelTimeout) + ')';
    sendCommand := TRUE;
    parameterSent := 14;

END_IF;


IF parameterSent = 14 THEN
    ROBOTSTATE := HOMED_STATE;
    parameterSent := 0;
END_IF;

txDoneTrig(MxRobotTx.out_done = TRUE);
IF txDoneTrig.Q THEN
    sendCommand := FALSE;
END_IF;

txTrig(sendCommand = TRUE);
MxRobotTx(commandToSend, SOCKETINSTANCE, txTrig.Q);
```

Figure 10 – Parameters Send Trigger

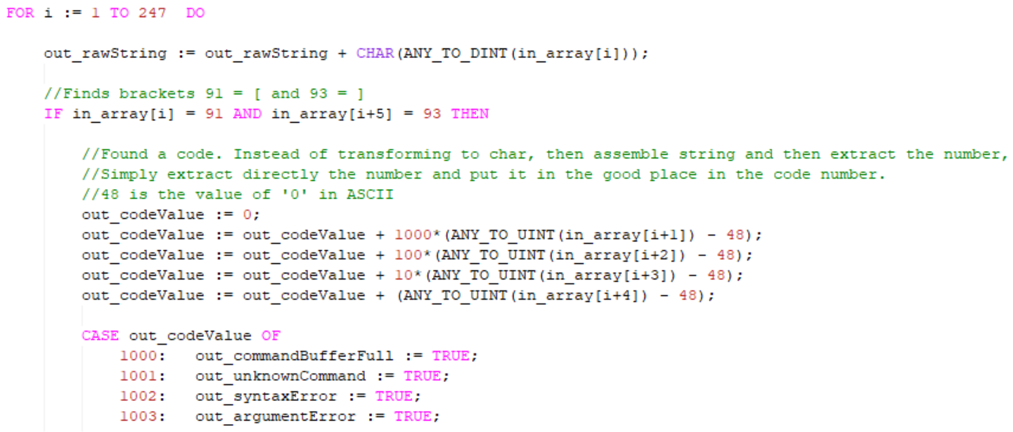


```
HOMING_STATE:

    //If the robot was already homed, we don't resend all the initial parameters
    IF homingDone THEN
        robotState := PARAMETERS_STATE;
    ELSIF homingAlreadyDone THEN
        robotState := HOMED_STATE;
    END_IF;
```

Figure 11 – Parameters program

3.3 Receiving Feedback

To receive messages from the robot we are using the built-in socket read function. The function is called once every 10ms. If messages from the robot are received faster than that, the controller will store them in an internal buffer. For every message received, the “CodeExtrator” function isolates the 4 digits ID of the message and toggles the associated Boolean variable. The logic that isolates the ID code is shown in Figure 12. If the response contains a variable value like a robot status or a robot pose, the code extractor will output the variable string.

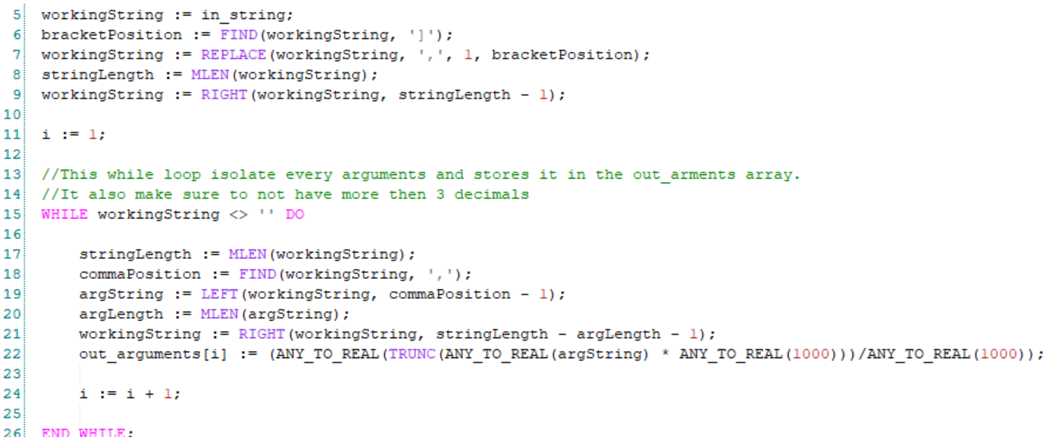


```
FOR i := 1 TO 247  DO

    out_rawString := out_rawString + CHAR(ANY_TO_DINT(in_array[i]));

    //Finds brackets 91 = [ and 93 = ]
    IF in_array[i] = 91 AND in_array[i+5] = 93 THEN

        //Found a code. Instead of transforming to char, then assemble string and then extract the number,
        //Simply extract directly the number and put it in the good place in the code number.
        //48 is the value of '0' in ASCII
        out_codeValue := 0;
        out_codeValue := out_codeValue + 1000*(ANY_TO_UINT(in_array[i+1]) - 48);
        out_codeValue := out_codeValue + 100*(ANY_TO_UINT(in_array[i+2]) - 48);
        out_codeValue := out_codeValue + 10*(ANY_TO_UINT(in_array[i+3]) - 48);
        out_codeValue := out_codeValue + (ANY_TO_UINT(in_array[i+4]) - 48);

        CASE out_codeValue OF
            1000:   out_commandBufferFull := TRUE;
            1001:   out_unknownCommand := TRUE;
            1002:   out_syntaxError := TRUE;
            1003:   out_argumentError := TRUE;
```

Figure 12 – “Code Extractor” Switch Case

3.4 Receiving variables

When a message is received, if it contains a variable information like a robot status or a joint value, those arguments needs to be isolated. To accomplish this, the “_MxArgumentsFinder” function was written.

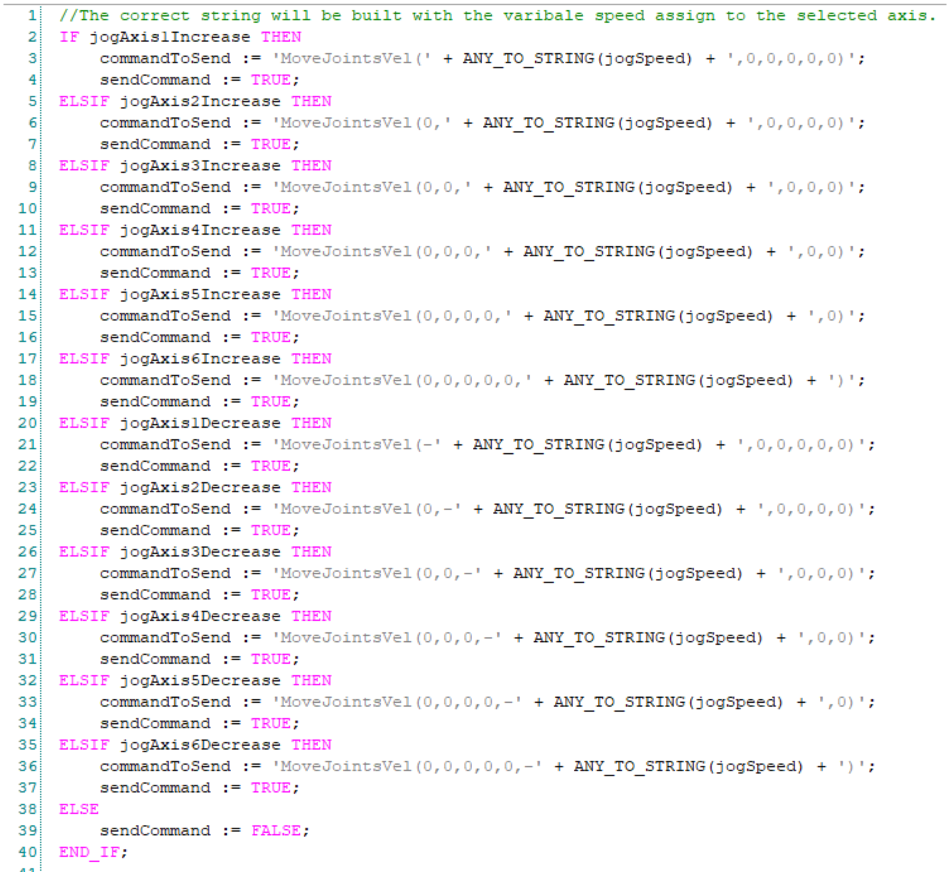


```
workingString := in_string;
bracketPosition := FIND(workingString, ']');
workingString := REPLACE(workingString, ',', 1, bracketPosition);
stringLength := MLEN(workingString);
workingString := RIGHT(workingString, stringLength - 1);

i := 1;

//This while loop isolate every arguments and stores it in the out_arments array.
//It also make sure to not have more then 3 decimals
WHILE workingString <> '' DO

    stringLength := MLEN(workingString);
    commaPosition := FIND(workingString, ',');
    argString := LEFT(workingString, commaPosition - 1);
    argLength := MLEN(argString);
    workingString := RIGHT(workingString, stringLength - argLength - 1);
    out_arguments[i] := (ANY_TO_REAL(TRUNC(ANY_TO_REAL(argString) * ANY_TO_REAL(1000)))/ANY_TO_REAL(1000));

    i := i + 1;

END_WHILE;
```

Figure 13 – Scan Block for Variables

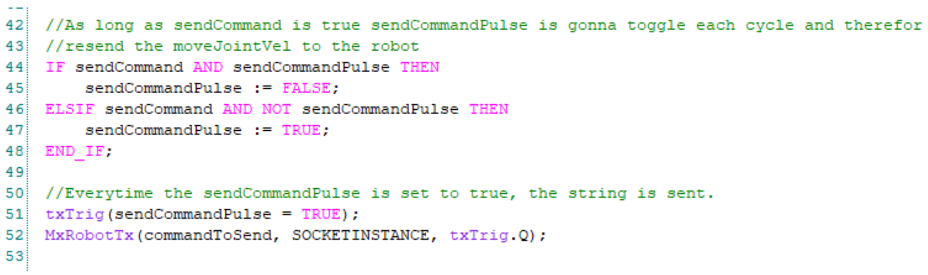
3.5 Jog Menu

Before using the jog menu, you should have already sent the 'ResetError', 'ActivateRobot' and 'Home' commands to the Meca500. These commands are the basic start commands needed to be executed by the robot prior to use.



```
//The correct string will be built with the varibale speed assign to the selected axis.
IF jogAxis1Increase THEN
    commandToSend := 'MoveJointsVel(' + ANY_TO_STRING(jogSpeed) + ',0,0,0,0,0)';
    sendCommand := TRUE;
ELSIF jogAxis2Increase THEN
    commandToSend := 'MoveJointsVel(0,' + ANY_TO_STRING(jogSpeed) + ',0,0,0,0)';
    sendCommand := TRUE;
ELSIF jogAxis3Increase THEN
    commandToSend := 'MoveJointsVel(0,0,' + ANY_TO_STRING(jogSpeed) + ',0,0,0)';
    sendCommand := TRUE;
ELSIF jogAxis4Increase THEN
    commandToSend := 'MoveJointsVel(0,0,0,' + ANY_TO_STRING(jogSpeed) + ',0,0)';
    sendCommand := TRUE;
ELSIF jogAxis5Increase THEN
    commandToSend := 'MoveJointsVel(0,0,0,0,' + ANY_TO_STRING(jogSpeed) + ',0)';
    sendCommand := TRUE;
ELSIF jogAxis6Increase THEN
    commandToSend := 'MoveJointsVel(0,0,0,0,0,' + ANY_TO_STRING(jogSpeed) + ')';
    sendCommand := TRUE;
ELSIF jogAxis1Decrease THEN
    commandToSend := 'MoveJointsVel(-' + ANY_TO_STRING(jogSpeed) + ',0,0,0,0,0)';
    sendCommand := TRUE;
ELSIF jogAxis2Decrease THEN
    commandToSend := 'MoveJointsVel(0,-' + ANY_TO_STRING(jogSpeed) + ',0,0,0,0)';
    sendCommand := TRUE;
ELSIF jogAxis3Decrease THEN
    commandToSend := 'MoveJointsVel(0,0,-' + ANY_TO_STRING(jogSpeed) + ',0,0,0)';
    sendCommand := TRUE;
ELSIF jogAxis4Decrease THEN
    commandToSend := 'MoveJointsVel(0,0,0,-' + ANY_TO_STRING(jogSpeed) + ',0,0)';
    sendCommand := TRUE;
ELSIF jogAxis5Decrease THEN
    commandToSend := 'MoveJointsVel(0,0,0,0,-' + ANY_TO_STRING(jogSpeed) + ',0)';
    sendCommand := TRUE;
ELSIF jogAxis6Decrease THEN
    commandToSend := 'MoveJointsVel(0,0,0,0,0,-' + ANY_TO_STRING(jogSpeed) + ')';
    sendCommand := TRUE;
ELSE
    sendCommand := FALSE;
END_IF;
```

Figure 14 – Jog Routine



```
//As long as sendCommand is true sendCommandPulse is gonna toggle each cycle and therefor
//resend the moveJointVel to the robot
IF sendCommand AND sendCommandPulse THEN
    sendCommandPulse := FALSE;
ELSIF sendCommand AND NOT sendCommandPulse THEN
    sendCommandPulse := TRUE;
END_IF;

//Everytime the sendCommandPulse is set to true, the string is sent.
txTrig(sendCommandPulse = TRUE);
MxRobotTx(commandToSend, SOCKETINSTANCE, txTrig.Q);
```

Figure 15 – Jog Routine

The logic of the Jog Routine is really simple. First, the correct string is built with the variable speed entered by the user in the "jogspeed" variable. Then, a toggle, therefor a rising edge, happens continuously as long as the job request for a specific axis is present.