UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

FINAL THESIS (TFE)

Bachelor's degree in Industrial Electronics and Automatic Control Engineering

DEVELOPMENT OF A FUNCTION BLOCK LIBRARY TO COMMAND OMRON COLLABORATIVE ROBOT FROM AN EXTERNAL CPU

Thesis and Annexes

Author: SÁNCHEZ BOLI, RUBÉN
Director: GÁMIZ CARO, JAVIER FRANCISCO
Summons: June 2021

Abstract

Collaborative robots are designed to be easily programmed by non-expert operators, but when the application requires the robot to be integrated and communicate with the rest of the machine, being the robot commanded by external controllers, the programming becomes more complex hence certain expertise and knowledge in robot programming is required. The target of this project is providing to non-skilled operators, a library with a set of Function Blocks to command the Omron collaborative robot from an external Omron controller without needs to learn how to build complex TCP/IP (*Transmission Control Protocol / Internet Protocol*) communication frames. The library is designed following the international standard IEC-61131 for Programmable Controllers and compliant PLCopen guidelines (worldwide association for industrial programming control).

In this thesis, it is described the robot software program, the protocol to send commands from external devices to the robot, the algorithm inside the Function Blocks, an example of Pick and Place sequence program, and a graphical interface for touch-screen to use the library and control the robot defining the parameters through drop-down lists and data entry fields, facilitating the user to use the Function Blocks. After the pertinent tests with real hardware and once the tests for debugging and implementation of functionality improvements have been completed, a package of Function Blocks has been obtained whose input parameters have a predefined range that is verified internally before sending the command to ensure this is correct and will be executed by the robot.

Resum

Els robots col·laboratius estan dissenyats per a ser programats fàcilment per operadors no experts, però quan l'aplicació requereix que el robot s’integri i comuniqui amb la resta de la màquina, en ser el robot comandat per controladors externs, la programació es torna més complexa, per la qual cosa es requereix una certa experiència i coneixement en programació de robots. L'objectiu d'aquest projecte és proporcionar als operadors no qualificats una llibreria amb un conjunt de Blocs de Funció per a controlar el robot col·laboratiu Omron des d'un controlador extern Omron, sense necessitat d'aprendre a construir complexes trames de comunicacions TCP/IP (*Transmission Control Protocol / Internet Protocol*). La llibreria està dissenyada seguint l'estàndard internacional IEC-61131 per a Controladors Programables i les directrius de PLCopen (associació mundial per al control de programació industrial).

En aquesta tesi, es descriu la programació del programari del robot, el protocol per a enviar comandos des de dispositius externs al robot, l’algoritme dels Blocs de Funció, un exemple de programa de seqüència *Pick and Place*, i una interfície gràfica per a pantalla tàctil per a usar la llibreria i controlar el robot definint els paràmetres a través de llistes desplegables i camps d'entrada de dades, facilitant així a l'usuari l'ús dels Blocs de Funció. Una vegada realitzades les proves pertinents amb maquinari real i finalitzades les proves de depuració i implementació de millores de funcionalitat, s'ha obtingut un paquet de Blocs de Funció on els paràmetres d'entrada tenen un rang predefinit que es verifica internament abans d'enviar el comando per a assegurar que aquest és correcte i serà executat pel robot.

Resumen

Los robots colaborativos están diseñados para ser programados fácilmente por operadores no expertos, pero cuando la aplicación requiere que el robot se integre y comunique con el resto de la máquina, al ser el robot comandado por controladores externos, la programación se vuelve más compleja, por lo que se requiere cierta experiencia y conocimiento en programación de robots. El objetivo de este proyecto es proporcionar a los operadores no calificados una librería con un conjunto de Bloques de Función para controlar el robot colaborativo Omron desde un controlador externo Omron, sin necesidad de aprender a construir complejas tramas de comunicaciones TCP/IP (*Transmission Control Protocol / Internet Protocol*). La librería está diseñada siguiendo el estándar internacional IEC-61131 para Controladores Programables y las directrices de PLCopen (asociación mundial para el control de programación industrial).

En esta tesis, se describe la programación del *software* del robot, el protocolo para enviar comandos desde dispositivos externos al robot, el algoritmo de los Bloques de Función, un ejemplo de programa de secuencia *Pick and Place*, y una interfaz gráfica para pantalla táctil para usar la librería y controlar el robot definiendo el parámetros a través de listas desplegables y campos de entrada de datos, facilitando así al usuario el uso de los Bloques de Función. Una vez realizadas las pruebas pertinentes con *hardware* real y finalizadas las pruebas de depuración e implementación de mejoras de funcionalidad, se ha obtenido un paquete de Bloques de Función cuyos parámetros de entrada tienen un rango predefinido que se verifica internamente antes de enviar el comando para asegurar que éste es correcto y será ejecutado por el robot.

Acknowledgements

To Irene, who has been by my side through good and bad times throughout my entire bachelor's degree.

Glossary

Hereunder it is described in alphabetical order the list of acronym names used along this thesis. Even though all these contractions are described the first time they appear in this document, this section can be considered as brief dictionary.

Cobot	Collaborative Robot
FB	Function Block
FIFO	First In, First Out
GUI	Graphical User Interface
HMI	Human Machine Interface
MAC	Machine automation Controllers
OEM	Original Equipment Manufacturer
PLC	Programmable Logic controller
PnP	Pick and Place
PTP	Point to Point
SI	System Integrators
SoC	Separation-Of-Concerns
TCP	Tool Center Point (robot tip)
TCP/IP	Transmission Control Protocol / Internet Protocol
VAC	Voltage Altern Current
VDC	Voltage Direct Current
GND	Ground

Index

1. Prologue

1.1. Origen of the thesis

Most of cobot (Collaborative Robot) manufacturers design their product simple as much as possible with the target of becoming accessible for non-expert users. Since the performance and complexity of collaborative robots are lower than industrial robots, high skilled people are not required to make them work. Cobot is conceived as stand-alone unit providing flexibility in those applications with repetitive tasks in high-mix and very low-volume demanding regular production change-over. Likely this is the tendency, but not all applications require regular production change-over.

There are applications in which most of the production is done in an industrial machine controlled by only one PLC (Programmable Logic Controller) equipped with fences and other safety devices. In those machines, some areas require fast production but human interaction in other zones, therefore cobots are used to allow the operator to entry to the robotic cell without stopping the production. Both, cobot and human, are working in the same workspace with no risks for the human because cobot supports power and force limitation for collaborative operations. But the interesting point is that all components in the machine are controlled by the same PLC, thus instead of just exchanging variables or signals between PLC and Cobot, the proper way to control the robot is sending movement commands, something not trivial with Omron Collaborative Robots because nowadays it is not provided an integrated solution and the user has to create their own programming to send the commands properly from and external PLC.

1.2. Motivation

Currently, there is a high demand in the industry to develop applications, like palletizing or machine tending, with Omron collaborative robots requiring centralized PLC control. Therefore, the aim of this project is providing to customers, like OEM (Original Equipment Manufacturer) or SI (System Integrators), a library with a set of Function Blocs to command the robot without needs to learn how to build complex communications frames. Some of the benefits of this library are:

- Easy to program.
- Fast implementation.
- Reliability.
- Standard solution.

Another emerging kind of application is a collaborative robot mounted on top of a mobile robot, also known as Mobile Manipulator, this concept arises from the need to move the robot to different work areas instead of having a robotic cell and transporting the workpieces to the robotic cell. This avoids having many transportation lines, like belts or conveyors, for moving the goods inside the factory.

In such robotics system design, there is a PLC working as master of both robots, the slaves. With the development of this library, the cobot can be easily controlled. Obviously, another different library would be required to control the mobile robot.

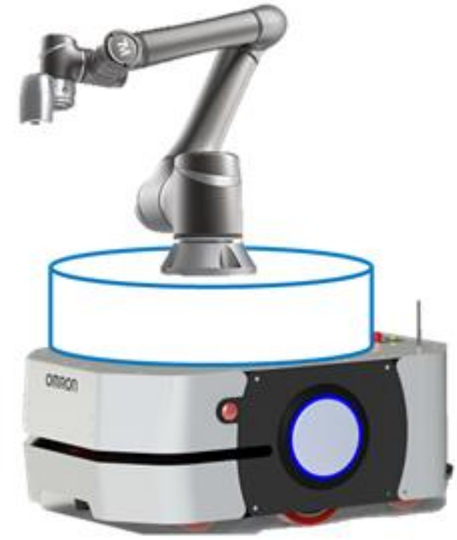


Figure 1.2.1 - Omron Mobile Manipulator (source: own).

2. Introduction

2.1. Thesis objectives

Following the guidelines proposed by PLCopen [1], it is intended to create a library containing a set of FBs (Function Block) encapsulating, in the correct syntax protocol, the string frame package that Omron cobot requires to by commanded from an external device by receiving TCP/IP messages.

The main objective is providing a set tools to easily command the robot from the PLC. Therefore, the user has no needs to be concerned about understanding the protocol required by the robot nor investing time developing the programming and debugging.

2.2. Thesis scope

Even though the objective is controlling a robot from an external device, the project scope is PLC based, providing to the user the following products:

- FB Library to control Omron cobots from external Omron PLC containing following functions:
    - Move the robot to an absolute position, in cartesian or joint coordinates.
    - Move the robot to a relative position, in cartesian or joint coordinates.
    - Move the robot with a circular trajectory in cartesian coordinates.
    - Change robot base coordinates.
    - Change robot tool offset, weight, and inertia.
    - Pause and resume robot program.
- Description for all FB’s, inputs/outputs datatypes and guidelines for their usage.
- Sysmac Studio project with one example of each FB and a PnP (Pick and Place) application sample with absolute position movements.
- HMI (Human Machine Interface) application for an intuitive usage of the robot with all the FB created and a PnP application example.

3. System Setup

3.1. Hardware

The setup arranged to design, develop, debug, and test the correct functionality of the FB library is composed by:

- Omron TM-series [2] collaborative robot is a 6-axis model with power and limiting function, featuring simple programming and integrated camera close to the TCP (Tool Center Point). Cobot is provided with its own controller in which power supply, digital and analogue inputs and outputs, safety functions and amplifiers are included. This robot is the target device which interprets the messages and executes the instructions when received from Omron NJ-series PLC.

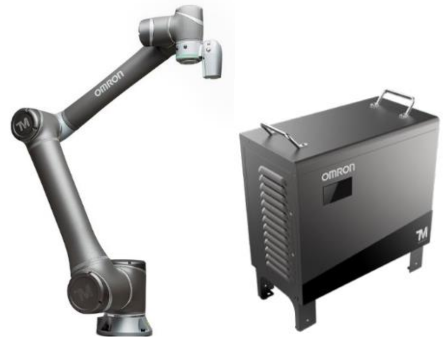


Figure 3.1.1 - Omron TM-Series collaborative robot and controller (source: [10]).

- Omron NJ-series [2] PLC is a machine controller for logic sequence, safety, motion, and database connection, among others, with 500µs of scan cycle time. The FB library has been developed to be used for this controller.

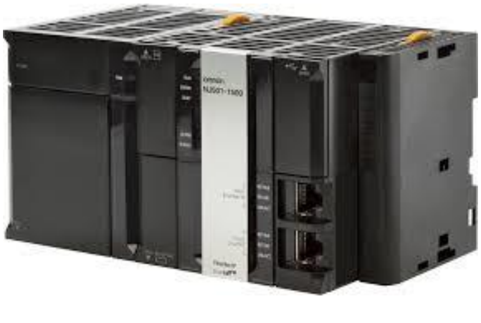


Figure 3.1.2 - Omron NJ-Series PLC (source: [8]).

- Omron NA-series [2] HMI is the innovative Omron touchscreen enabling faster control and monitoring. It provides to the user easy and intuitive 9-inches interface to command the robot using the FB’s library.

Figure 3.1.3 - Omron NA-Series HMI (source: [13]).

3.2. Software

3.2.1. TMflow

TMflow is the graphical environment that provides to users a complete interface for Omron TM-series collaborative robot motion and logic programming environments. TMflow uses a graphical flow chart language to process logic and robot motion.

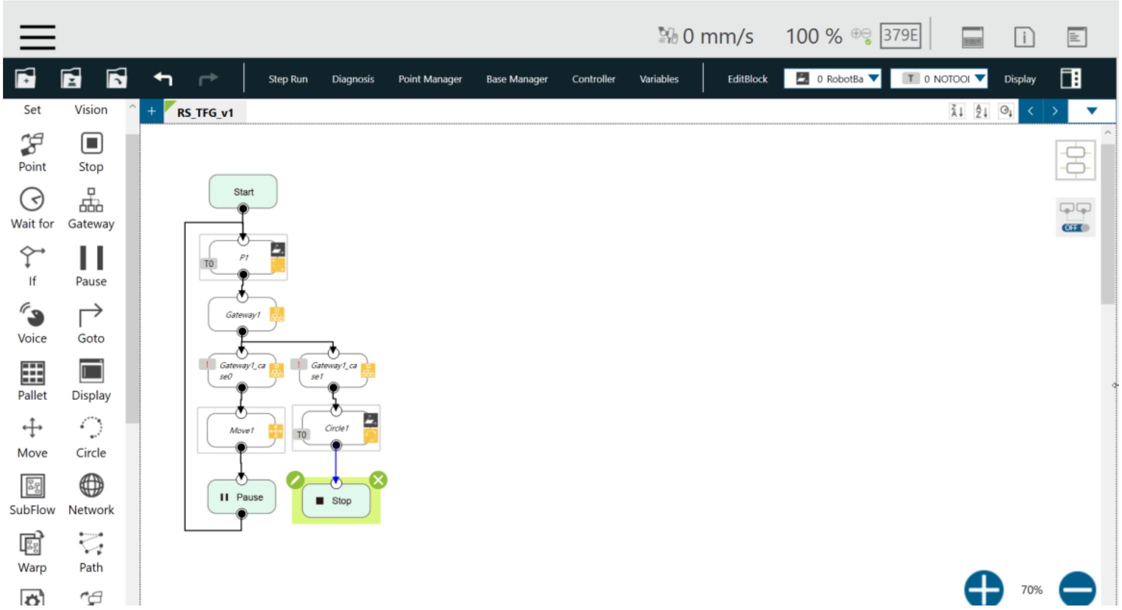


Figure 3.2.1 - TMflow software for robot programming (source: own).

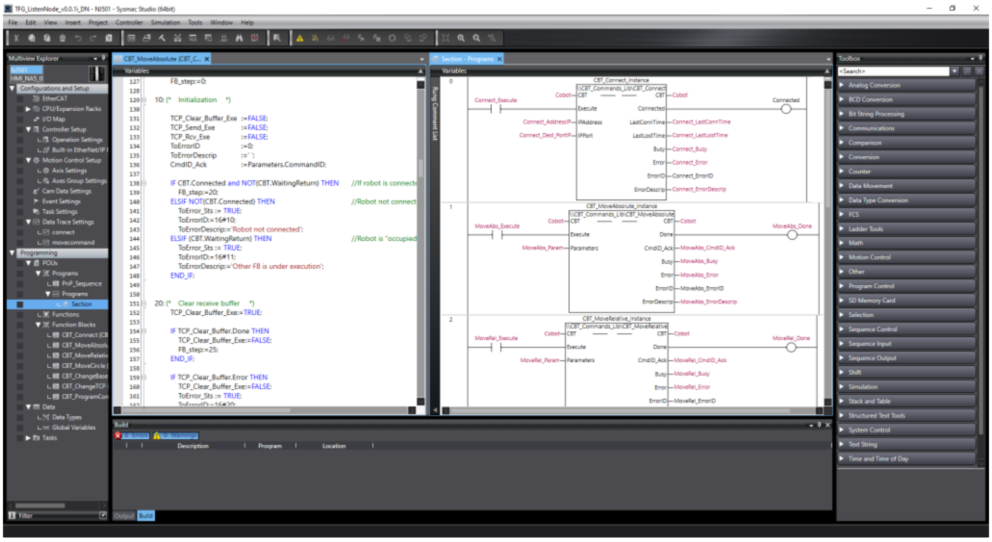
On the right of the project editor (figure 3.2.1) there is a toolbox with all the functions available. Those functions are encapsulated in boxes called *nodes*, by drag and drop, the operator can build the program. The programming is based on flowchart composed by nodes providing certain functionality like: Motion nodes, logic nodes, conditional nodes, and communication nodes. The flow is determined by the arrows connecting the nodes that are executed sequentially. One of the most relevant nodes allowing communications from external device is the one called *Listen Node*.

Safety parameters for power and limiting function can be adjusted: maximum force, speed, position of each joint, etc. Communication settings are also configurable.

3.2.2. Sysmac Studio

The Sysmac Studio provides an integrated development environment to set up, program, debug, and maintain Omron PLC units for motion, logic, safety, drives, vision, robots, and HMI. Sysmac Studio is fully compliant with open standard IEC 61131-3 [4] and supports Ladder, Structured Text, and Function Block programming.

Sysmac Studio allows to program the HMI device with VB.NET (Visual Basic .NET). It is an object-oriented programming language implemented in .NET Framework and developed by Microsoft.


TFG_ListenNode_v0.0.1_DN - NJ501 - Sysmac Studio (64bit)
File Edit View Insert Project Controller Simulation Tools Window Help
Multiview Explorer
CBT_MoveAbsolute (CBT_C...
Section - Programs
Toolbox
Build
Output
Build

```
        FB_step:=0;

    10: (*   Initialization    *)

        TCP_Clear_Buffer_Exe  :=FALSE;
        TCP_Send_Exe          :=FALSE;
        TCP_Rcv_Exe           :=FALSE;
        ToErrorID             :=0;
        ToErrorDescrip        :=' ';
        CmdID_Ack             :=Parameters.CommandID;

        IF CBT.Connected and NOT(CBT.WaitingReturn) THEN      //If robot is connecte
          FB_step:=20;
        ELSIF NOT(CBT.Connected) THEN                         //Robot not connect
          ToError_Sts := TRUE;
          ToErrorID:=16#10;
          ToErrorDescrip:='Robot not connected';
        ELSIF (CBT.WaitingReturn) THEN                        //Robot is "occupied
          ToError_Sts := TRUE;
          ToErrorID:=16#11;
          ToErrorDescrip:='Other FB is under execution';
        END_IF;


    20: (*   Clear receive buffer    *)
        TCP_Clear_Buffer_Exe:=TRUE;

        IF TCP_Clear_Buffer.Done THEN
          TCP_Clear_Buffer_Exe:=FALSE;
          FB_step:=25;
        END_IF;

        IF TCP_Clear_Buffer.Error THEN
          TCP_Clear_Buffer_Exe:=FALSE;
          ToError_Sts := TRUE;
```

Figure 3.2.2 – Sysmac Studio software for PLC programming (source: own).

3.3. Communications Architecture

The connection between robot, PLC, HMI, and personal computer is through EtherNet/IP fieldbus that encapsulates TCP/IP protocol frames as user data.

Each device is equipped with, at least, one ethernet port which is connected to the Omron Industrial Ethernet Switching Hub to provide communication between any device in the network. To be able to configure the Ethernet port settings, it is needed to connect the laptop with each device individually. HMI and PLC ports have been configured with Sysmac Studio software; Cobot port has been configured with TMflow software. In any case, the software was previously installed in the laptop.

Ethernet network cables are Category 6 (Cat 6), standardized twisted pair cable achieving 250MHz.

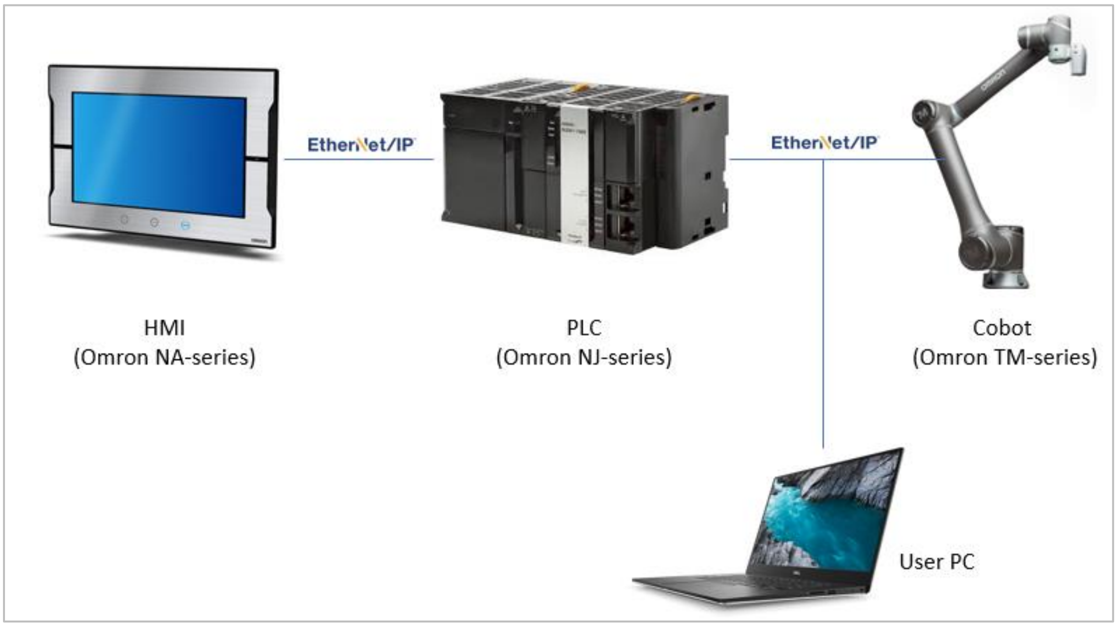


Figure 3.3.1 – Communications architectures based on Ethernet/IP (source: own).

3.4. Electrical connections

All devices are equipped with standard power supply connections. In case of the PLC, cobot, laptop, and 24Vdc power supply, they are powered with 230Vac. They can be plugged into any electrical socket available in the facility where the system will run.

Concerning HMI and Ethernet Switching Hub, 24Vdc are required. In both devices, there is a screws terminal block as power supply connector.

In the image below, the electrical drawing is a sketch designed to illustrate how the system is powered.

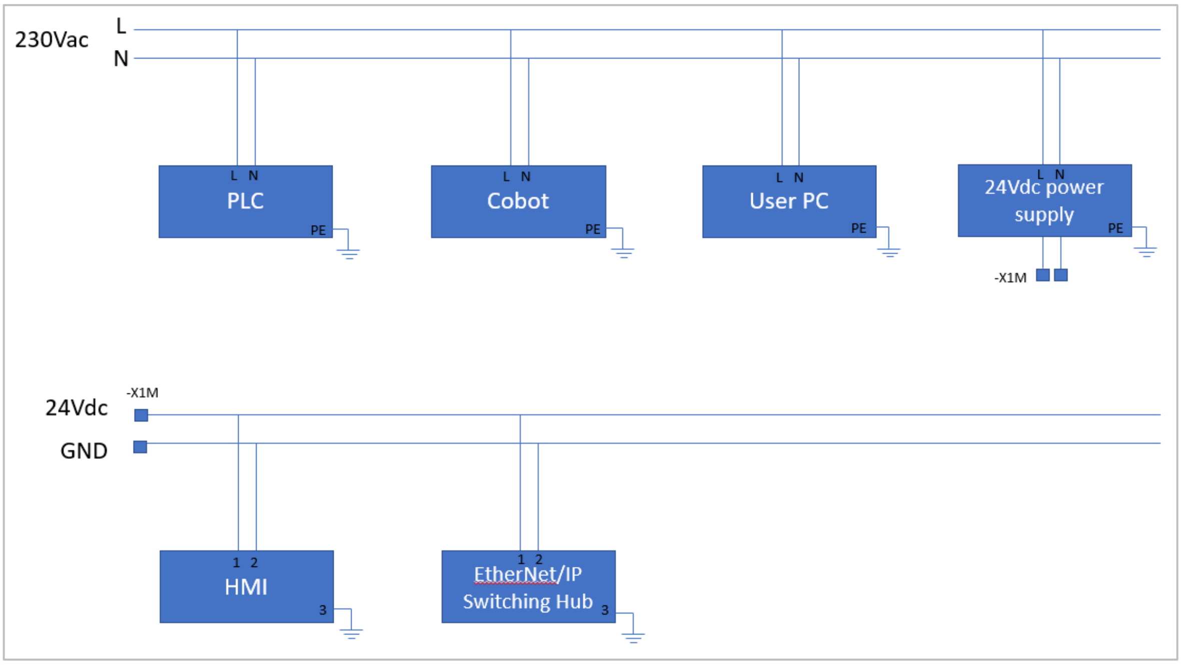


Figure 3.4.1 – Electrical connections (source: own).

4. Robot instructions

4.1. Description

Omron TM-series collaborative robot allows to establish a TCP/IP socket server connection (TCP listener) through the Listen Node function, which provides a specific protocol that allows the execution of the functions available in the Expression Editor [11] from TMflow software.

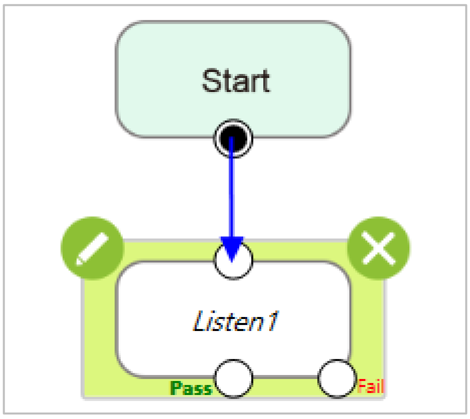
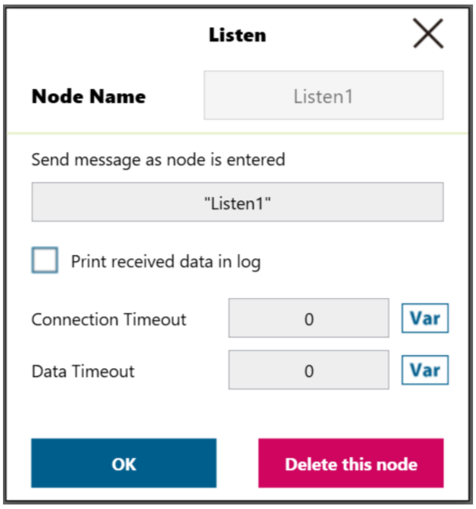


Figure 4.1.1 **-** In Listen Node a TCP/IP server (Socket Server) can be established and be connected by an external device (source: own).

When clicking on the pencil icon in the top-left corner of the node, a popup appears (Figure 4.1.2). In which several settings can be set:

- Node Name: identifies the node in the flowchart.
- Send message as node is entered: when the flow processed in the TMflow reaches this node, this message is sent by server (cobot) to the client (PLC).
- Print received data in log: enables the communication log on TMflow helping on debugging.
- Connection timeout: when entering in this node, if more than the time set (in milliseconds) is not connected, it will timeout. Use 0 to disable this timeout.
- Data Timeout: when entering in this node and connected, is no data is received during the time set (in milliseconds) it will timeout. Use 0 to disable this timeout.



Node Name: Listen1

Send message as node is entered: "Listen1"

☐ Print received data in log

Connection Timeout: 0

Data Timeout: 0

Figure 4.1.2 - Listen Node settings (source: own).

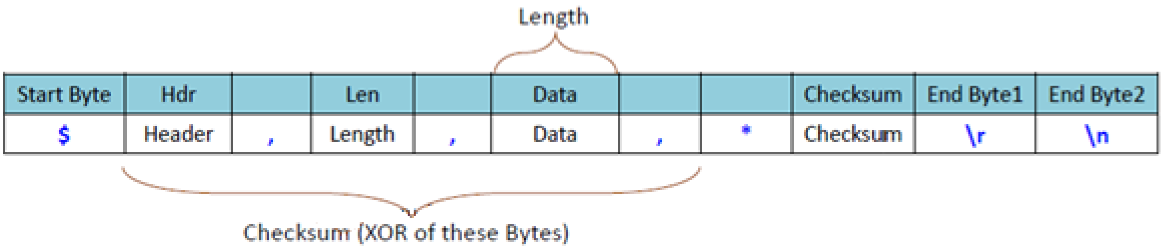
When the program flow execution enters in Listen Node, the flow will keep at this node until either of the two exit conditions is fulfilled:

- Pass: *ScriptExit*() command is executed and the project is stopped.
- Fail: 1. Connection Timeout.
  2. Data Timeout.

When a command is received by Listen node, it will be executed in FIFO (*First In, First Out*) order, meaning that the command is placed in the queue and executed in arrival order. In case a command is not valid, an error message is sent back to the client connection. Otherwise, if the command is valid, it will be placed on the queue.

4.2. Communications Protocol

Omron collaborative robot has its own communications protocol. Hence the commands sent from external devices must comply a very specific data structure. It is composed by data such header to identify the purpose of the message, length or quantity of bytes in the message, complete instruction including parameters, checksum to verify the message has been received correctly, and end bytes to identify the end of the message.



Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
$	Header	,	Length	,	Data	,	*	Checksum	\r	\n

***Figure 4.2.1* –** Data frame structure (source: [11]).

Name	Size	ASCII	HEX	Description
Start Byte	1	$	0x24	Start Byte for Communication
Header	*X*			Header for Communication
Separator	1	,	0x2C	Separator between Header and Length
Length	*Y*			Length of Data
Separator	1	,	0x2C	Separator between Length and Data
Data	*Z*			Communication Data
Separator	1	,	0x2C	Separator between Data and Checksum
Sign	1	*	0x2A	Begin Sign of Checksum
Checksum	*2*			Checksum of Communication
End Byte 1	1	\r	0x0D	
End Byte 2	1	\n	0x0A	End Byte of Communication

***Figure 4.2.2* –** Data frame contents description (source: [11]).

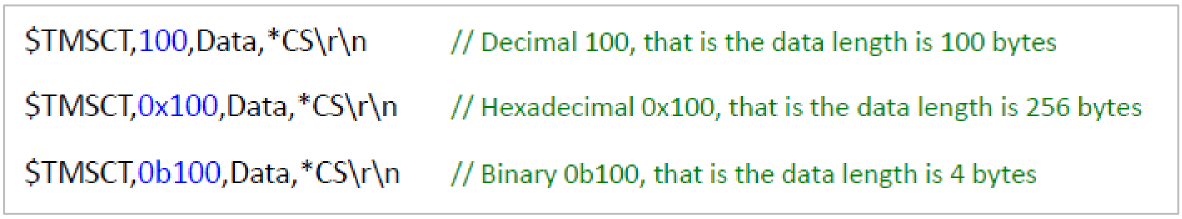
Header

Defines the purpose of the communication package. The data contained in the command can vary depending on the Header:

- TMSCT → Function command sent from PLC.
- TMSTA → Acquiring status information or properties data:
  - Send from PLC, asking if robot is in Listen node.
  - Send from robot, returning information (answer).
- CPERR → Communication data error (e.g., package error, checksum error, header error, etc.).

Length

Defines the length in byte type. The numeric format can be decimal, hexadecimal, or binary. Example:



```
$TMSCT,100,Data,*CS\r\n        // Decimal 100, that is the data length is 100 bytes
$TMSCT,0x100,Data,*CS\r\n      // Hexadecimal 0x100, that is the data length is 256 bytes
$TMSCT,0b100,Data,*CS\r\n      // Binary 0b100, that is the data length is 4 bytes
```

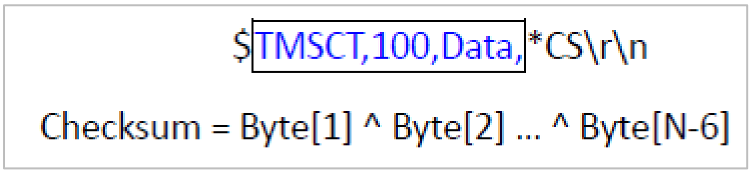
Figure 4.2.3 – Length numeric format examples (source: [11]).

Data

Content of the communication data. This can vary depending on the purpose of the function command and its parameters (refer to section *4.3. Function commands*).

Checksum

The checksum of the communication package is calculated with XOR (eXclusive Or / eXclusive disjunction) logical operation. The range for the checksum calculation starts from $ and finish at * (being $ and * symbols excluded). The representation of the checksum is fixed to 2 bytes in hexadecimal format (without 0x).



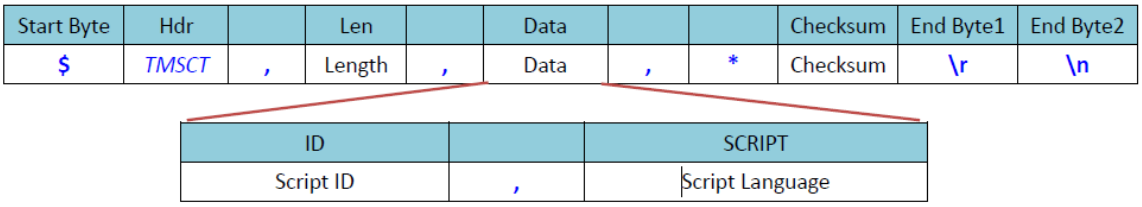
```
$TMSCT,100,Data,*CS\r\n
Checksum = Byte[1] ^ Byte[2] … ^ Byte[N-6]
```

Figure 4.2.4 – Checksum calculation (source: [11]).

Following subsection explains the three different headers accepted.

4.2.1. TMSCT

Defines the communication package as External Script Language. Use to send a command including parameters for its execution. Data contains two parts separated by comma: ID and Script.



Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
$	*TMSCT*	,	Length	,	Data	,	*	Checksum	\r	\n

ID		SCRIPT
Script ID	,	Script Language

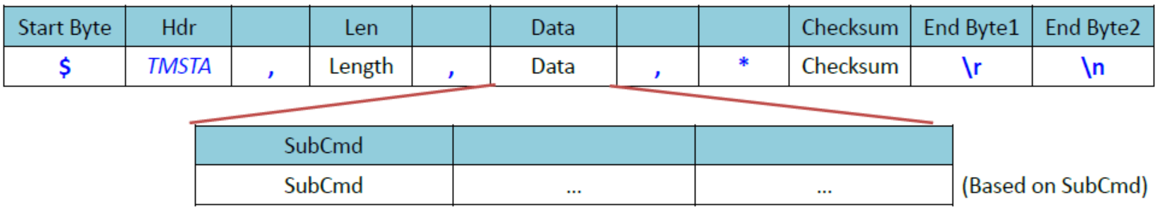
Figure 4.2.5 – Checksum calculation (source: [11]).

Data:

- ID Used as specifying the target SCRIPT of return message.
- , Separator.
- SCRIPT Multiline script containing the Script Language.

4.2.2. TMSTA

Defines the communication package to acquire status or properties. When sent from an external device, it is used to detect if the collaborative robot is in listener mode, consequently it is ready to accept and execute external commands. Data contains different subcommand (SubCMD).



Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
$	*TMSTA*	,	Length	,	Data	,	*	Checksum	\r	\n

SubCmd			
SubCmd	…	…	(Based on SubCmd)

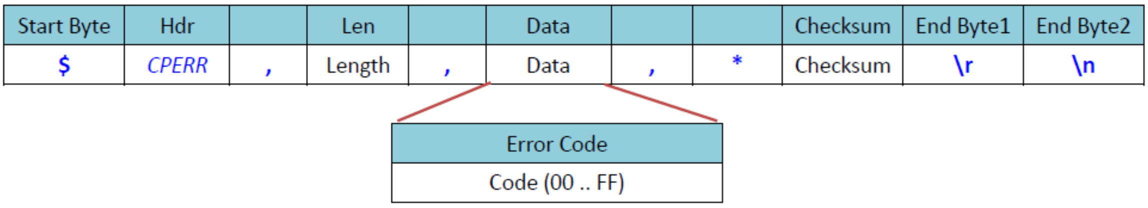
Figure 4.2.6 – Checksum calculation (source: [11]).

SubCmd:

- 00 Asks if robot is in external script control or not (Listen node).
- 01 Asks when robot motion has been completed.
- 90..99 Sends data message as variable value.

4.2.3. CPERR

Defines the communication package as Communication Error Protocol. It is used by robot to response to PLC in case an error has been detected in the received message.



Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
$	*CPERR*	,	Length	,	Data	,	*	Checksum	\r	\n

Error Code
Code (00 .. FF)

Figure 4.2.7 – Checksum calculation (source: [11]).

Error Code:

-	00	No error.
-	01	Packet error.
-	02	Checksum error.
-	03	Header error.
-	04	Packet data error.
-	F1	Not in Listen Node.

4.3. Function commands

Function commands can be only performed with external scripts when the project flow is in Listen Node and it receives a message with $TMSCT as header of the command. All motion functions are queued in the buffer and executed in arrival order.

The Function Commands implemented in this library are:

- PTP()
- Line()
- Circle()
- Move_PTP()
- Move_ Line()
- ChangeBase()
- ChangeTCP()
- QueueTag()
- Pause()
- Resume()
- ScriptExit()

These functions commands [11] are described below.

4.3.1. PTP()

Define and send PTP (Point to Point) absolute motion command into buffer for execution. See description below:

Syntax

```
bool PTP(
        string,
        float, float, float, float, float, float,
        int,
        int,
        int,
        bool,
        int, int, int
)
```

Parameters

string Definition of data format, combines three letters:

#1: Motion target format:

"C" expressed in Cartesian coordinates.

#2: Speed format:

"P" expressed as a percentage (%).

#3: Blending format:

"P" expressed as a percentage (%).

float, float, float, float, float, float

Motion target location: X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

int Speed setting expressed as a percentage (%).

int Time interval to accelerate to top speed (ms).

int Blending value expressed as a percentage (%).

bool Disable precise positioning.

true Disable precise positioning.

false Enable precise positioning.

int, int, int

Pose of robot: Config1, Config2, Config3.

- Config1:
  - RIGHTY = 0
  - LEFTY = 1

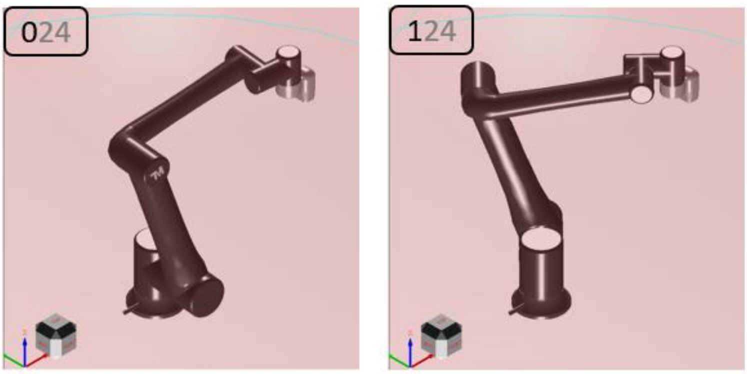


Figure 4.3.1 – Config1 pose of robot: RIGHTY = 0, LEFTY = 1 (source: own).

- Config2:
  - ABOVE = 2
  - BELOW = 3

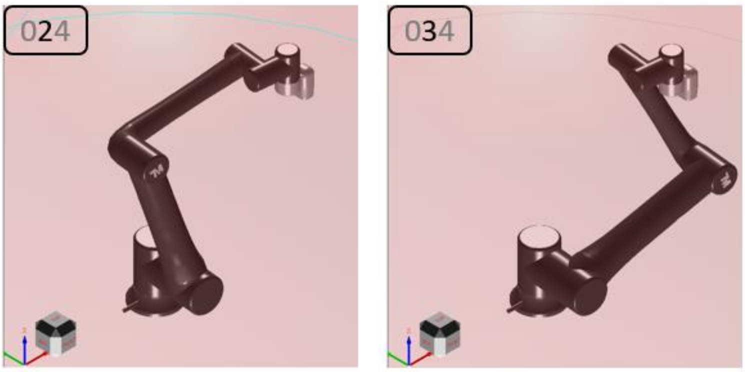


***Figure 4.3.2* –** Config2 pose of robot: ABOVE = 2, BELOW = 3 (source: own).

- Config3:
  - NOFLIP = 4
  - FLIP = 5

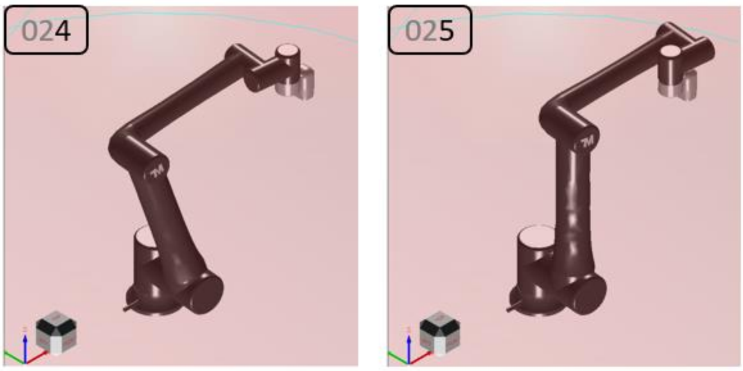


***Figure 4.3.3* –** Config3 pose of robot: NOFLIP = 4, FLIP = 5 (source: own).

Return

bool *True* Command accepted; *False* Command rejected (format error).

Note

Data format parameter includes: (1) “CPP”.

8 different robot pose configurations are possible:

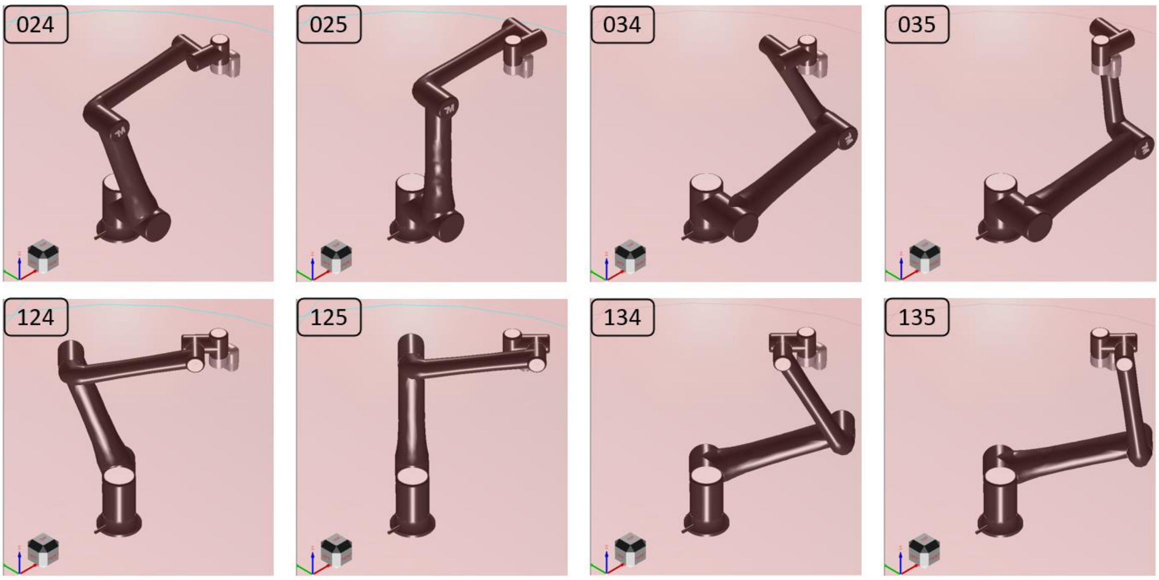


Figure 4.3.4 – Pose of robot can be in 8 different configurations: 024: RIGHTY, ABOVE, NOFLIP; 025: RIGHTY, ABOVE, FLIP; 034: RIGHTY, BELOW, NOFLIP; 035: RIGHTY, BELOW, FLIP; 124: LEFTY, ABOVE, NOFLIP; 125: LEFTY, ABOVE, FLIP; 134: LEFTY, BELOW, NOFLIP; 135: LEFTY, BELOW, FLIP (source: own).

Example

```
PTP("CPP",417.50,-122.30,343.90,180.00,0.00,90.00,10,200,0,false,0,2,4)
//Move to coordinate (417.50, -122.30, 343.90, 180.00, 0.00, 90.00), with PTP, speed=10%,
time to top speed=200ms, no blending, precise positioning disabled, pose = 024)
```

4.3.2. Line()

Define and send Line absolute motion command into buffer for execution. See description below:

Syntax

```
bool Line(
        string,
        float, float, float, float, float, float,
        int,
        int,
        int,
        bool
)
```

Parameters

string Definition of data format, combines three letters:

#1: Motion target format:

"C"	expressed in Cartesian coordinates.

#2: Speed format:

"P"	expressed as a percentage (%).
"A"	expressed in velocity (mm/s).

#3: Blending format:

"P"	expressed as a percentage (%).
"R"	expressed in radius (mm).

float, float, float, float, float, float

Motion target location: X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

int Speed setting expressed as a percentage (%) or in velocity (mm/s).

int Time interval to accelerate to top speed (ms).

int Blending value expressed as a percentage (%) or in radius (mm).

bool Disable precise positioning.

true Disable precise positioning.

false Enable precise positioning.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Note

Data format parameter includes: (1) "CPP", (2) "CPR", (3) "CAP" and (4) "CAR".

Example

```
Line("CAR",417.50,-122.30,343.90,180.00,0.00,90.00,100,200,50,false)
//Move to coordinate (417.50, -122.30, 343.90, 180.00, 0.00, 90.00), with Line, speed=100mm/s, time to top speed=200ms, blending radius = 50mm and precise positioning disabled.
```

4.3.3. Circle()

Define and send Circle absolute motion command into buffer for execution. See description below:

Syntax

```
bool Circle(
        string,
        float, float, float, float, float, float,
        float, float, float, float, float, float,
        int,
        int,
        int,
        int,
        bool
)
```

Parameters

string Definition of data format, combines three letters:

#1: Motion target format:

"C" expressed in Cartesian coordinates.

#2: Speed format:

"P" expressed as a percentage (%).

"A" expressed in velocity (mm/s).

#3: Blending format:

"P" expressed as a percentage (%).

float, float, float, float, float, float

A point on arc: X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

float, float, float, float, float, float

The end point of arc: X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

int Speed setting expressed as a percentage (%) or in velocity (mm/s).

int Time interval to accelerate to top speed (ms).

int Blending value expressed as a percentage (%).

Int Arc angle (◦). If non-zero value is given, the TCP will keep the same pose and move from current point to the assigned arc angle via the given point and end point on arc. If zero is given, the TCP will move from current point and pose to end point and pose via the point on arc with linear interpolation on pose.

bool Disable precise positioning.

true Disable precise positioning.

false Enable precise positioning.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Note

Data format parameter includes: (1) "CPP", (2) "CAP".

Example

```
Circle("CAP",417.50,-122.30,343.90,180.00,0.00,90.00,
381.70,208.74,343.90,180.00,0.00,135.00,100,200,50,270,false)
//Via point = (417.50, -122.30, 343.90, 180.00, 0.00, 90.00), end point = (381.70, 208.74, 343.90, 180.00, 0.00, 135.00), move on 270 degrees arc, speed=100mm/s, time to top speed=200ms, blending value radius = 50%, precise positioning disabled.
```

4.3.4. Move_PTP()

Define and send PTP relative motion command into buffer for execution. See description below:

Syntax

```
bool Move_PTP(
        string,
        float, float, float, float, float, float,
        int,
        int,
        int,
        bool
)
```

Parameters

string Definition of data format, combines three letters:

#1: Relative motion target format:

"C" expressed in Current base coordinates.

"T" expressed in Tool coordinates.

"J" expressed in Joint angle.

#2: Speed format:

"P" expressed as a percentage (%).

#3: Blending format:

"P" expressed as a percentage (%).

float, float, float, float, float, float

Relative motion target location. If expressed in current base coordinates or tool coordinates, it includes the tool end TCP relative motion value with respect to the specified coordinate: X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦). If defined with joint angle, it includes the angles of six joints: Joint1 (◦), Joint2 (◦), Joint3 (◦), Joint4 (◦), Joint5 (◦), Joint6 (◦).

int	Speed setting expressed as a percentage (%).
int	Time interval to accelerate to top speed (ms).
int	Blending value expressed as a percentage (%).

bool Disable precise positioning.

true	Disable precise positioning.
false	Enable precise positioning.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Note

Data format parameter includes: (1) "CPP", (2) "TPP", (3) "JPP".

Example

```
Move_PTP("TPP",0,0,10,45,0,0,10,20,0,false)
//Move (0, 0, 10, 45, 0, 0) with respect to tool coordinate, with PTP motion, speed =10%, time to top speed = 200ms, no blending, precise positioning disabled.
```

4.3.5. Move_Line()

Define and send Line relative motion command into buffer for execution. See description below:

Syntax

```
bool Move_Line(
        string,
        float, float, float, float, float, float,
        int,
        int,
        int,
        bool
)
```

Parameters

string Definition of data format, combines three letters:

#1: Relative motion target format:

"C" expressed in Current base coordinates.

"T" expressed in Tool coordinates.

#2: Speed format:

"P" expressed as a percentage (%).

"A" expressed in velocity (mm/s).

#3: Blending format:

"P" expressed as a percentage (%).

"R" expressed in radius (mm).

float, float, float, float, float, float

Relative motion target. It includes the tool end TCP relative motion value with respect to the specified current base or tool coordinate: X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

int Speed setting expressed as a percentage (%) or in velocity (mm/s).

int Time interval to accelerate to top speed (ms).

int Blending value expressed as a percentage (%) or in radius (mm).

bool Disable precise positioning.

true Disable precise positioning.

false Enable precise positioning.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Note

Data format parameter includes: (1) "CPP", (2) "CPR", (3) "CAP", (4) "CAR", (5) "TPP", (6) "TPR", (7) "TAP" and (8) "TAR".

Example

```
Move_Line("TPP",0,0,10,45,0,0,10,20,0,false)
//Move (0, 0, 10, 45, 0, 0) with respect to tool coordinate with Line motion, speed =10%,
time to top speed = 200ms, no blending, precise positioning disabled.
```

4.3.6. ChangeBase()

Define and send the command of changing the base of the follow-up motions into buffer for execution. See description below:

Syntax

```
bool ChangeBase(
        float, float, float, float, float, float
)
```

Parameters

float, float, float, float, float, float

Base parameters combining X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

Return

bool *True* Command accepted; *False* Command rejected (format error).

Example

```
ChangeBase(20,30,10,0,0,90)
//Change the base value to (20, 30, 10, 0, 0, 90).
```

4.3.7. ChangeTCP()

Define and send the command of changing the TCP of the follow-up motions into buffer for execution. See description below:

Syntax

```
bool ChangeBase(
        float, float, float, float, float, float,
        float,
        float, float, float, float, float, float, float, float, float
)
```

Parameters

float, float, float, float, float, float

TCP parameters combining X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

float

Tool's weight.

float, float, float, float, float, float, float, float, float

Tool's moment of inertia: (1)Ixx (kg·mm²), (2)Iyy (kg·mm²), (3)Izz (kg·mm²) and its frame of reference (4)X (mm), (5)Y (mm), (6)Z (mm), (7)RX (◦), (8)RY (◦), (9)RZ (◦).

Return

bool *True* Command accepted; *False* Command rejected (format error).

Example

```
ChangeTCP(0,0,150,0,0,90,2,2,0.5,0.5,0,0,-80,0,0,0)
//Change the TCP value to (0, 0, 150, 0, 0, 90), weight = 2kg, moment of inertia = (2, 0.5 ,0.5) , and frame of reference = (0, 0, -80, 0, 0, 0).
```

4.3.8. QueueTag()

Identify the robot motion with a Number to return feedback when robot motion finishes the current robot motion in process. See description below:

Syntax

```
bool QueueTag(
        int,
        int
)
```

Parameters

int

The tag number. Valid for integers between 1 and 15.

int

Wait for the tagging to continue processing or not:

0	Not wait.
1	Wait.

When the value is set to 0, no wait for tagging to continue processing.
When the value is set to 1, the process stays in the function and waits for the tagging to complete and continue processing.

Return

bool *True* When tagged successfully; *False* when tagged unsuccessfully.

Example

```
QueueTag(8,0)
//Tags the robot motion with number 8 for returning feedback when robot motion finishes
```

4.3.9. Pause()

Pause the project and the motion of the robot. See description below:

Syntax

```
bool Pause(
)
```

Parameters

void

No input values required.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Example

```
Pause()
```

4.3.10. Resume()

Resume the project and the motion of the robot. See description below:

Syntax

bool Resume(
)

Parameters

void
No input values required.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Example

Resume()

4.3.11. ScriptExit()

Exit external script control mode once the last motion command has been executed and finished. Listen Node is quitted, and not further external script functions commands are accepted. See description below:

Syntax

bool ScriptExit(
)

Parameters

void
No input values required.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Note

Exit the external script control mode and wait for the command to finish, and then quit the listen node and move on with the pass route.

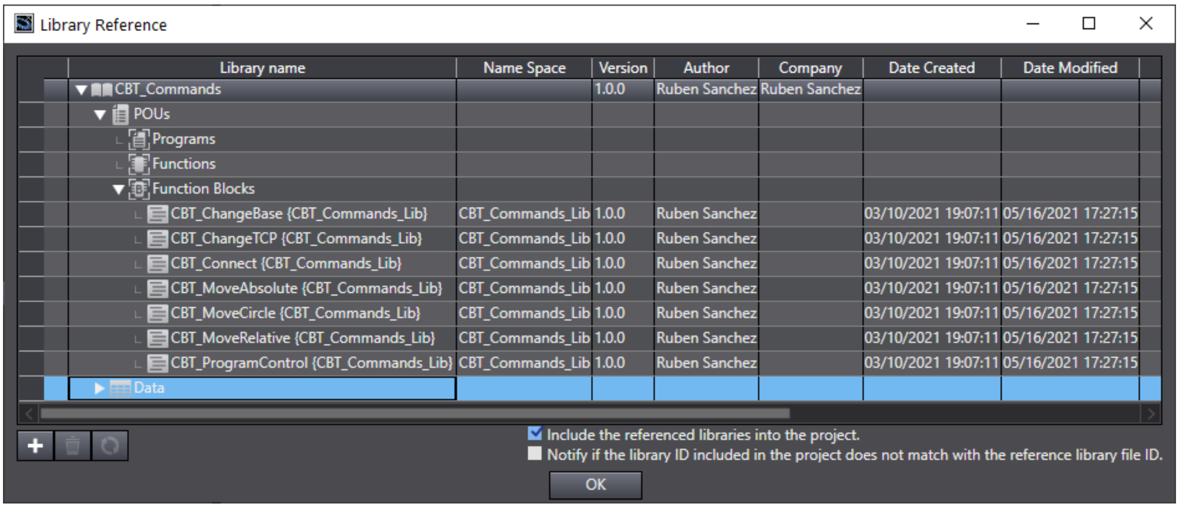
Example

ScriptExit()

5. Library

This library encapsulates many of the Expression Editor [11] functions from TMflow. Among them, the most relevant ones in terms of robot motion, coordinate parameters and program execution control. The seven Function Blocks developed for this library are:

- CBT_ChangeBase
- CBT_ChangeTCP
- CBT_Connect
- CBT_MoveAbsolute
- CBT_MoveRelative
- CBT_MoveCircle
- CBT_ProgramControl



Library name	Name Space	Version	Author	Company	Date Created	Date Modified
CBT_Commands		1.0.0	Ruben Sanchez	Ruben Sanchez		
POUs						
Programs						
Functions						
Function Blocks						
CBT_ChangeBase {CBT_Commands_Lib}	CBT_Commands_Lib	1.0.0	Ruben Sanchez		03/10/2021 19:07:11	05/16/2021 17:27:15
CBT_ChangeTCP {CBT_Commands_Lib}	CBT_Commands_Lib	1.0.0	Ruben Sanchez		03/10/2021 19:07:11	05/16/2021 17:27:15
CBT_Connect {CBT_Commands_Lib}	CBT_Commands_Lib	1.0.0	Ruben Sanchez		03/10/2021 19:07:11	05/16/2021 17:27:15
CBT_MoveAbsolute {CBT_Commands_Lib}	CBT_Commands_Lib	1.0.0	Ruben Sanchez		03/10/2021 19:07:11	05/16/2021 17:27:15
CBT_MoveCircle {CBT_Commands_Lib}	CBT_Commands_Lib	1.0.0	Ruben Sanchez		03/10/2021 19:07:11	05/16/2021 17:27:15
CBT_MoveRelative {CBT_Commands_Lib}	CBT_Commands_Lib	1.0.0	Ruben Sanchez		03/10/2021 19:07:11	05/16/2021 17:27:15
CBT_ProgramControl {CBT_Commands_Lib}	CBT_Commands_Lib	1.0.0	Ruben Sanchez		03/10/2021 19:07:11	05/16/2021 17:27:15
Data						

☑ Include the referenced libraries into the project.
□ Notify if the library ID included in the project does not match with the reference library file ID.

Figure 5.0.1 – Library reference when imported to the PLC project (source: own).

In some cases, different functions have been included inside the same FB. *PTP()* and *Line()* functions have been encapsulated in *CBT_MoveAbsolute*; or *Move_PTP()* and *Move_Line()* functions have been encapsulated in *CBT_MoveRelative*. In both cases, there is an input parameter in which 'Line' or 'PTP' option can be selected in order to execute one or another.

Once the customer wants to use the library, it must be loaded into the software to be used in the project in which the PLC is programmed. When user enters in programming editor in Sysmac Studio environment, all FBs in the library are displayed in the Toolbox (Figure 5.0.2), thus customer can drag & drop the FBs into the program.

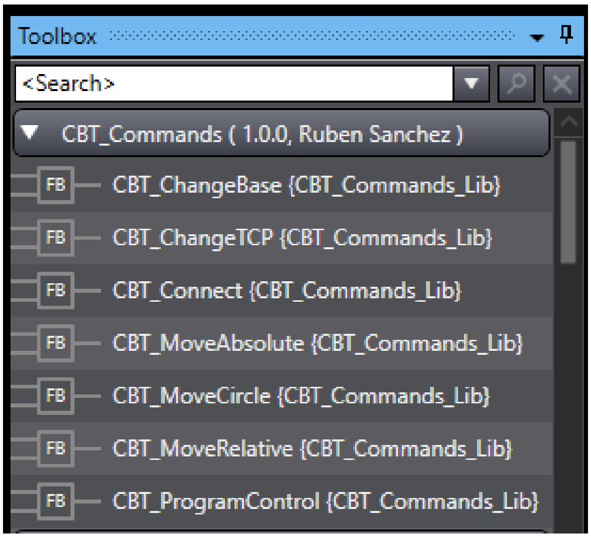

Toolbox
<Search>
CBT_Commands (1.0.0, Ruben Sanchez)
FB
CBT_ChangeBase {CBT_Commands_Lib}
FB
CBT_ChangeTCP {CBT_Commands_Lib}
FB
CBT_Connect {CBT_Commands_Lib}
FB
CBT_MoveAbsolute {CBT_Commands_Lib}
FB
CBT_MoveCircle {CBT_Commands_Lib}
FB
CBT_MoveRelative {CBT_Commands_Lib}
FB
CBT_ProgramControl {CBT_Commands_Lib}

Figure 5.0.2 – Library shown in the toolbox of Sysmac Studio once it has been imported onto the PLC project (source: own).

5.1. PLCopen and IEC 61131-3 compliant Function Blocks

PLCopen [1] is an independent organization, founded in 1992, providing support to the user community in terms of harmonization of control programming and application, and interfacing engineering for industrial automation.

IEC 61131-3 standard [4] for PLC is the unified suite of programming languages for programmable controllers. Programming languages like:

- Ladder diagram (LD).
- Function block diagram (FBD).
- Structured text (ST).
- Instruction list (IL).
- Sequential function chart (SFC).

PLCopen and IEC 61131-3 provide the basis to achieve higher efficiency during application development, maintenance over the life cycle, adding new functionalities, etc. This is the reason why it has been decided to follow their guidelines.

5.1.1. Function Block appearance

The document “Creating PLCopen compliant Function Block libraries” [3] has been a great help to understand how a Function Block must behave and how its appearance should be. The behaviour of the FBs has been described in section 5.1.2).

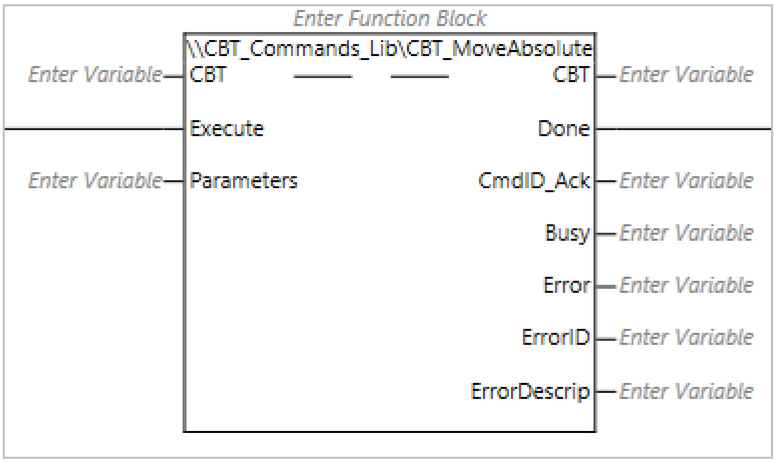
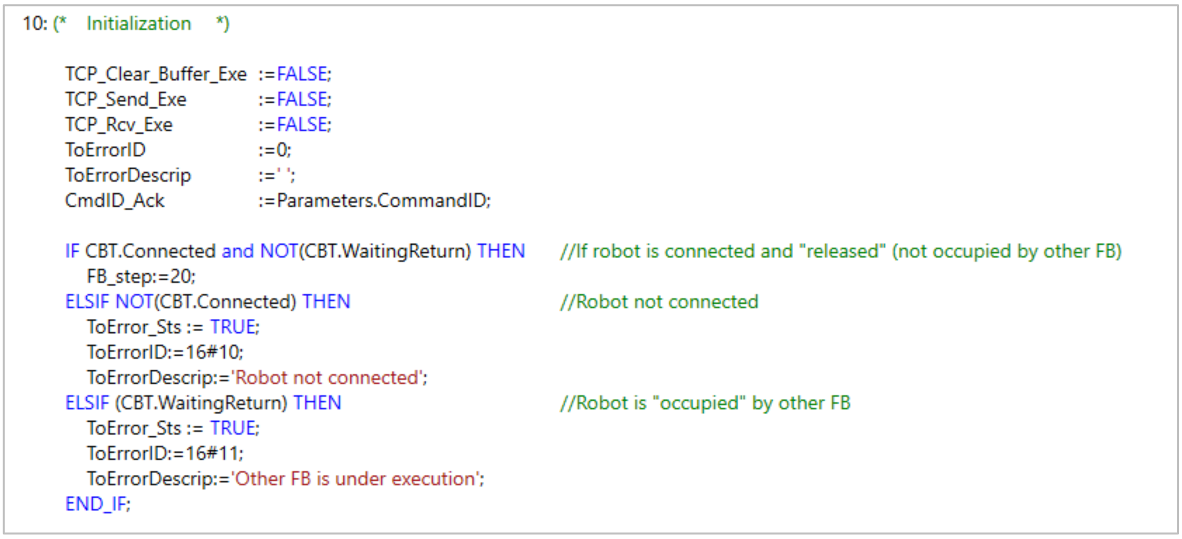


Figure 5.1.1 – Function block appearance similar for all FBs in the library (source: own).

As one of the programming languages stated in IEC 61131-3 standard, the algorithm inside each Function Block has developed in Structured text (ST) programming language.



```
10: (*   Initialization    *)

      TCP_Clear_Buffer_Exe  :=FALSE;
      TCP_Send_Exe          :=FALSE;
      TCP_Rcv_Exe           :=FALSE;
      ToErrorID             :=0;
      ToErrorDescrip        :=' ';
      CmdID_Ack             :=Parameters.CommandID;

      IF CBT.Connected and NOT(CBT.WaitingReturn) THEN     //If robot is connected and "released" (not occupied by other FB)
         FB_step:=20;
      ELSIF NOT(CBT.Connected) THEN                        //Robot not connected
         ToError_Sts := TRUE;
         ToErrorID:=16#10;
         ToErrorDescrip:='Robot not connected';
      ELSIF (CBT.WaitingReturn) THEN                       //Robot is "occupied" by other FB
         ToError_Sts := TRUE;
         ToErrorID:=16#11;
         ToErrorDescrip:='Other FB is under execution';
      END_IF;
```

Figure 5.1.2 – Part of the algorithm of a FB from the library programmed in Structured Text (source: own).

The script in each Function block has been implemented following a methodological principle in computations software engineering known as SoC (Separation-Of-Concerns) [5]. This principle states that a program design must be separated in distinct sections, one per concern.

Separation-Of-Concerns has been implemented in the library at two different levels. In terms of library contents, there is not only 1 FB doing all actions, there are 7 different FBs and each one has its own purpose, but conceptually, they can be classified in 4 groups:

- Communications connection (CBT_Connect).
- Robot movements (CBT_MoveAbsolute, CBT_MoveRelative, CBT_MoveCircle).
- Coordinates functions (ChangeBase, ChangeTCP).
- Program control (CBT_ProgramControl).

The second implementation of Separation-Of-Concerns is in terms of scripting structure inside each FB. There are 2 main and separated sequences that are linked each other but they have different purposes:

- "*State Diagram Control*" focused on monitoring execution state. It updates the FB outputs (Done, Busy, Error, etc.) depending on the FB inputs and the *Algorithm Sequence*.
- "*Algorithm Sequence*" interprets the data provided in the Parameters input, converts Parameters into strings, builds the TCP/IP frame, and sends the command to the robot.

Both sequences are described in the following sections.

5.1.2. State Diagram Control

State Diagram Control is the part of the script focused on output variable operation and timing. It is responsible of how the algorithm behaves depending on the input variables and how the output variables monitor the status of the internal algorithm. The common IO variables are:

- Execute: input variable that gives the execution condition for the FB.
- Done: output variable that shows the completion of the execution for the FB.
- Busy: output variable that shows execution in progress for the FB.
- Error: output variable that indicates error end flag for the FB.
- ErrorID: output variable that indicates error number (hexadecimal) for the FB.
- ErrorDescription: output variable that indicates error description for the FB.

Values of the output variables can be monitored to determine the status throughout instruction execution. Execution starts when *Execute* input changes to TRUE, then *Busy* output changes to TRUE, *Done* output changes to FALSE, and *Error* output changes to FALSE.

In case of "Normal End": *Busy* changes to FALSE and *Done* changes to TRUE. *Error* does not change (remains FALSE).

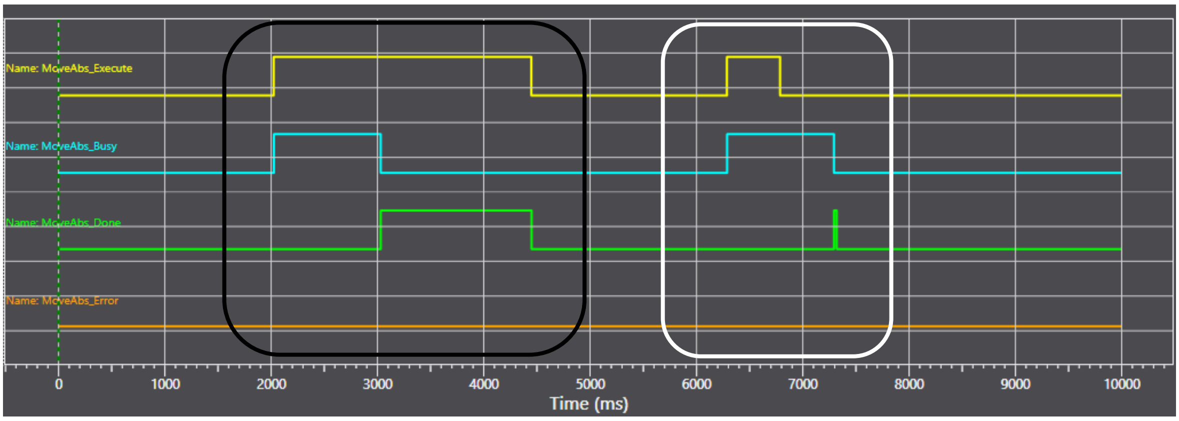


Figure 5.1.3 – Normal end output variable operation and timing chart for *Execute* (yellow), *Busy* (blue), *Done* (green), *Error* (orange). Black square highlights the example if *Execute* is TRUE until *Done* changes to TRUE. *Done* stays TRUE until *Execute* changes to FALSE. White square highlights the example if *Execute* changes back to FALSE before *Done* changes to TRUE. *Done* stays TRUE for only one task period (source: own).

In case of "Error End": *Busy* changes to FALSE and *Error* changes to TRUE. *Done* does not change (remains FALSE). *ErrorID* and *ErrorDescription* display valuable information for error identification.

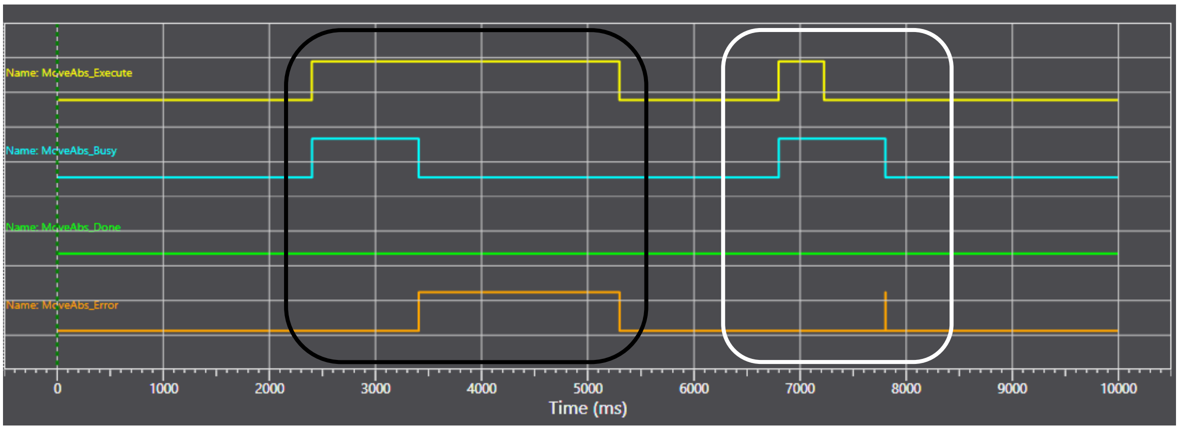


Figure 5.1.4 – Normal end output variable operation and timing chart for *Execute* (yellow), *Busy* (blue), *Done* (green), *Error* (orange). Black square highlights the example if *Execute* is TRUE until *Error* changes to TRUE. *Error* stays TRUE until *Execute* changes to FALSE. White square highlights the example if *Execute* changes back to FALSE before *Error* changes to TRUE. *Error* stays TRUE for only one task period. Error Description output remains showing the message until FB is executed again (source: own).

For those Function Blocks related to robot motion (CBT_MoveAbsolute, CBT_MoveRelative, CBT_MoveCircle) *Done* output behaves different depending on the *EnableBlending* input parameter:

- If *EnableBlending* is TRUE, *Done* changes to TRUE when command has been accepted by the robot even though the movement has not been completed yet. It allows to send next motion command and put it in the robot queue before current movement is completed, thus blending between movements can be applied by the robot.
- If *EnableBlending* is FALSE, *Done* changes to TRUE when the robot movement has been completed. Useful in those cases in which the robot should do actions like open/close the gripper or activate a signal once the robot has achieved a certain position.

5.1.3. Algorithm Sequence

The algorithm structure inside the Function Blocks is similar each other: When *Execute* input changes to TRUE, if the FB is not being executed at that moment, the values of all input parameters are saved in internal variables. The reason is preventing the values are not changed during FB execution. Then, if the robot is connected to the PLC and, after clearing the communications port buffer, it is verified that all values of input parameters are inside the acceptable range. Next step is converting all input parameters into string format to create the complete message. Once the number of characters is found, the checksum is obtained (these concepts are described in detail in section *5.3. Function Blocks description*). Finally, the message is sent to the robot and the FB waits until the return message is received from robot. Thus, the FB execution is finished.

The sequence verifies on each step if the operation is valid in order to proceed with the step after. Otherwise, if operation is not valid or an error exists, the FB will trigger the error output and will “Error end”. Therefore, *ErrorID* and *ErrorDescrip* provides valuable information helping to identify and debug the issue.

In general terms, all Function Blocks follow the same sequence: get data from input parameters, build the frame, send the frame to the robot, wait for response and finish execution (see Figure 5.1.3.1).

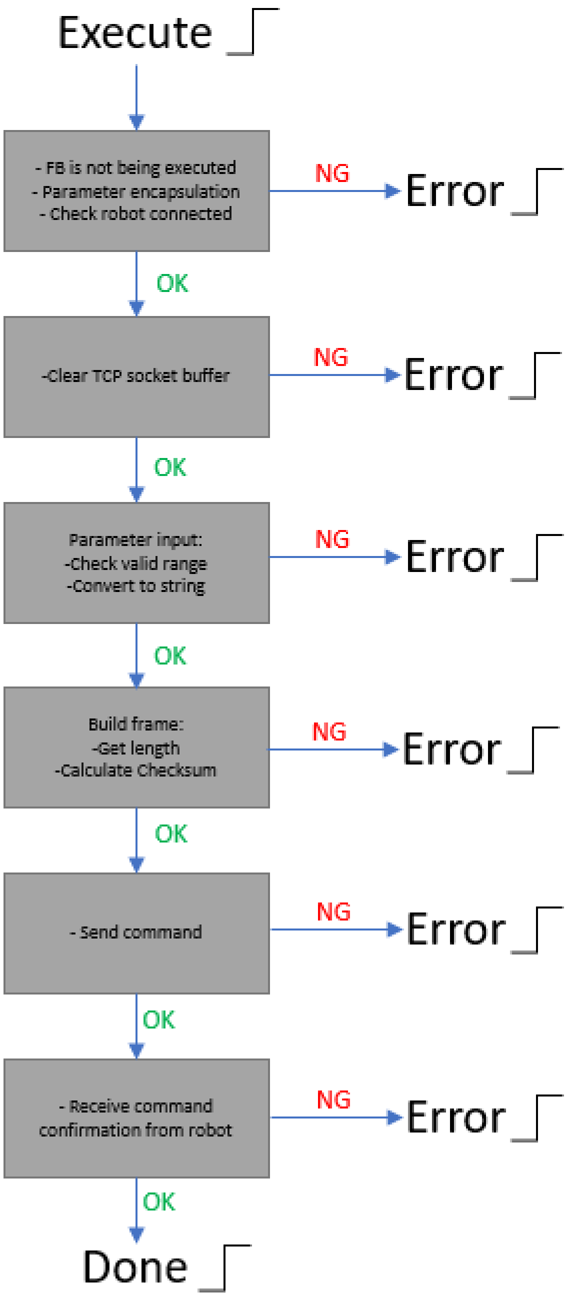


Figure 5.1.5 – Flow chart structure for all FBs in the library. On each step (square), and before executing the next step, there is a check: if there is an error or the operation is not valid, the FB triggers “Error End”. Each arrow represents where the flow goes depending on the result: *NG* means No Good, *OK* means continue with next step (source: own).

5.2. Library structure

5.2.1. Namespace and Datatypes

In the library, Function Blocks and Datatypes have been nested in Namespaces allowing to group the name of the FB and datatype definition to manage them reducing the chance of duplicated names and making the entities easier to access. Following two images show the notation for a name nested in the namespace (*CBT_Commands_Lib*) created for this library:

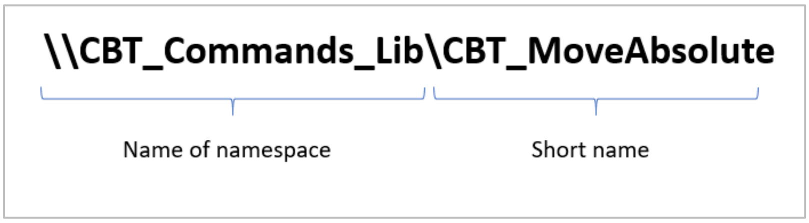


Figure 5.2.1 – Notation for a name that uses a namespace (source: own).

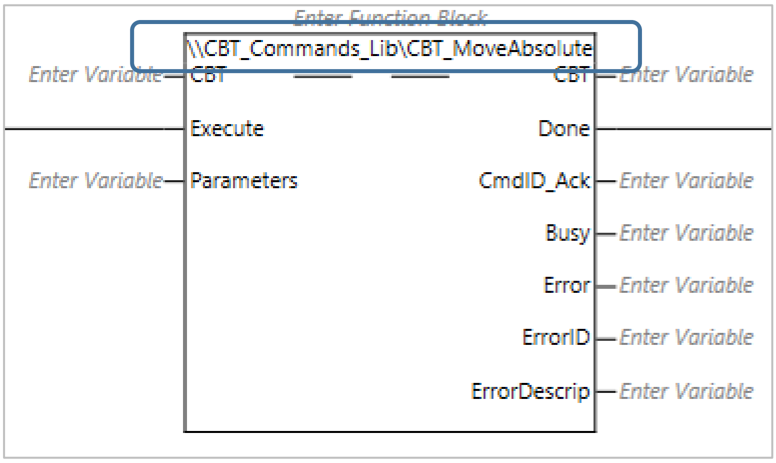
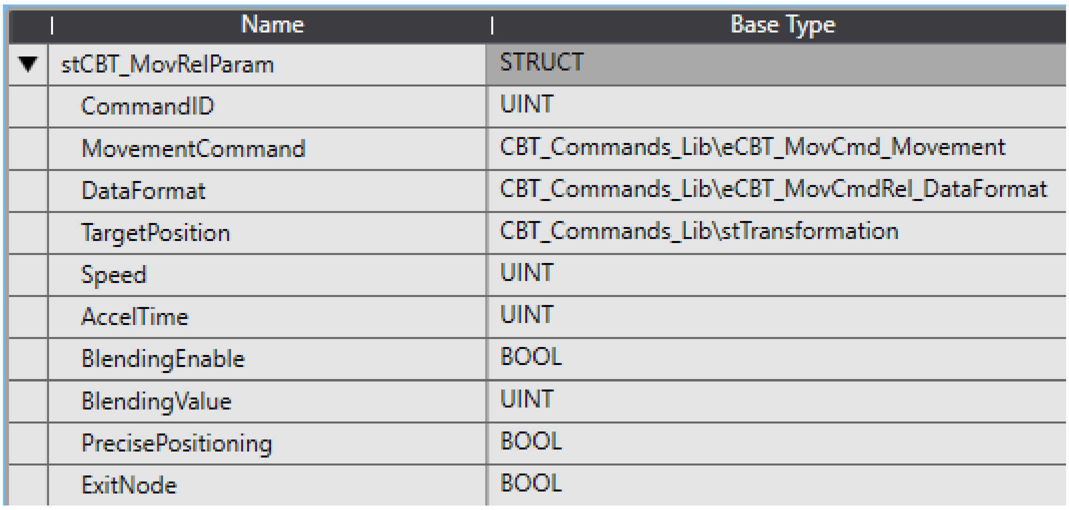


Figure 5.2.2 – Example of FB nested in a namespace with corresponding notation (source: own).

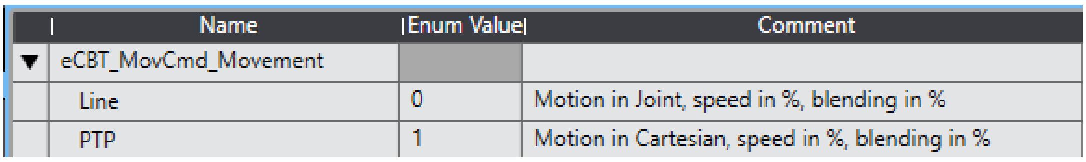
There are 2 main datatypes defined in this library. The prefix of their name identifies which kind of datatype they are:

- Datatypes starting with "stCBT_" are structure type.
- Datatypes starting with "eCBT_" are enumerated type.



	Name	Base Type
▼	stCBT_MovRelParam	STRUCT
	CommandID	UINT
	MovementCommand	CBT_Commands_Lib\eCBT_MovCmd_Movement
	DataFormat	CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat
	TargetPosition	CBT_Commands_Lib\stTransformation
	Speed	UINT
	AccelTime	UINT
	BlendingEnable	BOOL
	BlendingValue	UINT
	PrecisePositioning	BOOL
	ExitNode	BOOL

Figure 5.2.3 – Example of Structure datatype nested in CBT_Commands_Lib namespace (source: own).



	Name	Enum Value	Comment
▼	eCBT_MovCmd_Movement		
	Line	0	Motion in Joint, speed in %, blending in %
	PTP	1	Motion in Cartesian, speed in %, blending in %

Figure 5.2.4 – Example of Enumerated datatype nested in CBT_Commands_Lib namespace (source: own).

5.2.2. Function Block description

There are two types of variables for the Function Block: One type is system-defined variable, used to monitor the robot status and parameters (*In-out variables*). The second type variable are used to input arguments (*Input variables*) or to output execution status (*Output variables*). Some input variables are enumerated type which selection are made from a set of predefined enumerators.

The three variables for FB instructions are:

- *In-out variables* specify data to process with the instruction.
- *Input variables* are instruction arguments.
- *Output variables* are instruction execution status monitoring information.

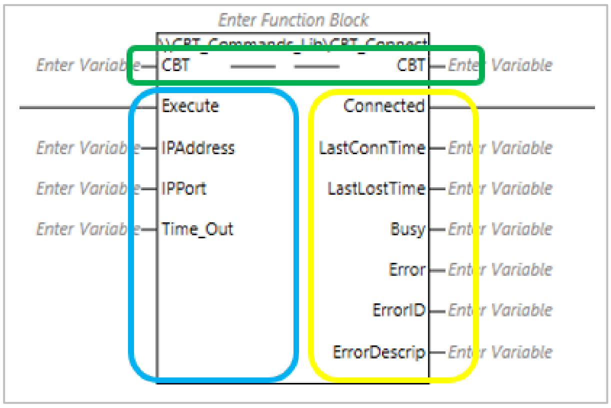


Figure 5.2.5 – System-defined variable/In-out variable (green), arguments/input variables (blue) and output status/output variables (yellow) (source: own).

5.3. Function Blocks description

Hereunder there is the description of each FB in the library. The first FB is described completely. The second one is described partially, just what is different from first FB. The FBs after are described partially, just what is different from previous FBs.

Each FB has its own purpose, but they can be classified by concept: Communications connection, robot motion, coordinate parameters and program execution control. The seven Function Blocks developed for this library are:

- CBT_Connect
- CBT_MoveAbsolute
- CBT_MoveRelative
- CBT_MoveCircle
- CBT_ChangeBase
- CBT_ChangeTCP
- CBT_ProgramControl

5.3.1. CBT_Connect

This Function Block allows to open a connection between the PLC and the collaborative robot. It also supervises the connection all the time it is in execution by intervals of 1 second:

- If robot program in TMflow is not running when *Execute* input changes to TRUE, the FB is cyclically trying to connect with the robot until robot program in TMflow runs (robot connected) or timeout is exceeded (Error End).
- If robot program in TMflow stops when connection is stablished (the FB execution is in progress), robot will close the connection (Error End).
- If *Execute* input changes to FALSE, PLC is disconnected, and connection is closed (Normal End).
- Meanwhile PLC is connected to robot, the *Connected* output remains TRUE, otherwise it changes to FALSE.
- Once the connection becomes effective, *LastConnTime* output shows date and time when the connection was done.
- Immediately after the connection breaks down, *LastLostTime* output shows date and time when the connection was lost. In *ErrorDescrip* output, the reason of the disconnection can be known.

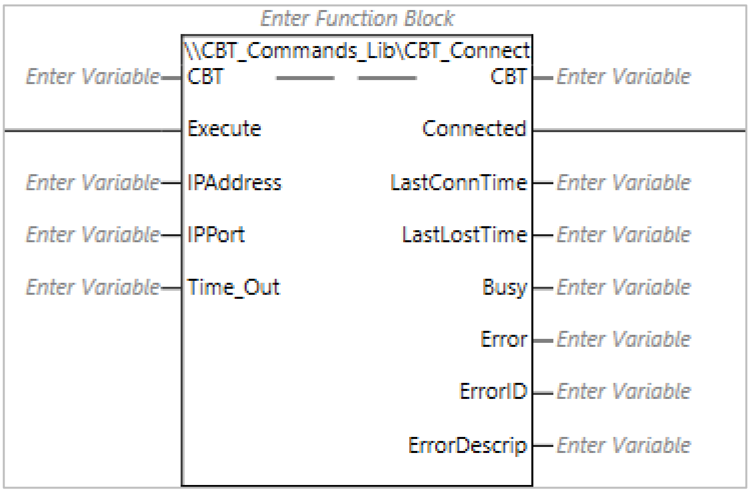


Figure 5.3.1.1 – CBT_Connect Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In-Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
IPAddress	Input	STRING[50]	---	---	Destination IP address
IPPort	Input	UINT	0 to +65535	0	Destination TCP port number
Time_Out	Input	UINT	10 to +300	0	Time in seconds in which the FB is trying to connect
Connected	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the cobot is connected
LastConnTime	Output	DATE_AND_TIME	---	---	Date time of last connection stablished
LastLostTime	Output	DATE_AND_TIME	---	---	Date time of last connection lost
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.1.1 – CBT_Connect variables type, range, default value and description (source: own).

The variable name *stCBT_Connection* is an in-out structure variable common in all Function Blocks in the Library. *stCBT_Connection* identifies the robot IP address and robot IP port, monitors if the robot is connected and if it is commanded by any Function Block in the library.

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.1.2 – stCBT_Connection variables type, range, default value and description (source: own).

CBT_Connect FB requests a connection between local TCP port number *SrcAdr.PortNo* and destination TCP port number *DstAdr.PortNo* at destination address *DstAdr.IpAdr*. Settings contained in the datatype of Socket which structure is *_sSOCKET* (specifications are as shown in Table 5.3.1.3).

Connected signal is TRUE when connection is done. The main purpose is to identify if the robot program is in Listen Node allowing to the FBs to send the command to the robot. *Connected* signal is FALSE if the robot program is not in the Listen Node thus robot cannot execute any command from the external device.

WaitingReturn signal is TRUE when a FB has sent a command to the robot and the FB is waiting to return message from robot program. Once the message is received by the PLC, *WaitingReturn* changes to FALSE.

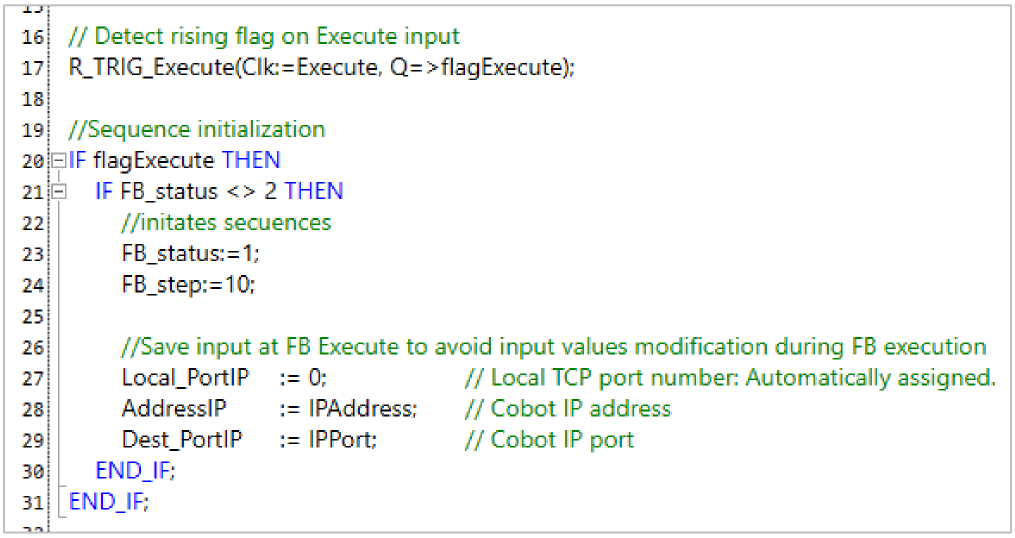
Name	I/O	Data Type	Valid Range	Default	Description
Socket		_sSOCKET	---	---	Socket
Handle		UDINT	Depends on data type	0	Handle for data communications
SrcAdr		_sSOCKET_ADDRESS	---	---	Local IP address and port number
SrcAdr.PortNo		UDINT	1 to 65535	0	Port number
SrcAdr.IpAdr		STRING	Depends on data type	' '	Source IP address
DstAdr		_sSOCKET_ADDRESS	---	---	Local IP address and port number
DstAdr.PortNo		UDINT	1 to 65535	0	Port number
DstAdr.IpAdr		STRING	Depends on data type	' '	Destination IP address

Table 5.3.1.3 – *_sSOCKET* variable structure type, range, default value and description (source: own).

Function Block Script

Once the upward signal differentiation of *Execute* input is detected, the sequence initialization is done:

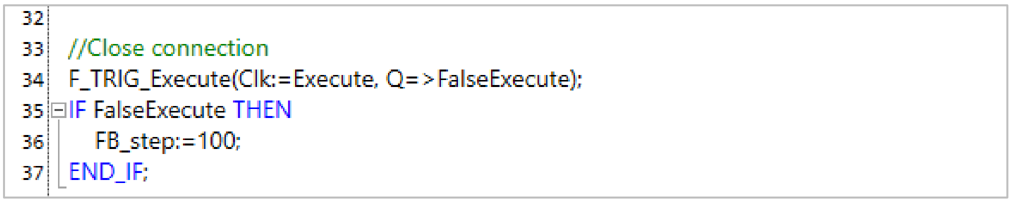
- If the FB status is different than 2 (this is no *Busy*) then, initiates the State Diagram Control (FB_status:=1). If FB is in Busy state, it omits the re-execution.
- Initiates the Algorithm Sequence (FB_Step:=10).
- Encapsulates the input parameters into internal variables avoiding the user to change those values during FB execution.



```
// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
    IF FB_status <> 2 THEN
        //initates secuences
        FB_status:=1;
        FB_step:=10;

        //Save input at FB Execute to avoid input values modification during FB execution
        Local_PortIP    := 0;           // Local TCP port number: Automatically assigned.
        AddressIP       := IPAddress;   // Cobot IP address
        Dest_PortIP     := IPPort;      // Cobot IP port
    END_IF;
END_IF;
```

Figure 5.3.1.2 – Sequence initialization similar for all FBs (source: own).

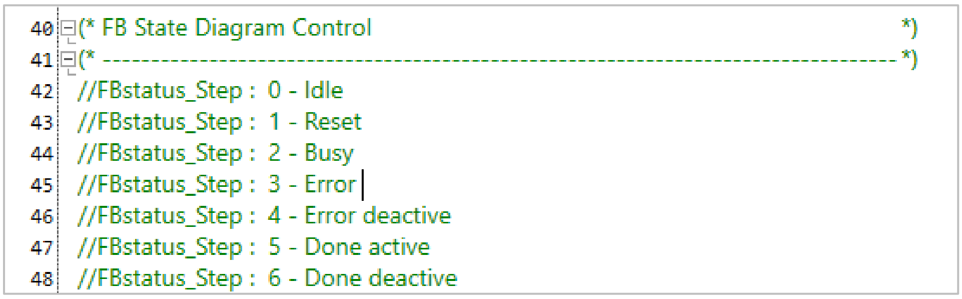
Any time in which *Execute* signal becomes FALSE, Algorithm Sequence jumps to Request Closing step (FB_Step:=100). Therefore the connection is closed and FB is finished.



```

//Close connection
F_TRIG_Execute(Clk:=Execute, Q=>FalseExecute);
IF FalseExecute THEN
    FB_step:=100;
END_IF;
```

Figure 5.3.1.3 – Close connection common for all FBs (source: own).

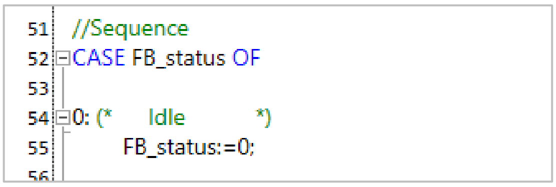
State Diagram Control is part of the script focused on output variable operation and timing. It is compoused by 7 diferent steps:



```
(* FB State Diagram Control                                                          *)
(* ---------------------------------------------------------------------------------*)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive
```

Figure 5.3.1.4 – FB State Diagram Control steps common for all FBs (source: own).

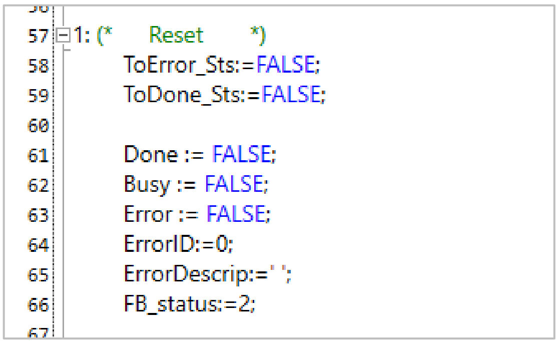
The Function Block remains in step 0 when it is idle or not in usage.



```
 //Sequence
CASE FB_status OF

0: (*     Idle          *)
      FB_status:=0;
```

Figure 5.3.1.5 – FB State Diagram Control step 0 common for all FBs (source: own).

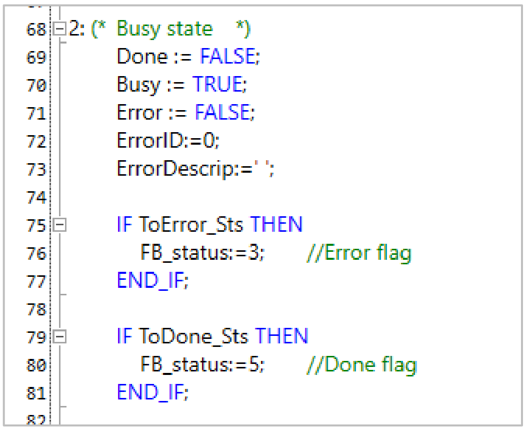
Step 1 initializes all outputs: Boolean are FALSE, Numeric are equal to 0 and String variables are emptied.



```
1: (*     Reset       *)
      ToError_Sts:=FALSE;
      ToDone_Sts:=FALSE;

      Done := FALSE;
      Busy := FALSE;
      Error := FALSE;
      ErrorID:=0;
      ErrorDescrip:=' ';
      FB_status:=2;
```

Figure 5.3.1.6 – FB State Diagram Control step 1 common for all FBs (source: own).

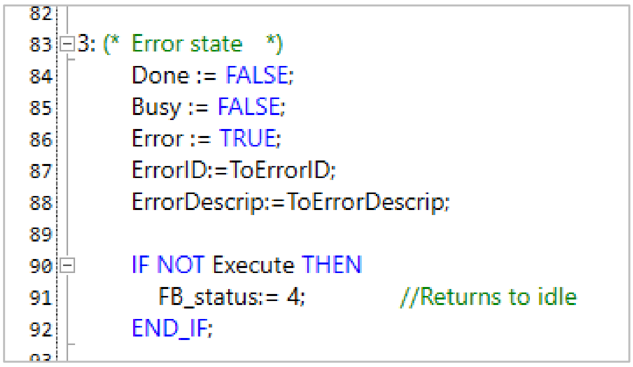
The Function Block remains in step 2 meanwhile it is under execution (*Busy* state). It waits for *Error* flag of *Done* flag from Algorithm Sequence to change its step and refresh output signals.



```
2: (*  Busy state    *)
      Done := FALSE;
      Busy := TRUE;
      Error := FALSE;
      ErrorID:=0;
      ErrorDescrip:=' ';

      IF ToError_Sts THEN
         FB_status:=3;       //Error flag
      END_IF;

      IF ToDone_Sts THEN
         FB_status:=5;       //Done flag
      END_IF;
```

Figure 5.3.1.7 – FB State Diagram Control step 2 common for all FBs (source: own).

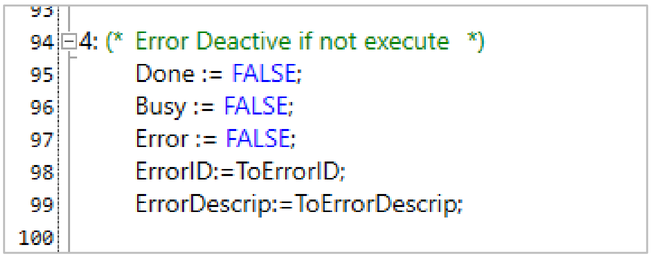
Whether an error occurs, then it moves from Step 2 to Step 3 in which *Busy* changes to FALSE and *Error* signal is TRUE. Once *Execute* signal is FALSE, it moves to Step 4. This ensures *Error* signal is TRUE at least for 1 PLC scan cycle.



```
3: (*  Error state   *)
        Done := FALSE;
        Busy := FALSE;
        Error := TRUE;
        ErrorID:=ToErrorID;
        ErrorDescrip:=ToErrorDescrip;

        IF NOT Execute THEN
            FB_status:= 4;          //Returns to idle
        END_IF;
```

Figure 5.3.1.8 – FB State Diagram Control step 3 common for all FBs (source: own).

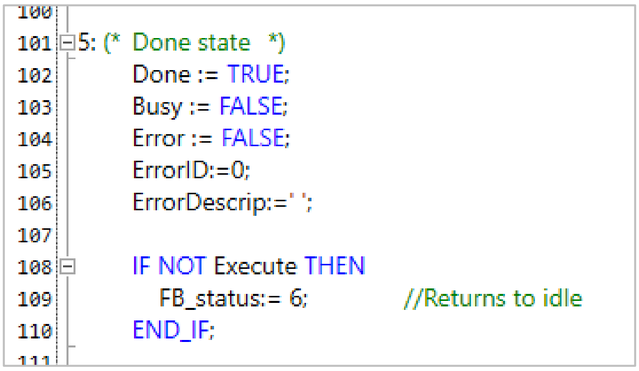
In Step 4, all boolean signals are set to FALSE, only *ErrorDescription* output monitors the information from the last error triggered.



```
4: (*  Error Deactive if not execute   *)
        Done := FALSE;
        Busy := FALSE;
        Error := FALSE;
        ErrorID:=ToErrorID;
        ErrorDescrip:=ToErrorDescrip;
```

Figure 5.3.1.9 – FB State Diagram Control step 4 common for all FBs (source: own).

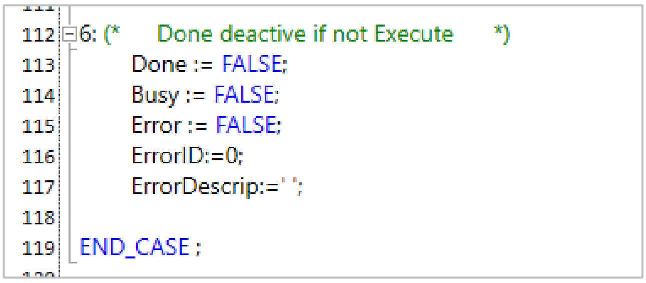
Otherwise, if execution is finished correctly, then it moves from Step 2 to Step 5 in which *Busy* changes to FALSE and *Done* signal is TRUE. Once *Execute* signal is set to FALSE, it moves to Step 6. This ensures *Done* signal is TRUE at least for 1 PLC scan cycle.



```
5: (*  Done state   *)
        Done := TRUE;
        Busy := FALSE;
        Error := FALSE;
        ErrorID:=0;
        ErrorDescrip:=' ';

        IF NOT Execute THEN
            FB_status:= 6;           //Returns to idle
        END_IF;
```

Figure 5.3.1.10 – FB State Diagram Control step 5 common for all FBs (source: own).

In Step 6, all boolean signals are set to FALSE, and *ErrorDescription* output is emptied.

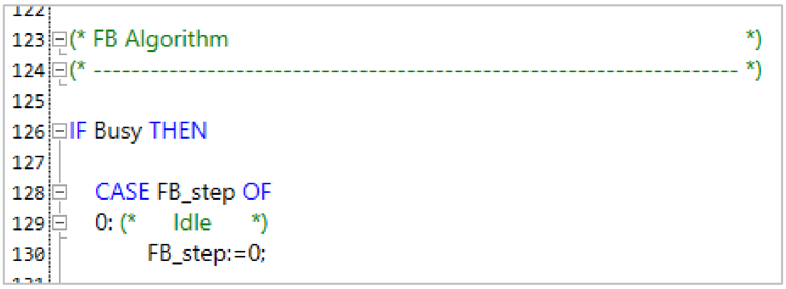


```
6: (*     Done deactive if not Execute       *)
        Done := FALSE;
        Busy := FALSE;
        Error := FALSE;
        ErrorID:=0;
        ErrorDescrip:=' ';

END_CASE ;
```

Figure 5.3.1.11 – FB State Diagram Control step 6 common for all FBs (source: own).

Whether the State Diagram Control is in Step 2, therefore in *Busy* status, the Algorithm Sequence inside the Function Block is executed:

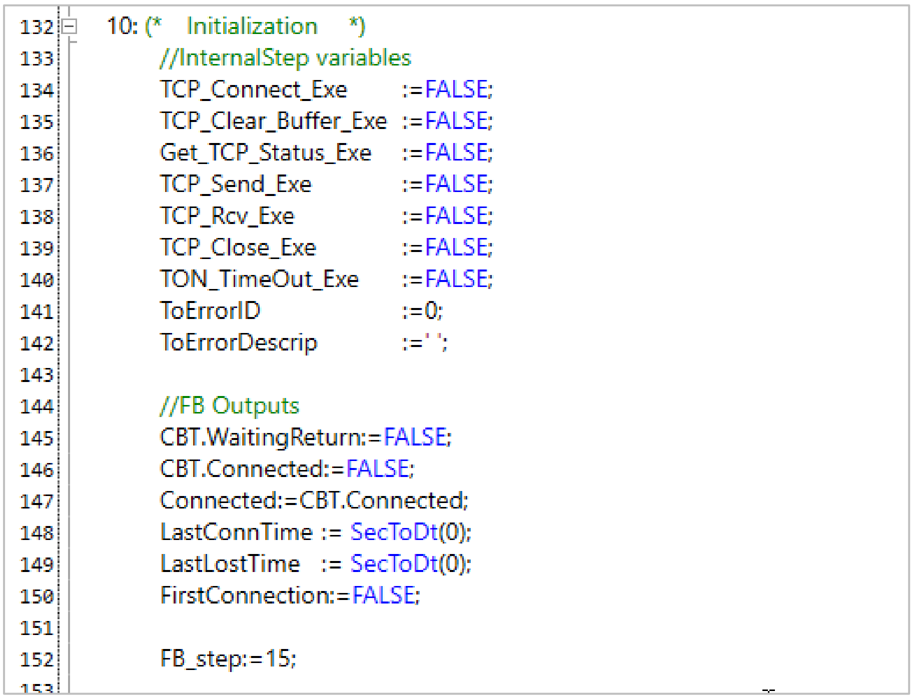
The Function Block remains in step 0 when it is idle or not in usage.



```
(* FB Algorithm                                                            *)
(* ----------------------------------------------------------------------- *)

IF Busy THEN

    CASE FB_step OF
    0: (*      Idle       *)
            FB_step:=0;
```

Figure 5.3.1.12 – CBT_Connect FB algorithm step 0 when idle (source: own).

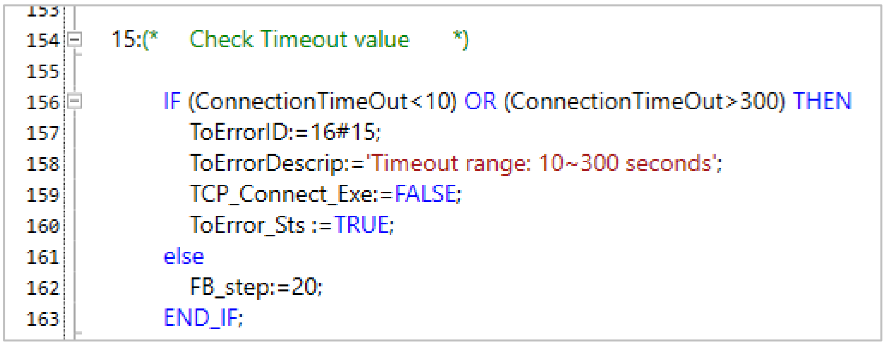
Once *Execute* input is TRUE, then FB_step is 10 and all variables are initialized.



```
10: (*   Initialization     *)
        //InternalStep variables
        TCP_Connect_Exe        :=FALSE;
        TCP_Clear_Buffer_Exe   :=FALSE;
        Get_TCP_Status_Exe     :=FALSE;
        TCP_Send_Exe           :=FALSE;
        TCP_Rcv_Exe            :=FALSE;
        TCP_Close_Exe          :=FALSE;
        TON_TimeOut_Exe        :=FALSE;
        ToErrorID              :=0;
        ToErrorDescrip         :=' ';

        //FB Outputs
        CBT.WaitingReturn:=FALSE;
        CBT.Connected:=FALSE;
        Connected:=CBT.Connected;
        LastConnTime := SecToDt(0);
        LastLostTime   := SecToDt(0);
        FirstConnection:=FALSE;

        FB_step:=15;
```

Figure 5.3.1.13 – CBT_Connect FB algorithm step 10 for variables initialization (source: own).

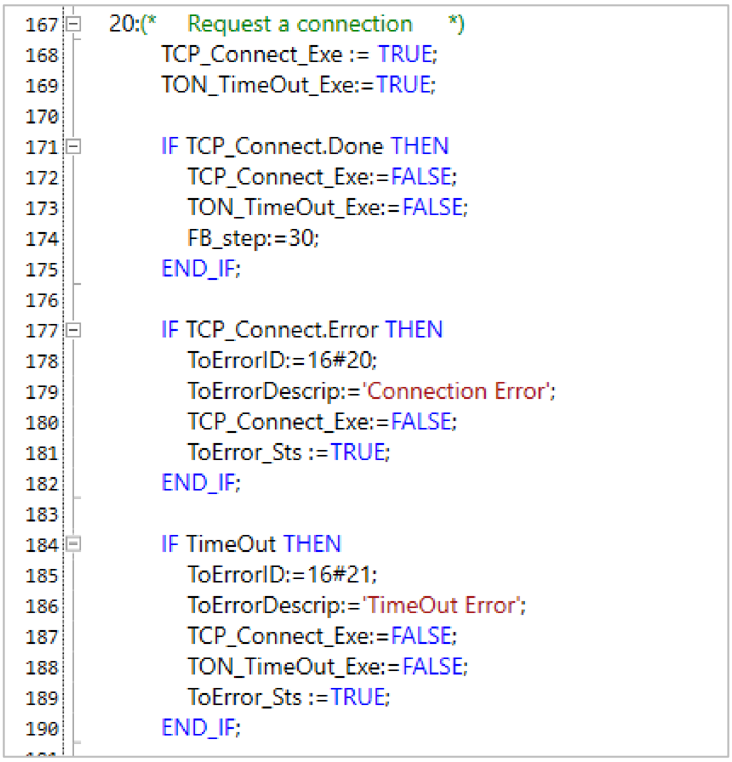
Step 15 checks if the timeout value is in the acceptable range (between 10 and 300 milliseconds).



```
15:(*    Check Timeout value      *)

        IF (ConnectionTimeOut<10) OR (ConnectionTimeOut>300) THEN
           ToErrorID:=16#15;
           ToErrorDescrip:='Timeout range: 10~300 seconds';
           TCP_Connect_Exe:=FALSE;
           ToError_Sts :=TRUE;
        else
           FB_step:=20;
        END_IF;
```

Figure 5.3.1.14 – CBT_Connect FB algorithm step 15 checking acceptable range for timeout parameter (source: own).

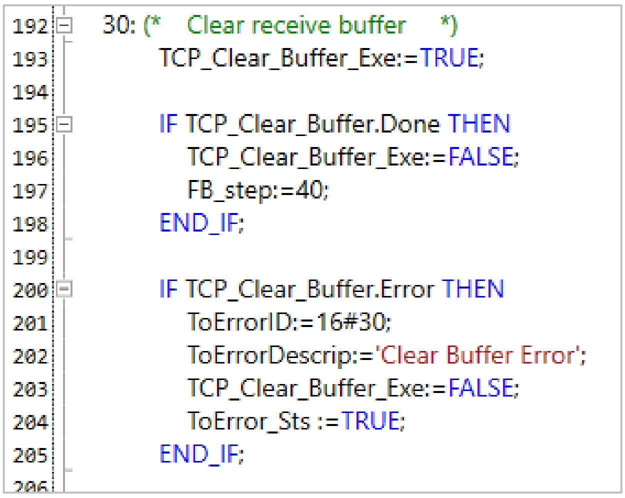
Step 20 connects built-in EtherNet/IP on the PLC to the robot TCP port. If robot program in TMflow is not running when Execute input changes to TRUE, the FB is cyclically trying to connect with the robot until robot program in TMflow runs (robot connected) or timeout is exceeded (Error End).



```
20:(*   Request a connection      *)
        TCP_Connect_Exe := TRUE;
        TON_TimeOut_Exe:=TRUE;

        IF TCP_Connect.Done THEN
            TCP_Connect_Exe:=FALSE;
            TON_TimeOut_Exe:=FALSE;
            FB_step:=30;
        END_IF;

        IF TCP_Connect.Error THEN
            ToErrorID:=16#20;
            ToErrorDescrip:='Connection Error';
            TCP_Connect_Exe:=FALSE;
            ToError_Sts :=TRUE;
        END_IF;

        IF TimeOut THEN
            ToErrorID:=16#21;
            ToErrorDescrip:='TimeOut Error';
            TCP_Connect_Exe:=FALSE;
            TON_TimeOut_Exe:=FALSE;
            ToError_Sts :=TRUE;
        END_IF;
```

Figure 5.3.1.15 – CBT_Connect FB algorithm step 20 requesting connection (source: own).

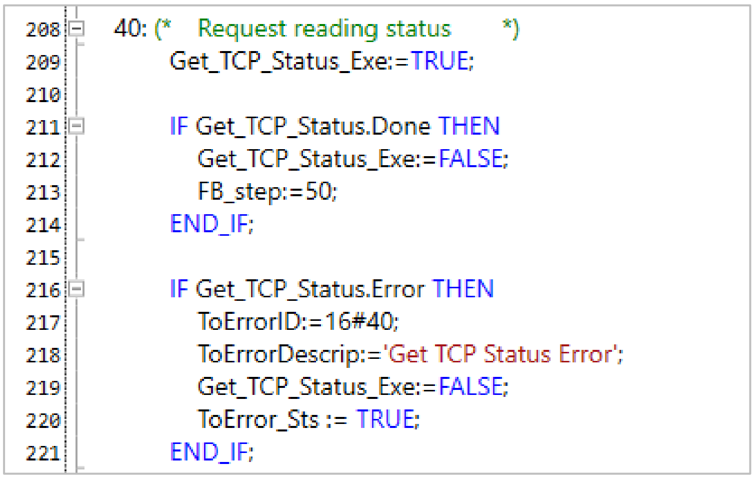
Step 30 clears the receive buffer for the TCP socket on the built-in EtherNet/IP port on the PLC.



```
30: (*   Clear receive buffer     *)
        TCP_Clear_Buffer_Exe:=TRUE;

        IF TCP_Clear_Buffer.Done THEN
            TCP_Clear_Buffer_Exe:=FALSE;
            FB_step:=40;
        END_IF;

        IF TCP_Clear_Buffer.Error THEN
            ToErrorID:=16#30;
            ToErrorDescrip:='Clear Buffer Error';
            TCP_Clear_Buffer_Exe:=FALSE;
            ToError_Sts :=TRUE;
        END_IF;
```

Figure 5.3.1.16 – CBT_Connect FB algorithm step 30 clearing the TCP received buffer (source: own).

Step 40 gets the TCP connection status of the TCP socket.



```
40: (*   Request reading status      *)
        Get_TCP_Status_Exe:=TRUE;

        IF Get_TCP_Status.Done THEN
            Get_TCP_Status_Exe:=FALSE;
            FB_step:=50;
        END_IF;

        IF Get_TCP_Status.Error THEN
            ToErrorID:=16#40;
            ToErrorDescrip:='Get TCP Status Error';
            Get_TCP_Status_Exe:=FALSE;
            ToError_Sts := TRUE;
        END_IF;
```

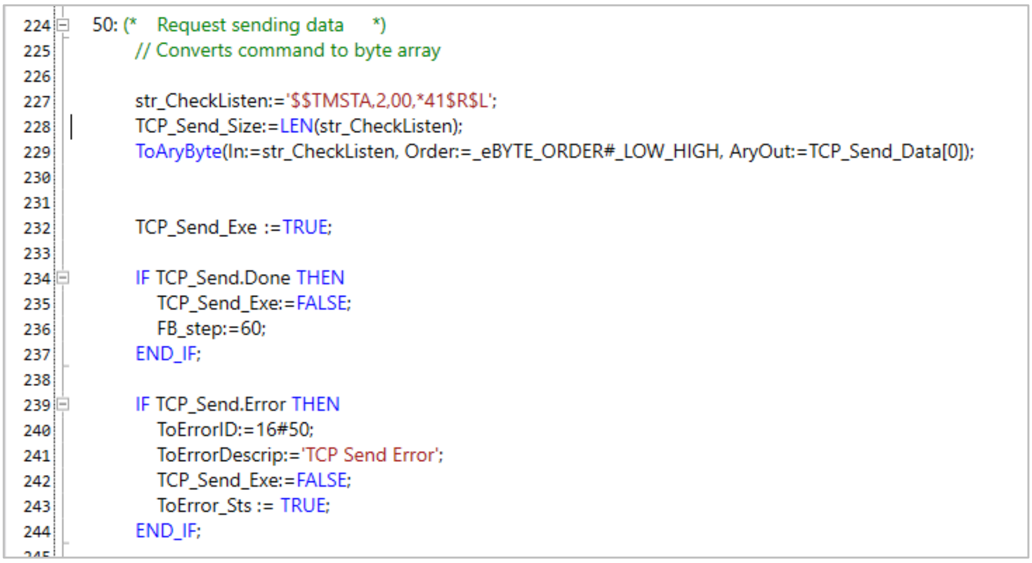
Figure 5.3.1.17 – CBT_Connect FB algorithm step 40 checking the status of the TCP socket (source: own).

Step 50 converts string type message in byte array format to be able to be sent.

- defines the string message:
  - str_CheckListen:='$$TMSTA,2,00,*41$R$L';
- finds the number of characters in a text string:
  - TCP_Send_Size:=LEN(str_CheckListen);
- separates a variable into bytes and stores them in a BYTE array:
  - ToAryByte(In:=str_CheckListen, Order:=_eBYTE_ORDER#_LOW_HIGH, AryOut:=TCP_Send_Data[0]);
- sends the str_CheckListen message to the robot.

Description of (str_CheckListen:='$$TMSTA,2,00,*41$R$L';):

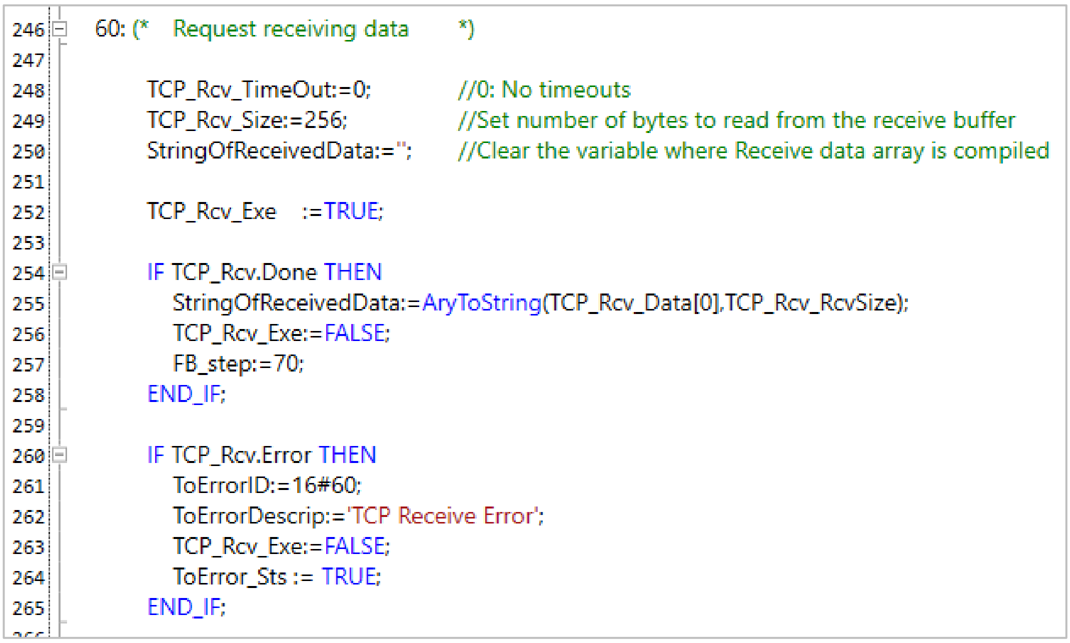
- $$TMSTA → Communication package acquiring status // Sysmac syntax requires $$, the 1st is sysmac systax to recognize the 2nd $ as character.
- 2 → Indicates the length of 00 is 2 bytes.
- 00 → Indicates if cobot is in external script control mode or not (is Cobot in Listen node or not)
- 41 → Checksum
- $R → $R in Sysmac syntax is \R in ASCII (carriage)
- $L → $L in Sysmac syntax is \L in ASCII (enter)



```
50: (*   Request sending data     *)
      // Converts command to byte array

      str_CheckListen:='$$TMSTA,2,00,*41$R$L';
      TCP_Send_Size:=LEN(str_CheckListen);
      ToAryByte(In:=str_CheckListen, Order:=_eBYTE_ORDER#_LOW_HIGH, AryOut:=TCP_Send_Data[0]);


      TCP_Send_Exe :=TRUE;

      IF TCP_Send.Done THEN
         TCP_Send_Exe:=FALSE;
         FB_step:=60;
      END_IF;

      IF TCP_Send.Error THEN
         ToErrorID:=16#50;
         ToErrorDescrip:='TCP Send Error';
         TCP_Send_Exe:=FALSE;
         ToError_Sts := TRUE;
      END_IF;
```

Figure 5.3.1.18 – CBT_Connect FB algorithm step 50 sending the message to get connection with robot (source: own).

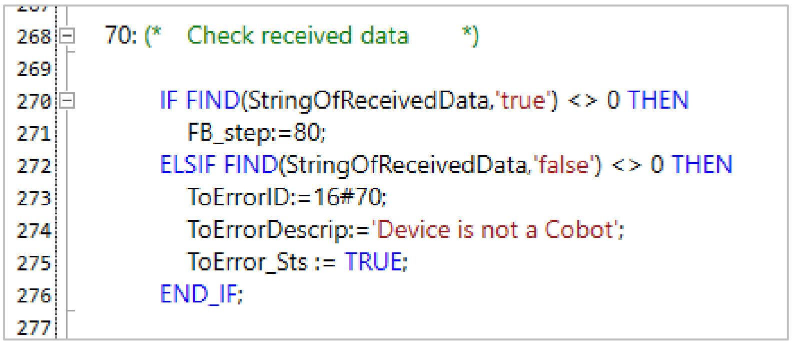
Step 60 reads the data from the receive buffer for a TCP socket on the built-in EtherNet/IP port on the PLC.



```
60: (*   Request receiving data      *)

      TCP_Rcv_TimeOut:=0;             //0: No timeouts
      TCP_Rcv_Size:=256;              //Set number of bytes to read from the receive buffer
      StringOfReceivedData:='';       //Clear the variable where Receive data array is compiled

      TCP_Rcv_Exe    :=TRUE;

      IF TCP_Rcv.Done THEN
         StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_RcvSize);
         TCP_Rcv_Exe:=FALSE;
         FB_step:=70;
      END_IF;

      IF TCP_Rcv.Error THEN
         ToErrorID:=16#60;
         ToErrorDescrip:='TCP Receive Error';
         TCP_Rcv_Exe:=FALSE;
         ToError_Sts := TRUE;
      END_IF;
```

Figure 5.3.1.19 – CBT_Connect FB algorithm step 60 receiving TCP message from robot (source: own).

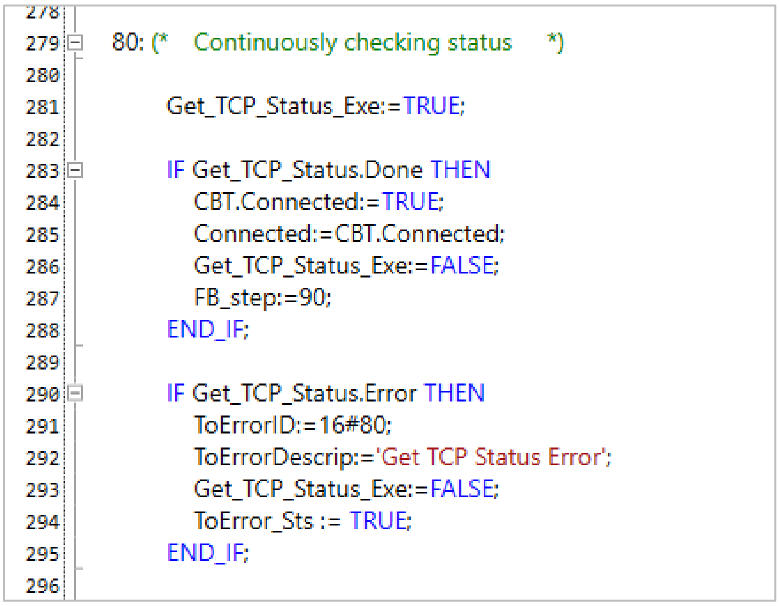
Step 70 checks the data received from robot to TCP socket in the PLC. If the message includes the word "true", the robot is in Listen Node and the connection can be done (moves to step 80). Otherwise, if the message includes the word "false", the robot is not on Listen Node, connection cannot be done and error is triggered.



```
70: (*   Check received data        *)

        IF FIND(StringOfReceivedData,'true') <> 0 THEN
            FB_step:=80;
        ELSIF FIND(StringOfReceivedData,'false') <> 0 THEN
            ToErrorID:=16#70;
            ToErrorDescrip:='Device is not a Cobot';
            ToError_Sts := TRUE;
        END_IF;

```

Figure 5.3.1.20 – CBT_Connect FB algorithm step 70 checking data received (source: own).

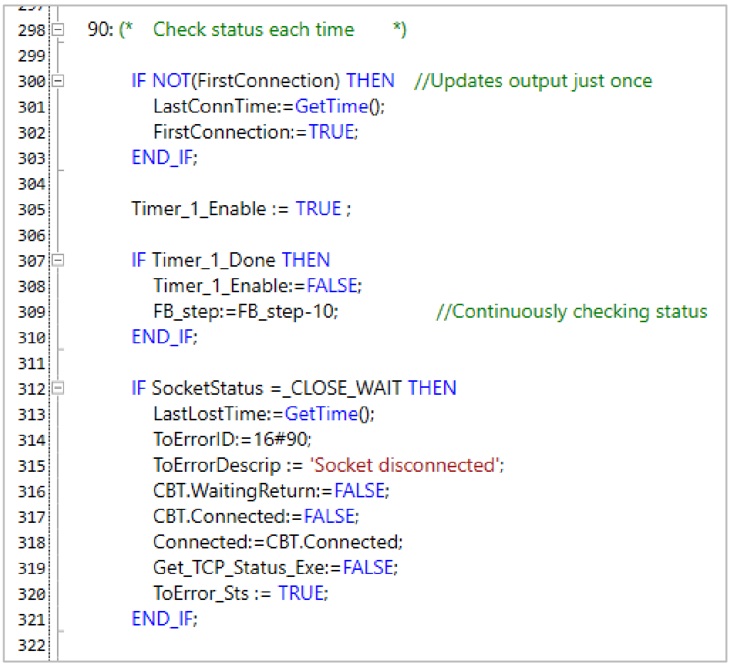
Step 80 gets the TCP connection status of the TCP socket.



```
80: (*   Continuously checking status      *)

        Get_TCP_Status_Exe:=TRUE;

        IF Get_TCP_Status.Done THEN
            CBT.Connected:=TRUE;
            Connected:=CBT.Connected;
            Get_TCP_Status_Exe:=FALSE;
            FB_step:=90;
        END_IF;

        IF Get_TCP_Status.Error THEN
            ToErrorID:=16#80;
            ToErrorDescrip:='Get TCP Status Error';
            Get_TCP_Status_Exe:=FALSE;
            ToError_Sts := TRUE;
        END_IF;

```

Figure 5.3.1.21 – CBT_Connect FB algorithm step 80 gets TCP connection status (source: own).

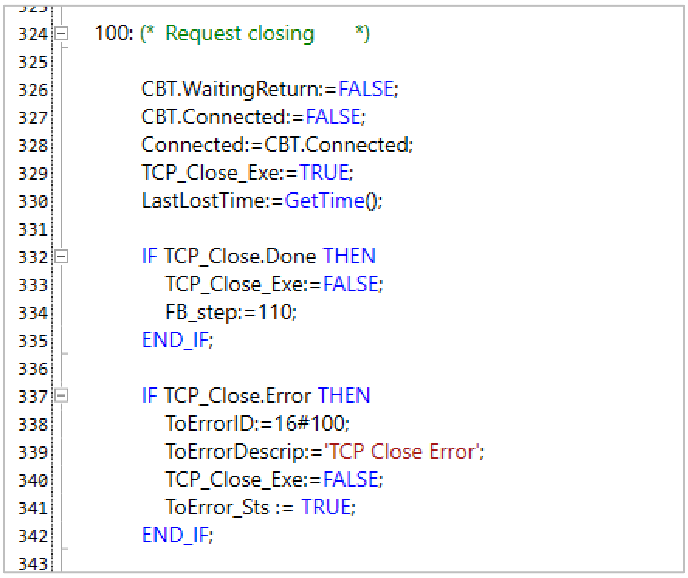
Step 90 is cyclically checking if the connection is still alive, otherwise *CBT.Connected* changes to FALSE. If robot program stops when connection is stablished (the FB execution is in progress), robot will close the connection (Error End).



```
90: (*   Check status each time      *)

    IF NOT(FirstConnection) THEN   //Updates output just once
        LastConnTime:=GetTime();
        FirstConnection:=TRUE;
    END_IF;

    Timer_1_Enable := TRUE ;

    IF Timer_1_Done THEN
        Timer_1_Enable:=FALSE;
        FB_step:=FB_step-10;             //Continuously checking status
    END_IF;

    IF SocketStatus =_CLOSE_WAIT THEN
        LastLostTime:=GetTime();
        ToErrorID:=16#90;
        ToErrorDescrip := 'Socket disconnected';
        CBT.WaitingReturn:=FALSE;
        CBT.Connected:=FALSE;
        Connected:=CBT.Connected;
        Get_TCP_Status_Exe:=FALSE;
        ToError_Sts := TRUE;
    END_IF;

```

Figure 5.3.1.22 – CBT_Connect FB algorithm step 90 continuously checking connection (source: own).

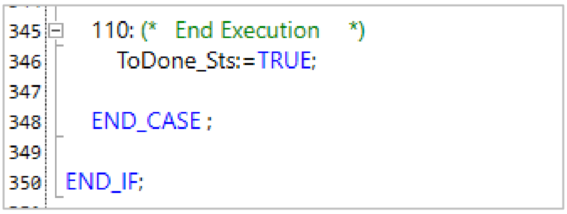
If *Execute* input changes to FALSE, then step 100 is executed.



```
100: (*  Request closing      *)

    CBT.WaitingReturn:=FALSE;
    CBT.Connected:=FALSE;
    Connected:=CBT.Connected;
    TCP_Close_Exe:=TRUE;
    LastLostTime:=GetTime();

    IF TCP_Close.Done THEN
        TCP_Close_Exe:=FALSE;
        FB_step:=110;
    END_IF;

    IF TCP_Close.Error THEN
        ToErrorID:=16#100;
        ToErrorDescrip:='TCP Close Error';
        TCP_Close_Exe:=FALSE;
        ToError_Sts := TRUE;
    END_IF;

```

Figure 5.3.1.23 – CBT_Connect FB algorithm step 100 closing the TCP socket connection (source: own).

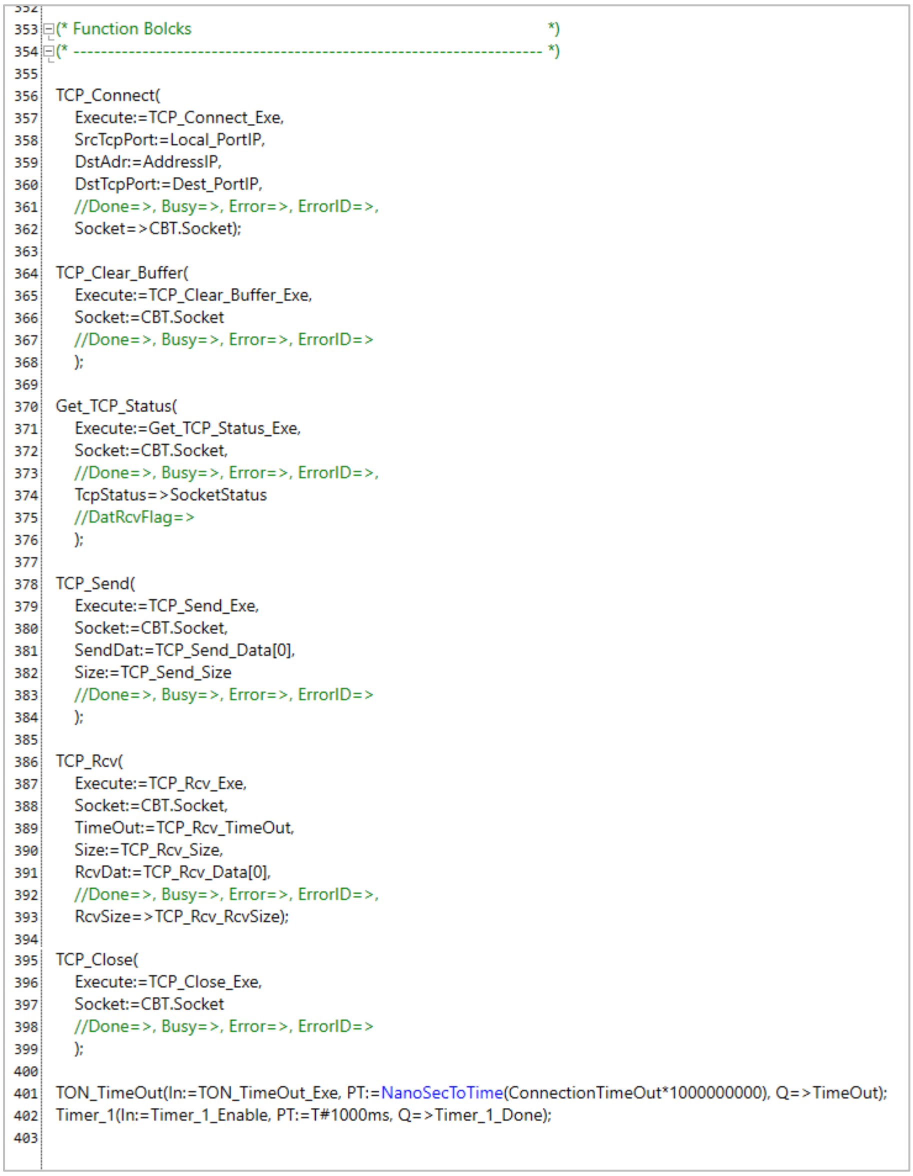
Step 110 finishes the execution of the Function Block.



```
    110: (*  End Execution    *)
        ToDone_Sts:=TRUE;

    END_CASE ;

END_IF;
```

Figure 5.3.1.24 – CBT_Connect FB algorithm step 110 finishing the execution (source: own).

The last part of the script is reserved for TCP related FBs included be default in Sysmac Studio.



```
(* Function Bolcks                                                  *)
(* ---------------------------------------------------------------- *)

TCP_Connect(
    Execute:=TCP_Connect_Exe,
    SrcTcpPort:=Local_PortIP,
    DstAdr:=AddressIP,
    DstTcpPort:=Dest_PortIP,
    //Done=>, Busy=>, Error=>, ErrorID=>,
    Socket=>CBT.Socket);

TCP_Clear_Buffer(
    Execute:=TCP_Clear_Buffer_Exe,
    Socket:=CBT.Socket
    //Done=>, Busy=>, Error=>, ErrorID=>
    );

Get_TCP_Status(
    Execute:=Get_TCP_Status_Exe,
    Socket:=CBT.Socket,
    //Done=>, Busy=>, Error=>, ErrorID=>,
    TcpStatus=>SocketStatus
    //DatRcvFlag=>
    );

TCP_Send(
    Execute:=TCP_Send_Exe,
    Socket:=CBT.Socket,
    SendDat:=TCP_Send_Data[0],
    Size:=TCP_Send_Size
    //Done=>, Busy=>, Error=>, ErrorID=>
    );

TCP_Rcv(
    Execute:=TCP_Rcv_Exe,
    Socket:=CBT.Socket,
    TimeOut:=TCP_Rcv_TimeOut,
    Size:=TCP_Rcv_Size,
    RcvDat:=TCP_Rcv_Data[0],
    //Done=>, Busy=>, Error=>, ErrorID=>,
    RcvSize=>TCP_Rcv_RcvSize);

TCP_Close(
    Execute:=TCP_Close_Exe,
    Socket:=CBT.Socket
    //Done=>, Busy=>, Error=>, ErrorID=>
    );

TON_TimeOut(In:=TON_TimeOut_Exe, PT:=NanoSecToTime(ConnectionTimeOut*1000000000), Q=>TimeOut);
Timer_1(In:=Timer_1_Enable, PT:=T#1000ms, Q=>Timer_1_Done);
```

Figure 5.3.1.25 – CBT_Connect FB internal functions instances for TCP communications (source: own).

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
15	Timeout range: 10~300 seconds	Set parameter in acceptable range
20	Connection Error	Use CBT_Connect FB for connection
21	TimeOut Error	Cobot is not in Listen Node
30	Clear Buffer Error	Check Ethernet connection wiring
40	Get TCP Status Error	Check Ethernet connection wiring
50	TCP Send Error	Check Ethernet connection wiring
60	TCP Receive Error	Check Ethernet connection wiring
70	Target device is not a Cobot	Cobot is not in Listen Node
80	Get TCP Status Error	Check Ethernet connection wiring
90	Socket disconnected	Check Ethernet connection wiring
100	TCP Close Error	Check Ethernet connection wiring

Table 5.3.1.4 – CBT_Connect error list description and action (source: own).

5.3.2. CBT_MoveAbsolute

This Function Block sends the command to move the robot to an absolute target position. Two different types of motion can be defined:

- PTP: Robot moves to the target point along the closest path of the joint angle space.
- Line: Tool moves to the target point in a straight line.

Among other parameters, user must set the Cobot In/out variable to identify the target robot with its IP address and communications port, the command ID to identify the command in the feedback sent by the robot, speed including units, acceleration time, if blending is required and if it is performed by percentage or by radius, and finally the robot arm configuration. This last parameter defines if the target position will be achieved with lefty or righty configuration, with above or below configuration and with flip or noflip configuration when the movement is defined as PTP (Point to point). Arm configuration is not available for Linear interpolated movement, in that case the arm configuration is kept as it was at the moment the motion started.

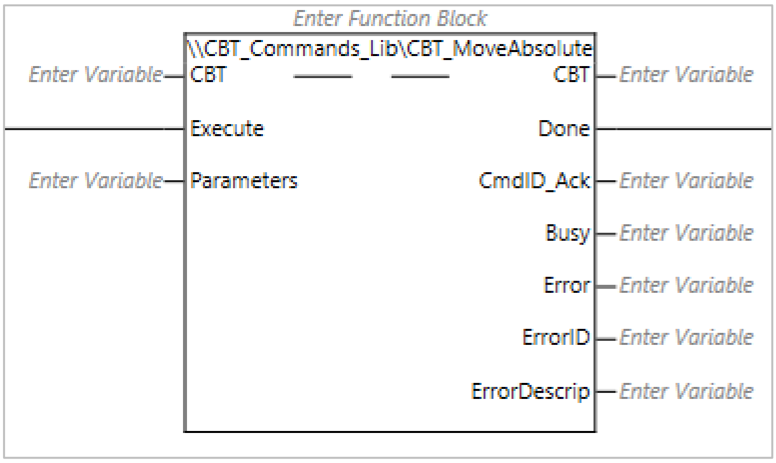


Figure 5.3.2.1 – CBT_MoveAbsolute Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In/Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
Parameters	Input	CBT_Commands_Lib\stCBT_MovAbsParam	---	---	Motion parameters
Done	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is completed
CmdID_Ack	Output	UINT	0 to +65535	0	Command identification number
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.2.1 – CBT_MoveAbsolute variables type, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.2.2 – stCBT_Connection datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
CommandID	UINT	2 to 9	0	Identification for the return message for this command
MovementCommand	CBT_Commands_Lib\ eCBT_MovCmd_Movement	---	---	Motion type
DataFormat	CBT_Commands_Lib\ eCBT_MovCmdAbs_DataFormat	---	---	Motion strategy
TargetPosition	CBT_Commands_Lib\ stCBT_Transformation	---	---	Target position
Speed	UINT	0 to 4500	0	Speed expressed as a percentage (%) or in velocity (mm/s)
AccelTime	UINT	150 to 9999	0	Time interval to accelerate to top speed (ms)
BlendingEnable	BOOL	TRUE or FALSE	FALSE	Enables blending with next motion command
BlendingValue	UINT	0 to 100	0	Blending value expressed as a percentage (%) or in radius (mm)
PrecisePositioning	BOOL	TRUE or FALSE	FALSE	Whether robot moves to the point precisely
RobotPoseEnable	BOOL	TRUE or FALSE	FALSE	Enables if arm configuration can be decided
RobotPose	CBT_Commands_Lib\ stCBT_RobotPose	---	---	Arm configuration type
ExitNode	BOOL	TRUE or FALSE	FALSE	Quit Listen node in robot program

Table 5.3.2.3 – stCBT_MovAbsParam datatype, range, default value and description (source: own).

Name	Enum Value	Description
Line	0	Motion in Joint, speed in %, blending in %
PTP	1	Motion in Cartesian, speed in %, blending in %

Table 5.3.2.4 – eCBT_MovCmd_Movement datatype, value and description (source: own).

Name	Enum Value	Description
JPP_Abs	0	Motion in Joint, speed in %, blending in %
CPP_Abs	1	Motion in Cartesian, speed in %, blending in %
CPR_Abs	2	Motion in Cartesian, speed in %, blending in radius
CAP_Abs	3	Motion in Cartesian, speed in mm/s, blending in %
CAR_Abs	4	Motion in Cartesian, speed in mm/s, blending in radius

Table 5.3.2.5 – eCBT_MovCmdAbs_DataFormat datatype, value and description (source: own).

Name	Data Type	Valid Range	Default	Description
X	REAL	---	0	X coordinates in mm
Y	REAL	---	0	Y coordinates in mm
Z	REAL	---	0	Z coordinates in mm
RX	REAL	---	0	RX coordinates in degrees
RY	REAL	---	0	RY coordinates in degrees
RZ	REAL	---	0	RZ coordinates in degrees

Table 5.3.2.6 – stCBT_Transformation datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Pose1	UINT	0 to 1	0	RIGHTY = 0, LEFTY = 1
Pose2	UINT	2 to 3	0	ABOVE = 2, BELOW = 3
Pose3	UINT	4 to 5	0	NOFLIP = 4, FLIP = 5

Table 5.3.2.7 – stCBT_RobotPose datatype, range, default value and description (source: own).

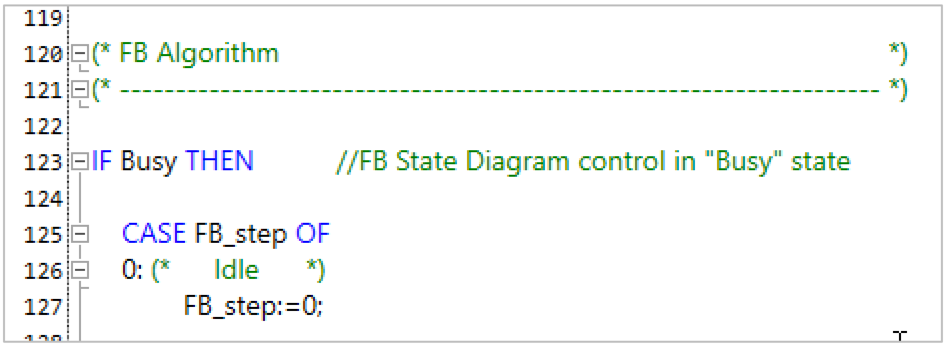
Function Block Script

As all Function Blocks in this Library, once the upward signal differentiation of *Execute* input is detected and it is not in *Busy* state, the sequence initialization is started, otherwise an error is triggered. Sequence initialization has been described in *5.3.1.CBT_Connect* section.

As all Function Blocks in this Library, State Diagram Control is part of the script focused on output variable operation and timing. State Diagram Control has been described in *5.3.1.CBT_Connect* section.

Hereunder, the description of the specific script of CBT_MoveAbsolute FB:

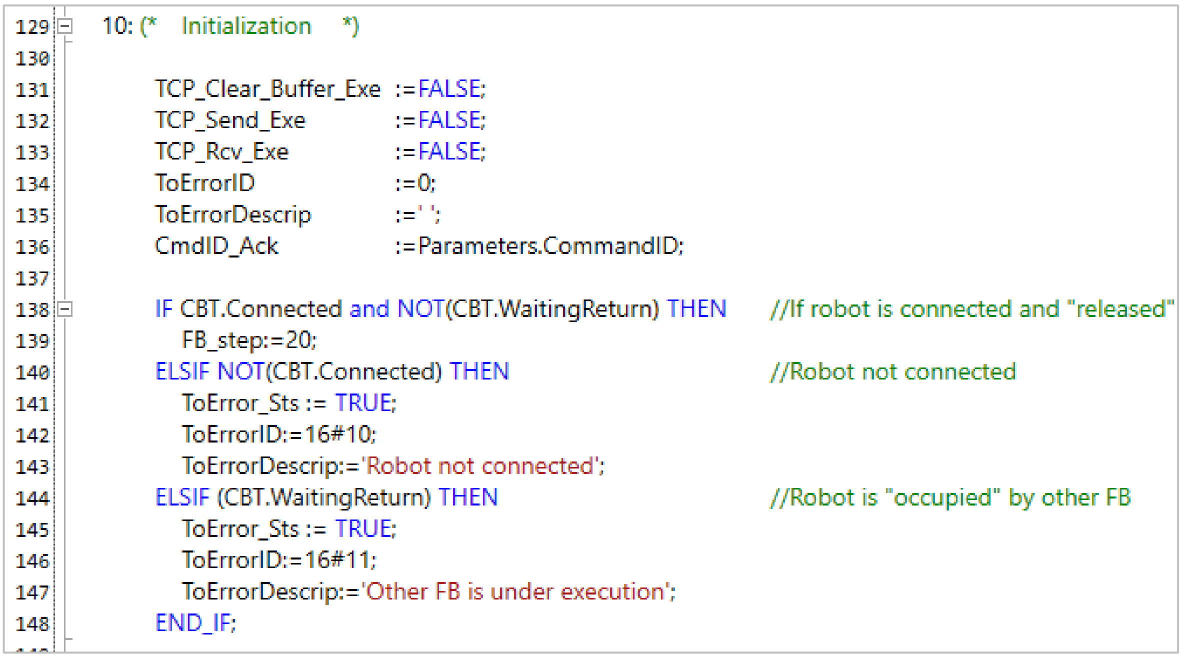
It remains in step 0 when it is idle or not in usage.



```

(* FB Algorithm                                                      *)
(* ----------------------------------------------------------------- *)

IF Busy THEN        //FB State Diagram control in "Busy" state

    CASE FB_step OF
    0: (*     Idle      *)
            FB_step:=0;
```

Figure 5.3.2.2 – CBT_MoveAbsolute FB algorithm step 0 when idle (source: own).

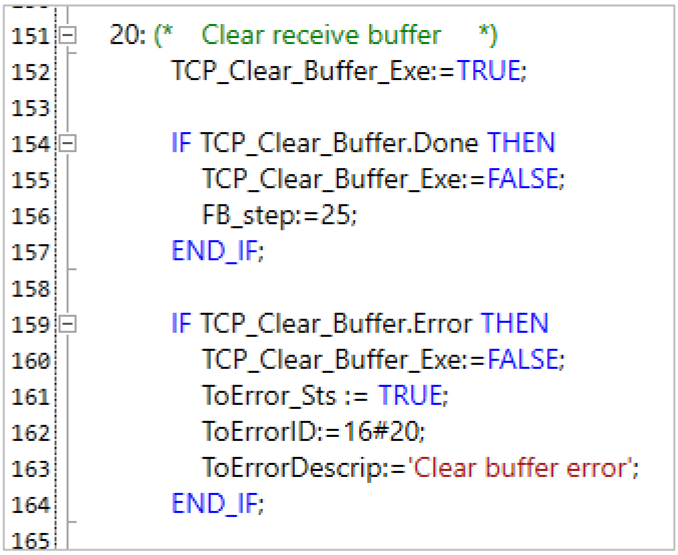
Once *Execute* input is TRUE, then FB_step changes to 10, all variables are initialized and checks if the robot is connected and free to be used (not occupied by other FB).



```
10: (*   Initialization    *)

        TCP_Clear_Buffer_Exe  :=FALSE;
        TCP_Send_Exe          :=FALSE;
        TCP_Rcv_Exe           :=FALSE;
        ToErrorID             :=0;
        ToErrorDescrip        :=' ';
        CmdID_Ack             :=Parameters.CommandID;

        IF CBT.Connected and NOT(CBT.WaitingReturn) THEN      //If robot is connected and "released"
            FB_step:=20;
        ELSIF NOT(CBT.Connected) THEN                         //Robot not connected
            ToError_Sts := TRUE;
            ToErrorID:=16#10;
            ToErrorDescrip:='Robot not connected';
        ELSIF (CBT.WaitingReturn) THEN                        //Robot is "occupied" by other FB
            ToError_Sts := TRUE;
            ToErrorID:=16#11;
            ToErrorDescrip:='Other FB is under execution';
        END_IF;
```

Figure 5.3.2.3 – CBT_MoveAbsolute FB algorithm step 10 for variables initialization, checking connection and robot ready (source: own).

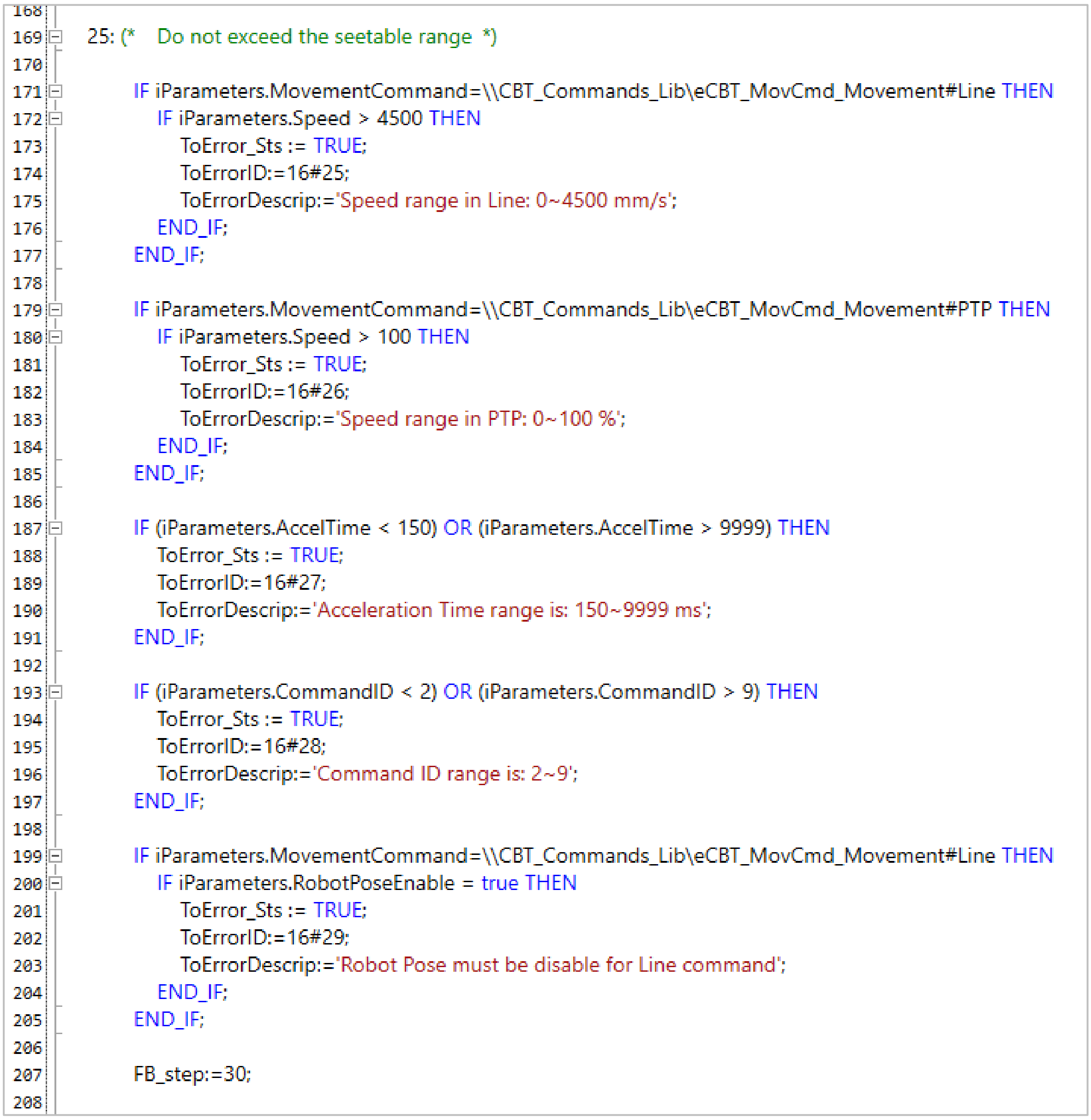
Step 20 clears the receive buffer for the TCP socket on the built-in EtherNet/IP on the PLC.



```
20: (*   Clear receive buffer    *)
        TCP_Clear_Buffer_Exe:=TRUE;

        IF TCP_Clear_Buffer.Done THEN
            TCP_Clear_Buffer_Exe:=FALSE;
            FB_step:=25;
        END_IF;

        IF TCP_Clear_Buffer.Error THEN
            TCP_Clear_Buffer_Exe:=FALSE;
            ToError_Sts := TRUE;
            ToErrorID:=16#20;
            ToErrorDescrip:='Clear buffer error';
        END_IF;
```

Figure 5.3.2.4 – CBT_MoveAbsolute FB algorithm step 20 clearing the TCP received buffer (source: own).

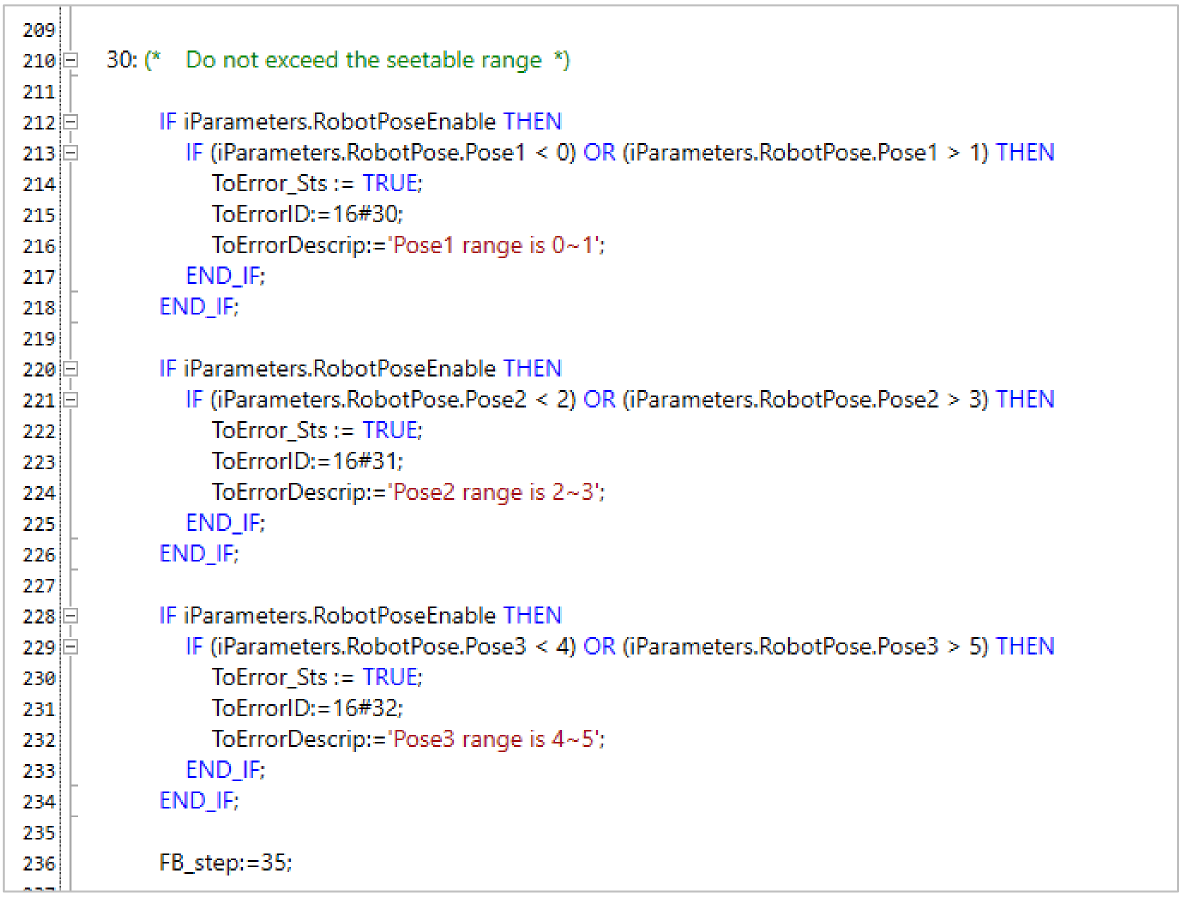
Step 25 verifies if the values for motion parameters are within the acceptable range.



```
25: (*   Do not exceed the seetable range  *)

    IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line THEN
        IF iParameters.Speed > 4500 THEN
            ToError_Sts := TRUE;
            ToErrorID:=16#25;
            ToErrorDescrip:='Speed range in Line: 0~4500 mm/s';
        END_IF;
    END_IF;

    IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP THEN
        IF iParameters.Speed > 100 THEN
            ToError_Sts := TRUE;
            ToErrorID:=16#26;
            ToErrorDescrip:='Speed range in PTP: 0~100 %';
        END_IF;
    END_IF;

    IF (iParameters.AccelTime < 150) OR (iParameters.AccelTime > 9999) THEN
        ToError_Sts := TRUE;
        ToErrorID:=16#27;
        ToErrorDescrip:='Acceleration Time range is: 150~9999 ms';
    END_IF;

    IF (iParameters.CommandID < 2) OR (iParameters.CommandID > 9) THEN
        ToError_Sts := TRUE;
        ToErrorID:=16#28;
        ToErrorDescrip:='Command ID range is: 2~9';
    END_IF;

    IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line THEN
        IF iParameters.RobotPoseEnable = true THEN
            ToError_Sts := TRUE;
            ToErrorID:=16#29;
            ToErrorDescrip:='Robot Pose must be disable for Line command';
        END_IF;
    END_IF;

    FB_step:=30;
```

Figure 5.3.2.5 – CBT_MoveAbsolute FB algorithm step 25 checking acceptable value range for motion parameters (source: own).

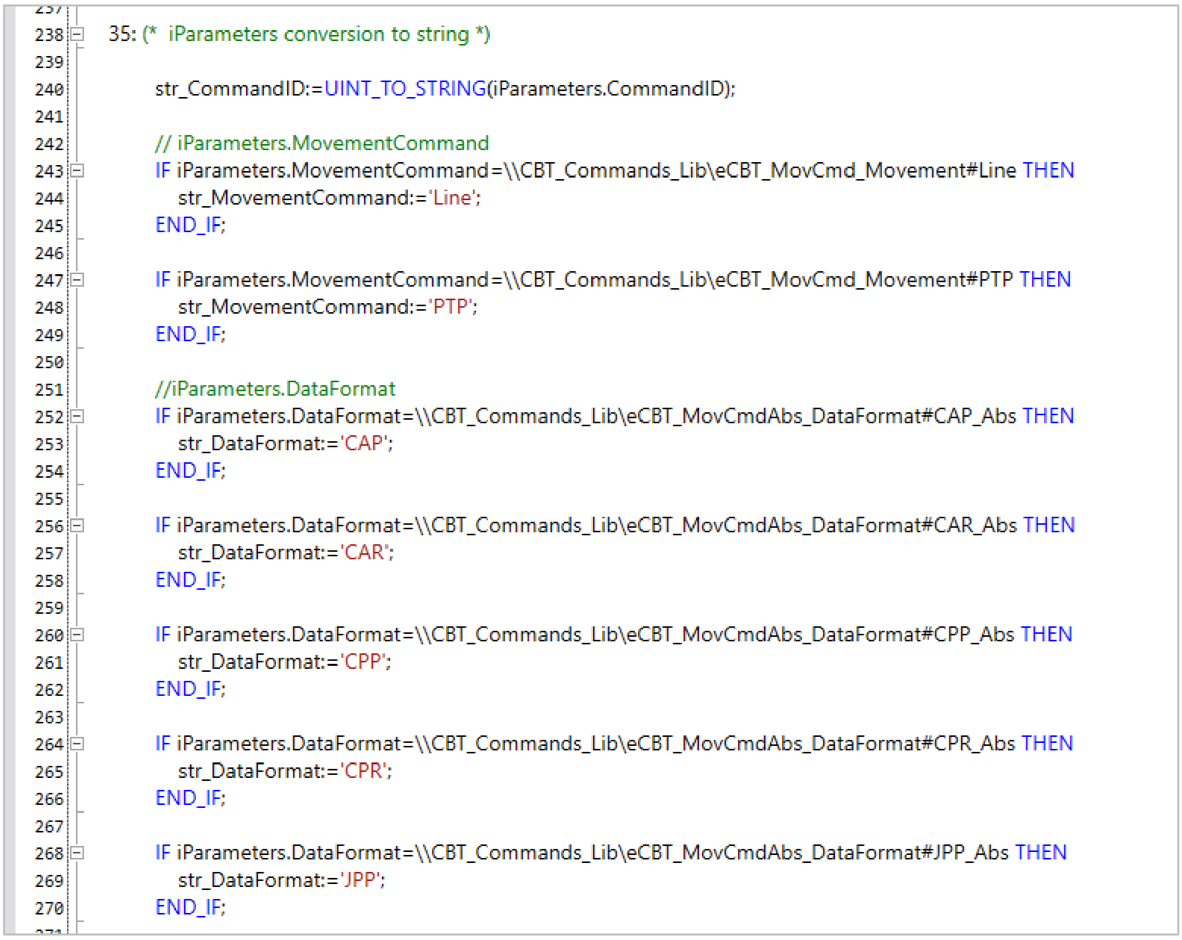
Step 30 detects if the values in the arm configuration parameter are within the acceptable range.



```
30: (*   Do not exceed the seetable range  *)

    IF iParameters.RobotPoseEnable THEN
      IF (iParameters.RobotPose.Pose1 < 0) OR (iParameters.RobotPose.Pose1 > 1) THEN
        ToError_Sts := TRUE;
        ToErrorID:=16#30;
        ToErrorDescrip:='Pose1 range is 0~1';
      END_IF;
    END_IF;

    IF iParameters.RobotPoseEnable THEN
      IF (iParameters.RobotPose.Pose2 < 2) OR (iParameters.RobotPose.Pose2 > 3) THEN
        ToError_Sts := TRUE;
        ToErrorID:=16#31;
        ToErrorDescrip:='Pose2 range is 2~3';
      END_IF;
    END_IF;

    IF iParameters.RobotPoseEnable THEN
      IF (iParameters.RobotPose.Pose3 < 4) OR (iParameters.RobotPose.Pose3 > 5) THEN
        ToError_Sts := TRUE;
        ToErrorID:=16#32;
        ToErrorDescrip:='Pose3 range is 4~5';
      END_IF;
    END_IF;

    FB_step:=35;
```

Figure 5.3.2.6 – CBT_MoveAbsolute FB algorithm step 30 checking acceptable value range in arm configuration parameter (source: own).

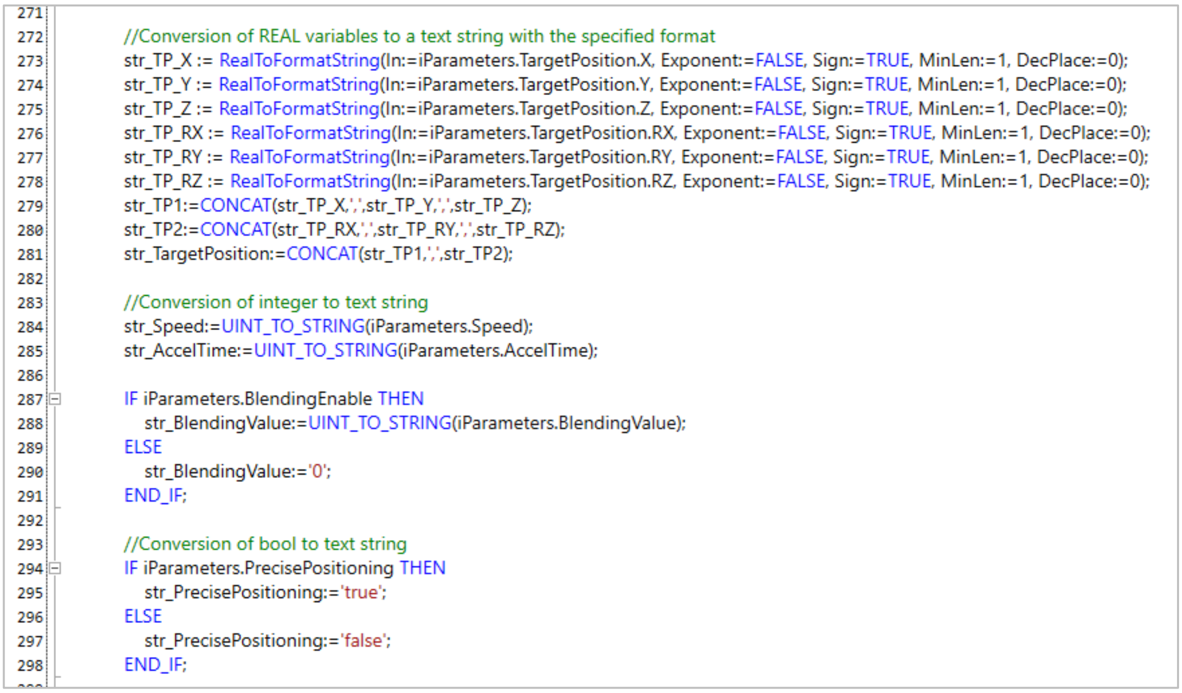
Step 35 creates the corresponding string variable (str_MovementCommand) depending on the movement command parameter and the string variable (str_DataFormat) for the data format parameter.



```
35: (*  iParameters conversion to string *)

    str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

    // iParameters.MovementCommand
    IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line THEN
        str_MovementCommand:='Line';
    END_IF;

    IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP THEN
        str_MovementCommand:='PTP';
    END_IF;

    //iParameters.DataFormat
    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CAP_Abs THEN
        str_DataFormat:='CAP';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CAR_Abs THEN
        str_DataFormat:='CAR';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CPP_Abs THEN
        str_DataFormat:='CPP';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CPR_Abs THEN
        str_DataFormat:='CPR';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#JPP_Abs THEN
        str_DataFormat:='JPP';
    END_IF;
```

Figure 5.3.2.7 – CBT_MoveAbsolute FB algorithm step 35 creating the string of the data format parameter (source: own).

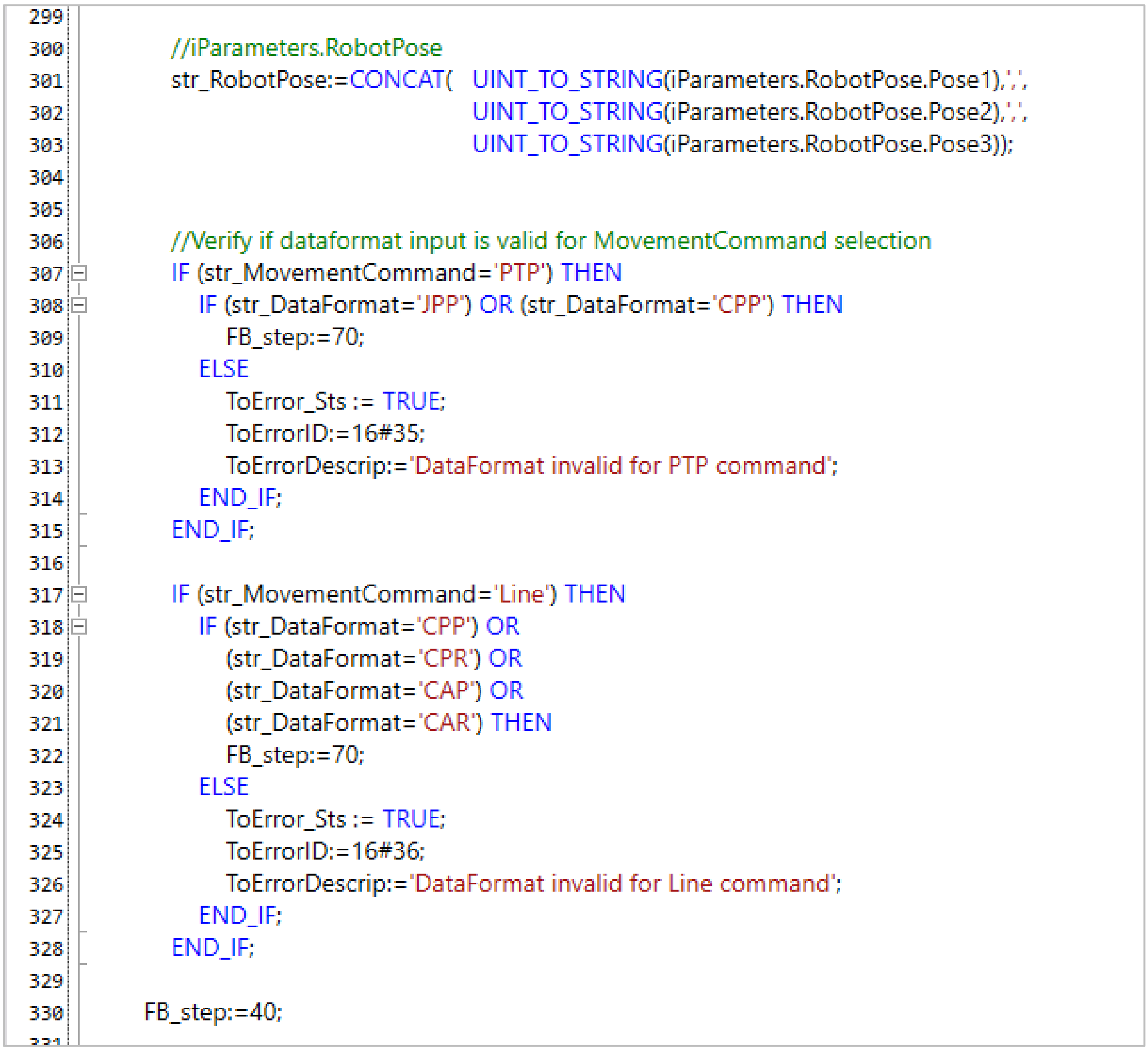
It converts target position parameter from real to string (str_TargetPosition), speed parameter to string variable (str_Speed), acceleration time to string (str_AccelTime), blending value to string (str_BlendingValue) and precise positioning to string (str_PrecisePositioning).



```
//Conversion of REAL variables to a text string with the specified format
str_TP_X := RealToFormatString(In:=iParameters.TargetPosition.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
str_TP_Y := RealToFormatString(In:=iParameters.TargetPosition.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
str_TP_Z := RealToFormatString(In:=iParameters.TargetPosition.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
str_TP_RX := RealToFormatString(In:=iParameters.TargetPosition.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
str_TP_RY := RealToFormatString(In:=iParameters.TargetPosition.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
str_TP_RZ := RealToFormatString(In:=iParameters.TargetPosition.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
str_TP1:=CONCAT(str_TP_X,',',str_TP_Y,',',str_TP_Z);
str_TP2:=CONCAT(str_TP_RX,',',str_TP_RY,',',str_TP_RZ);
str_TargetPosition:=CONCAT(str_TP1,',',str_TP2);

//Conversion of integer to text string
str_Speed:=UINT_TO_STRING(iParameters.Speed);
str_AccelTime:=UINT_TO_STRING(iParameters.AccelTime);

IF iParameters.BlendingEnable THEN
    str_BlendingValue:=UINT_TO_STRING(iParameters.BlendingValue);
ELSE
    str_BlendingValue:='0';
END_IF;

//Conversion of bool to text string
IF iParameters.PrecisePositioning THEN
    str_PrecisePositioning:='true';
ELSE
    str_PrecisePositioning:='false';
END_IF;
```

Figure 5.3.2.8 – CBT_MoveAbsolute FB algorithm step 35 creating the string variables for the target position, speed value, acceleration time value, blending value and precise positioning value (source: own).

It joins the robot pose parameters (str_RobotPose) and verifies if the data format selected is acceptable for the selected movement command.



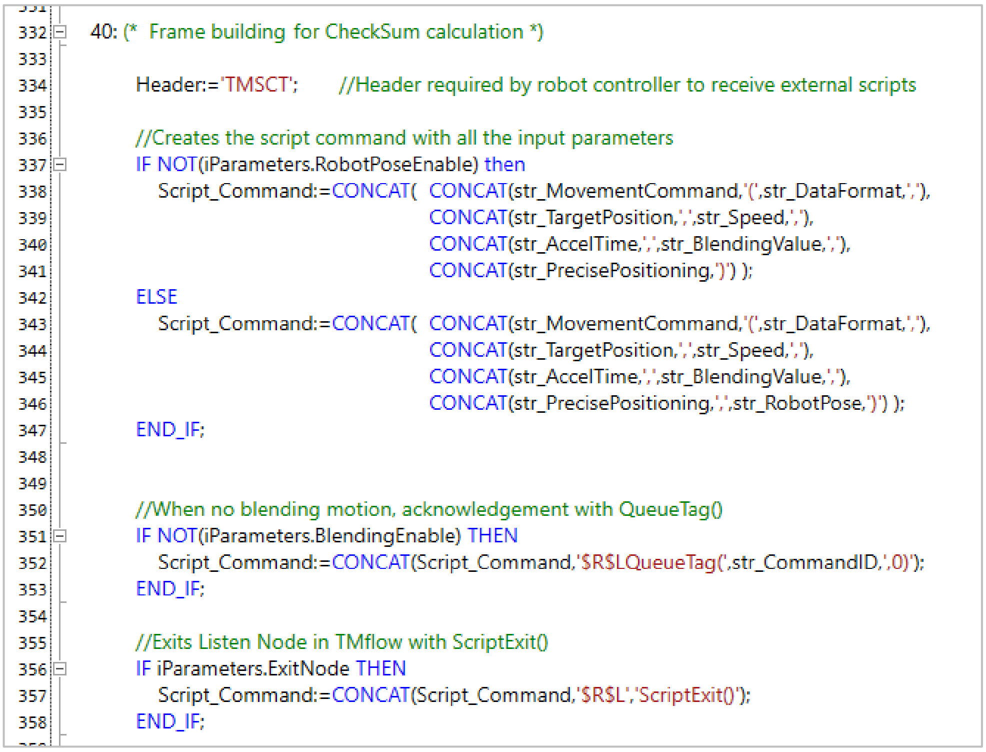
```
        //iParameters.RobotPose
        str_RobotPose:=CONCAT(   UINT_TO_STRING(iParameters.RobotPose.Pose1),',',
                                 UINT_TO_STRING(iParameters.RobotPose.Pose2),',',
                                 UINT_TO_STRING(iParameters.RobotPose.Pose3));


        //Verify if dataformat input is valid for MovementCommand selection
        IF (str_MovementCommand='PTP') THEN
            IF (str_DataFormat='JPP') OR (str_DataFormat='CPP') THEN
                FB_step:=70;
            ELSE
                ToError_Sts := TRUE;
                ToErrorID:=16#35;
                ToErrorDescrip:='DataFormat invalid for PTP command';
            END_IF;
        END_IF;

        IF (str_MovementCommand='Line') THEN
            IF (str_DataFormat='CPP') OR
                (str_DataFormat='CPR') OR
                (str_DataFormat='CAP') OR
                (str_DataFormat='CAR') THEN
                FB_step:=70;
            ELSE
                ToError_Sts := TRUE;
                ToErrorID:=16#36;
                ToErrorDescrip:='DataFormat invalid for Line command';
            END_IF;
        END_IF;

    FB_step:=40;
```

Figure 5.3.2.9 – CBT_MoveAbsolute FB algorithm step 35 creating the string of the arm configuration parameter and checks if there is some incorrect data format parameter depending on the movement command (source: own).

Step 40 builds the message (script_command) to be sent to the robot including:

- Movement command
- Data format
- Target position
- Speed
- Acceleration time
- Blending value
- Precise positioning
- Robot pose

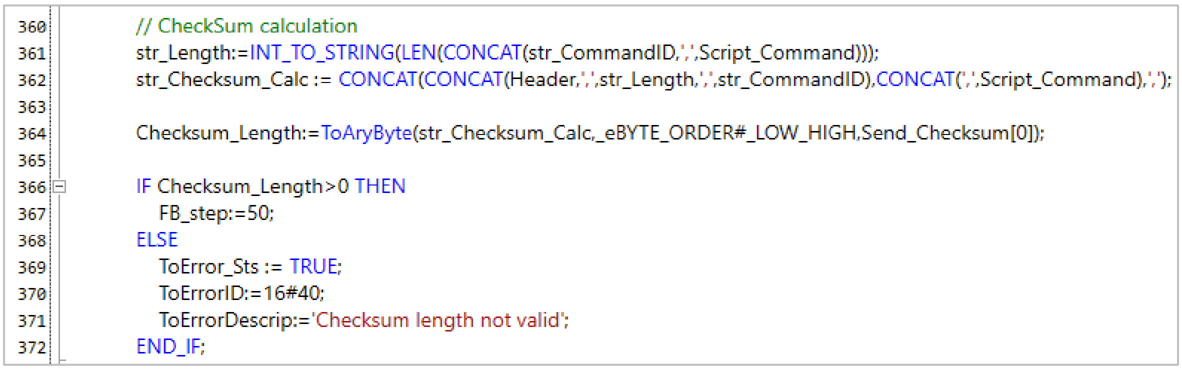
- ➢ If blending is disabled, QueueTag() command is included in the message. It identifies the robot motion with the CommandID number for the acknowledge when robot motion finishes the current robot motion in process. Hence, *Done* output changes to TRUE when the robot movement has been completed.
- ➢ If ExitNode is enabled, ScriptExit() command is included in the message. It exits external script control mode once the last motion command has been executed and finished. Listen Node is quitted, and not further external script functions commands are accepted.



```
40: (*  Frame building  for CheckSum calculation *)

        Header:='TMSCT';       //Header required by robot controller to receive external scripts

        //Creates the script command with all the input parameters
        IF NOT(iParameters.RobotPoseEnable) then
            Script_Command:=CONCAT(  CONCAT(str_MovementCommand,'(',str_DataFormat,','),
                                     CONCAT(str_TargetPosition,',',str_Speed,','),
                                     CONCAT(str_AccelTime,',',str_BlendingValue,','),
                                     CONCAT(str_PrecisePositioning,')') );
        ELSE
            Script_Command:=CONCAT(  CONCAT(str_MovementCommand,'(',str_DataFormat,','),
                                     CONCAT(str_TargetPosition,',',str_Speed,','),
                                     CONCAT(str_AccelTime,',',str_BlendingValue,','),
                                     CONCAT(str_PrecisePositioning,',',str_RobotPose,')') );
        END_IF;


        //When no blending motion, acknowledgement with QueueTag()
        IF NOT(iParameters.BlendingEnable) THEN
            Script_Command:=CONCAT(Script_Command,'$R$LQueueTag(',str_CommandID,',0)');
        END_IF;

        //Exits Listen Node in TMflow with ScriptExit()
        IF iParameters.ExitNode THEN
            Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
        END_IF;
```

Figure 5.3.2.10 – CBT_MoveAbsolute FB algorithm step 40 building the message with the complete command (source: own).

Once the message has been built, it is needed to find the number of characters of (str_CommandID + Script_Command) and to store it in the string (str_Length). Then, joining all parts (Header + str_Length + str_CommandID + Script_Command) to calculate the checksum (str_Checksum_Calc).

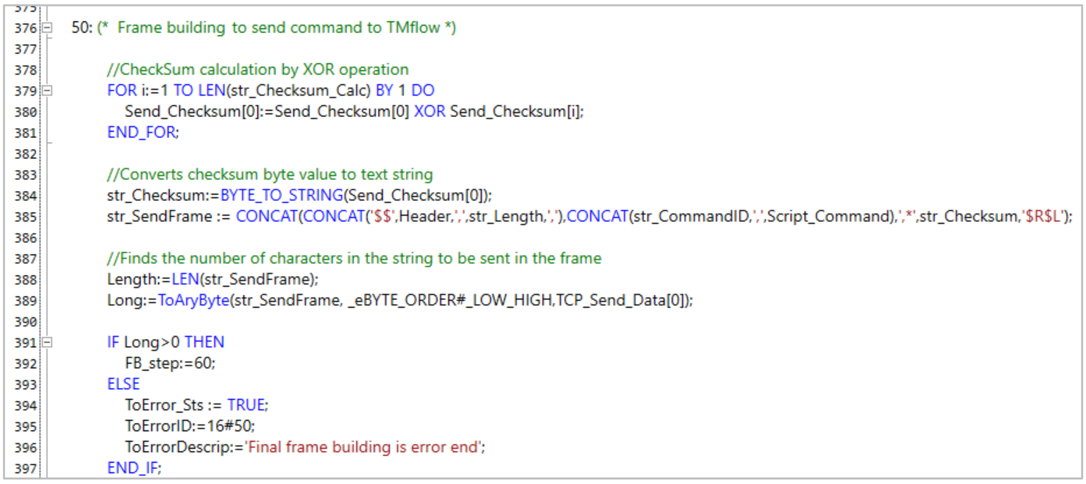


```
        // CheckSum calculation
        str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));
        str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');

        Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);

        IF Checksum_Length>0 THEN
            FB_step:=50;
        ELSE
            ToError_Sts := TRUE;
            ToErrorID:=16#40;
            ToErrorDescrip:='Checksum length not valid';
        END_IF;
```

Figure 5.3.2.11 – CBT_MoveAbsolute FB algorithm step 40 building the message with the complete command and finds the number of characters in the string (source: own).

Step 50 calculates, with XOR operation, the string for the checksum (str_Checksum) and creates the complete message to be sent (str_SendFrame) composed by:

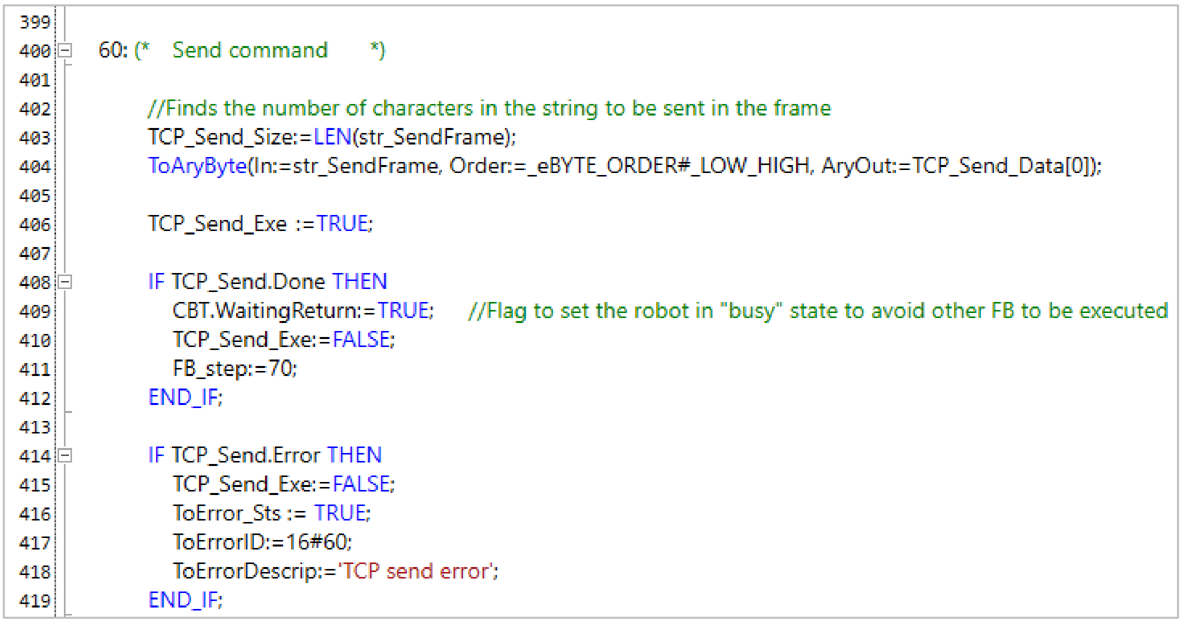
- Header
- Length of the message
- Command ID
- String command
- CheckSum calculation



```
    50: (*  Frame building  to send command to TMflow *)

        //CheckSum calculation by XOR operation
        FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
            Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
        END_FOR;

        //Converts checksum byte value to text string
        str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
        str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');

        //Finds the number of characters in the string to be sent in the frame
        Length:=LEN(str_SendFrame);
        Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

        IF Long>0 THEN
            FB_step:=60;
        ELSE
            ToError_Sts := TRUE;
            ToErrorID:=16#50;
            ToErrorDescrip:='Final frame building is error end';
        END_IF;
```

Figure 5.3.2.12 – CBT_MoveAbsolute FB algorithm step 50 building the complete message (source: own).

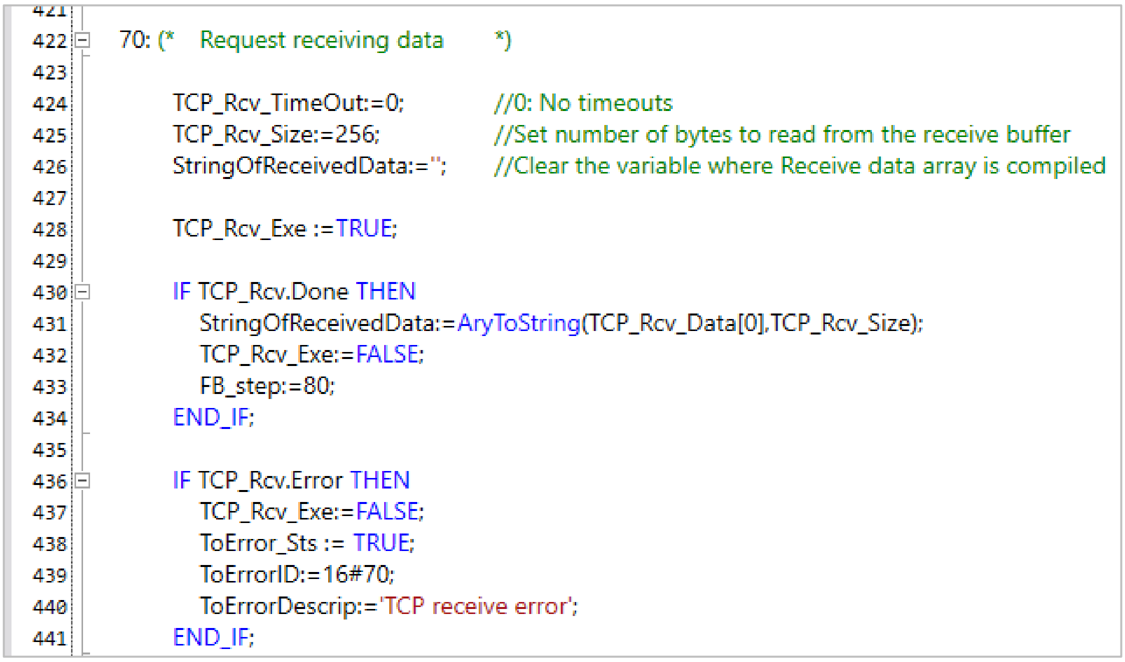
Step 60 creates an array of strings and sends the complete message to the robot.



```
60: (*   Send command      *)

        //Finds the number of characters in the string to be sent in the frame
        TCP_Send_Size:=LEN(str_SendFrame);
        ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH, AryOut:=TCP_Send_Data[0]);

        TCP_Send_Exe :=TRUE;

        IF TCP_Send.Done THEN
            CBT.WaitingReturn:=TRUE;     //Flag to set the robot in "busy" state to avoid other FB to be executed
            TCP_Send_Exe:=FALSE;
            FB_step:=70;
        END_IF;

        IF TCP_Send.Error THEN
            TCP_Send_Exe:=FALSE;
            ToError_Sts := TRUE;
            ToErrorID:=16#60;
            ToErrorDescrip:='TCP send error';
        END_IF;
```

Figure 5.3.2.13 – CBT_MoveAbsolute FB algorithm step 60 sending the complete message (source: own).

Step 70 reads the data from the receive buffer for the TCP socket on the built-in EtherNet/IP on the PLC.



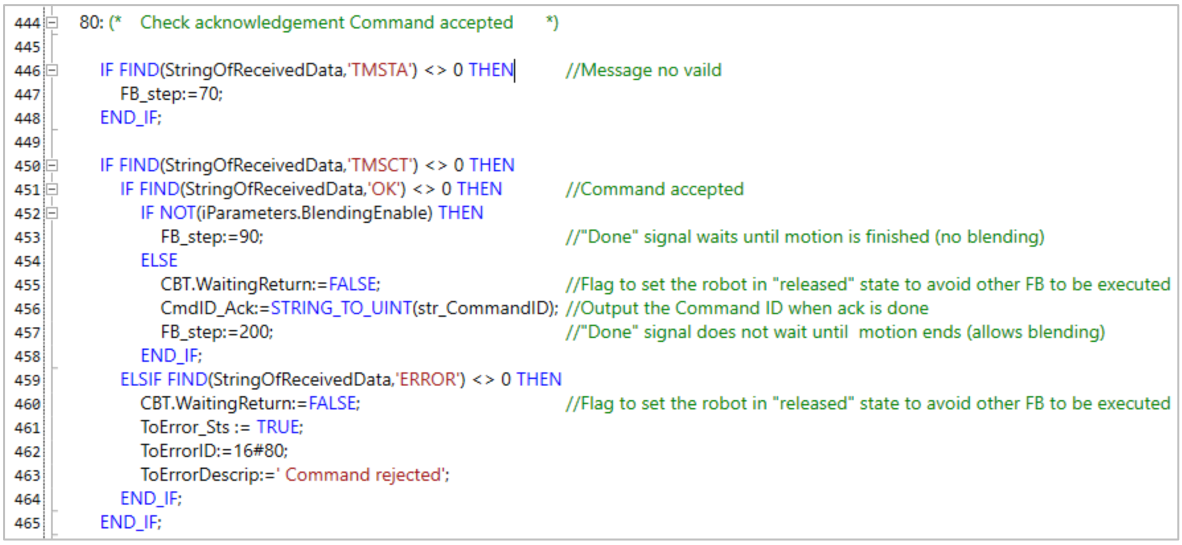
```
70: (*   Request receiving data        *)

        TCP_Rcv_TimeOut:=0;           //0: No timeouts
        TCP_Rcv_Size:=256;            //Set number of bytes to read from the receive buffer
        StringOfReceivedData:='';     //Clear the variable where Receive data array is compiled

        TCP_Rcv_Exe :=TRUE;

        IF TCP_Rcv.Done THEN
            StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
            TCP_Rcv_Exe:=FALSE;
            FB_step:=80;
        END_IF;

        IF TCP_Rcv.Error THEN
            TCP_Rcv_Exe:=FALSE;
            ToError_Sts := TRUE;
            ToErrorID:=16#70;
            ToErrorDescrip:='TCP receive error';
        END_IF;
```

Figure 5.3.2.14 – CBT_MoveAbsolute FB algorithm step 70 receiving TCP message from robot (source: own).

Step 80 checks the data received in the TCP socket. If the message includes the word “OK”, the command has been accepted by the robot.

If blending is disabled, *Done* output will remain FALSE until the motion command has been completed and acknowledged by the robot. FB sequence then jumps to step 90.

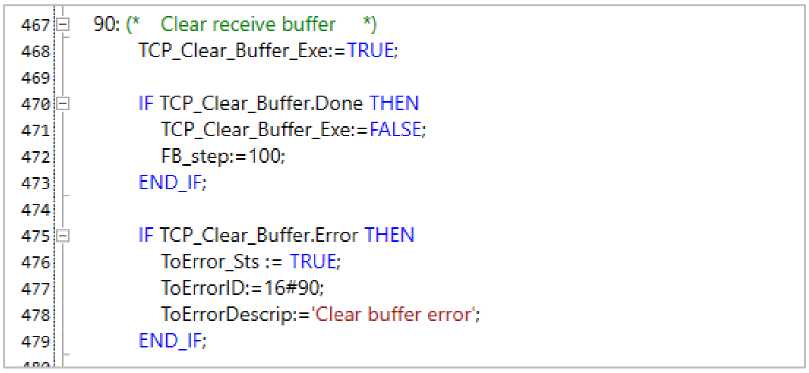
If blending is enabled, *Done* output is TRUE at this moment (does not wait until motion completed). FB sequence then jumps to step 200.



```
80: (*   Check acknowledgement Command accepted      *)

  IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN          //Message no vaild
    FB_step:=70;
  END_IF;

  IF FIND(StringOfReceivedData,'TMSCT') <> 0 THEN
    IF FIND(StringOfReceivedData,'OK') <> 0 THEN           //Command accepted
      IF NOT(iParameters.BlendingEnable) THEN
        FB_step:=90;                                       //"Done" signal waits until motion is finished (no blending)
      ELSE
        CBT.WaitingReturn:=FALSE;                          //Flag to set the robot in "released" state to avoid other FB to be executed
        CmdID_Ack:=STRING_TO_UINT(str_CommandID);          //Output the Command ID when ack is done
        FB_step:=200;                                      //"Done" signal does not wait until  motion ends (allows blending)
      END_IF;
    ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
      CBT.WaitingReturn:=FALSE;                            //Flag to set the robot in "released" state to avoid other FB to be executed
      ToError_Sts := TRUE;
      ToErrorID:=16#80;
      ToErrorDescrip:=' Command rejected';
    END_IF;
  END_IF;
```

Figure 5.3.2.15 – CBT_MoveAbsolute FB algorithm step 80 checking what information has been received (source: own).

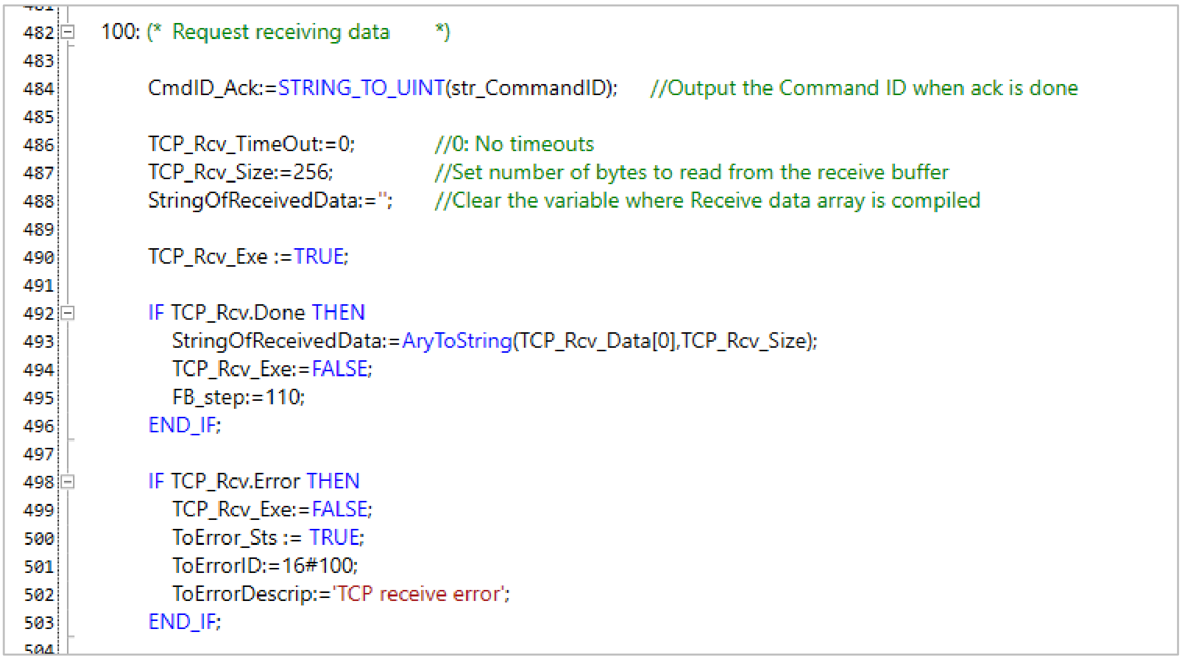
Step 90 clears the buffer of received data because Blending is disabled, and the FB is waiting for return message from robot as acknowledge of motion completed.



```
90: (*   Clear receive buffer     *)
    TCP_Clear_Buffer_Exe:=TRUE;

    IF TCP_Clear_Buffer.Done THEN
      TCP_Clear_Buffer_Exe:=FALSE;
      FB_step:=100;
    END_IF;

    IF TCP_Clear_Buffer.Error THEN
      ToError_Sts := TRUE;
      ToErrorID:=16#90;
      ToErrorDescrip:='Clear buffer error';
    END_IF;
```

Figure 5.3.2.16 – CBT_MoveAbsolute FB algorithm step 90 clearing the buffer of received data (source: own).

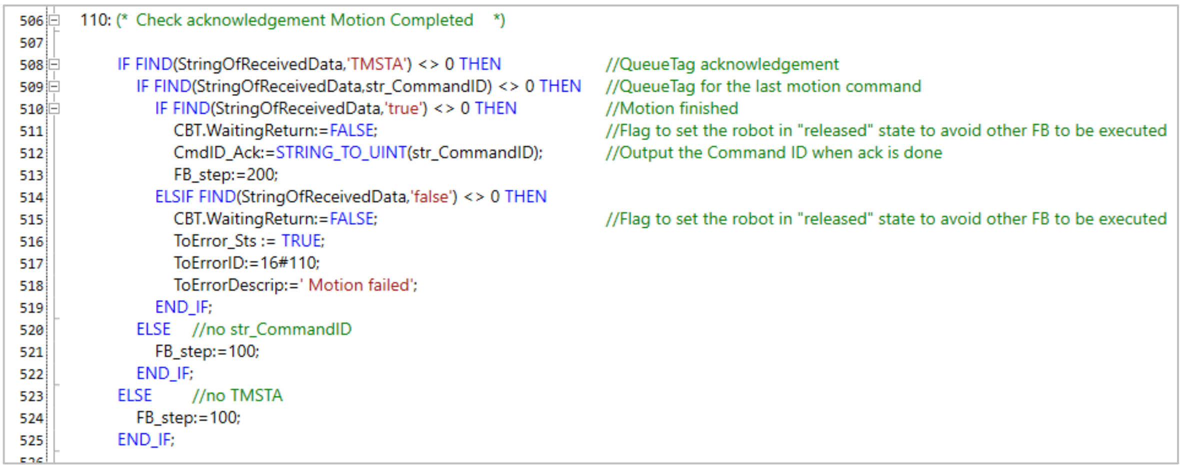
Step 100 reads the data from the receive buffer for the TCP socket on the built-in EtherNet/IP on the PLC.



```
100: (*  Request receiving data       *)

        CmdID_Ack:=STRING_TO_UINT(str_CommandID);     //Output the Command ID when ack is done

        TCP_Rcv_TimeOut:=0;             //0: No timeouts
        TCP_Rcv_Size:=256;              //Set number of bytes to read from the receive buffer
        StringOfReceivedData:='';       //Clear the variable where Receive data array is compiled

        TCP_Rcv_Exe :=TRUE;

        IF TCP_Rcv.Done THEN
           StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
           TCP_Rcv_Exe:=FALSE;
           FB_step:=110;
        END_IF;

        IF TCP_Rcv.Error THEN
           TCP_Rcv_Exe:=FALSE;
           ToError_Sts := TRUE;
           ToErrorID:=16#100;
           ToErrorDescrip:='TCP receive error';
        END_IF;
```

Figure 5.3.2.17 – CBT_MoveAbsolute FB algorithm step 100 receiving TCP message from robot (source: own).

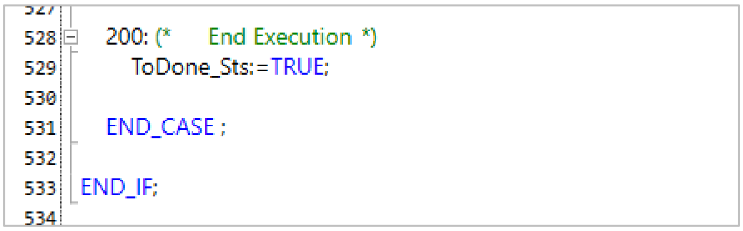
Step 110 verifies the data received in the TCP socket. If the message includes *TMSTA,* it means this message contains acknowledge information of the motion command identified with *str_CommandID*. If message contains the word “true”, the motion has been completed. Otherwise, and error occurred.



```
110: (*  Check acknowledgement Motion Completed    *)

        IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN                  //QueueTag acknowledgement
           IF FIND(StringOfReceivedData,str_CommandID) <> 0 THEN        //QueueTag for the last motion command
              IF FIND(StringOfReceivedData,'true') <> 0 THEN            //Motion finished
                 CBT.WaitingReturn:=FALSE;                              //Flag to set the robot in "released" state to avoid other FB to be executed
                 CmdID_Ack:=STRING_TO_UINT(str_CommandID);              //Output the Command ID when ack is done
                 FB_step:=200;
              ELSIF FIND(StringOfReceivedData,'false') <> 0 THEN
                 CBT.WaitingReturn:=FALSE;                              //Flag to set the robot in "released" state to avoid other FB to be executed
                 ToError_Sts := TRUE;
                 ToErrorID:=16#110;
                 ToErrorDescrip:=' Motion failed';
              END_IF;
           ELSE    //no str_CommandID
              FB_step:=100;
           END_IF;
        ELSE       //no TMSTA
           FB_step:=100;
        END_IF;
```

Figure 5.3.2.18 – CBT_MoveAbsolute FB algorithm step 110 verifying data received and acknowledge (source: own).

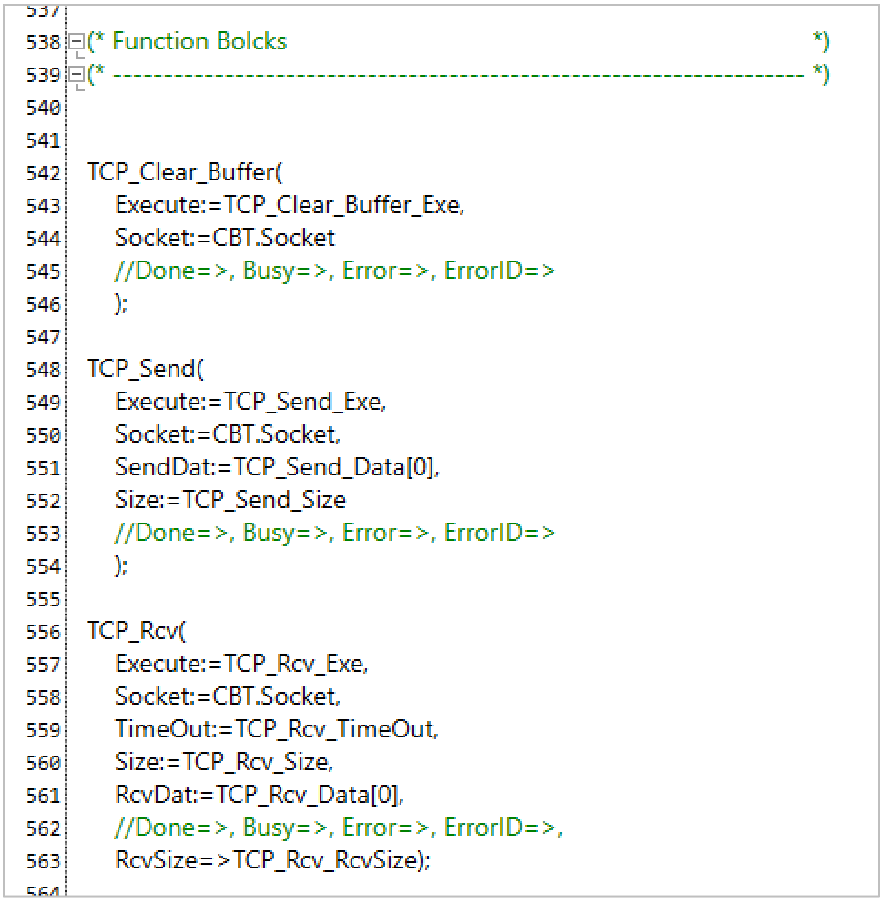
Step 200 finishes the execution of the Function Block.



```
    200: (*     End Execution  *)
       ToDone_Sts:=TRUE;

    END_CASE ;

END_IF;
```

Figure 5.3.2.19 – CBT_MoveAbsolute FB algorithm step 200 finishing the execution (source: own).

The last part of the script is reserved for TCP related FBs included be default in Sysmac Studio.



```
(* Function Bolcks                                                    *)
(* ------------------------------------------------------------------ *)


TCP_Clear_Buffer(
   Execute:=TCP_Clear_Buffer_Exe,
   Socket:=CBT.Socket
   //Done=>, Busy=>, Error=>, ErrorID=>
   );

TCP_Send(
   Execute:=TCP_Send_Exe,
   Socket:=CBT.Socket,
   SendDat:=TCP_Send_Data[0],
   Size:=TCP_Send_Size
   //Done=>, Busy=>, Error=>, ErrorID=>
   );

TCP_Rcv(
   Execute:=TCP_Rcv_Exe,
   Socket:=CBT.Socket,
   TimeOut:=TCP_Rcv_TimeOut,
   Size:=TCP_Rcv_Size,
   RcvDat:=TCP_Rcv_Data[0],
   //Done=>, Busy=>, Error=>, ErrorID=>,
   RcvSize=>TCP_Rcv_RcvSize);
```

Figure 5.3.2.20 – CBT_MoveAbsolute FB internal functions instances for TCP communications (source: own).

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
10	Robot not connected	Use CBT_Connect FB for connection
11	Other FB is under execution	Wait until other FB is done
20	Clear buffer error	Check Ethernet connection wiring
25	Speed range in Line: 0~4500 mm/s	Set parameter in acceptable range
26	Speed range in PTP: 0~100 %	Set parameter in acceptable range
27	Acceleration Time range is: 150~9999 ms	Set parameter in acceptable range
28	Command ID range is: 2~9	Set parameter in acceptable range
29	Robot Pose must be disabled for Line command	Set parameter in acceptable range
30	Pose1 range is 0~1	Set parameter in acceptable range
31	Pose2 range is 2~3	Set parameter in acceptable range
32	Pose3 range is 4~5	Set parameter in acceptable range
35	DataFormat invalid for PTP command	Set parameter in acceptable range
36	DataFormat invalid for Line command	Set parameter in acceptable range
40	Checksum length not valid	Re-execution is required
50	Final frame building is error end	Re-execution is required
60	TCP send error	Check Ethernet connection wiring
70	TCP receive error	Check Ethernet connection wiring
80	Command rejected	Re-execution is required
90	Clear buffer error	Check Ethernet connection wiring
100	TCP receive error	Check Ethernet connection wiring
110	Motion failed	Re-execution is required

Table 5.3.2.8 – CBT_MoveAbsolute error list description and action (source: own).

5.3.3. CBT_MoveRelative

This Function Block sends the command to move the robot to a relative target position. Two different types of motion can be defined:

- PTP: Robot moves to the target point along the closest path of the joint angle space.
- Line: Tool moves to the target point in a straight line.

Among other parameters, user must set the Cobot In/out variable to identify the target robot with its IP address and communications port, the command ID to identify the command in the feedback sent by the robot, speed including units, acceleration time, if blending is required and if it is performed by percentage or by radius.

Unlike absolute movements with CBT_MoveAbsolute, arm configuration is not available for relative movements, in this case the arm configuration is kept as it was at the moment the motion started.

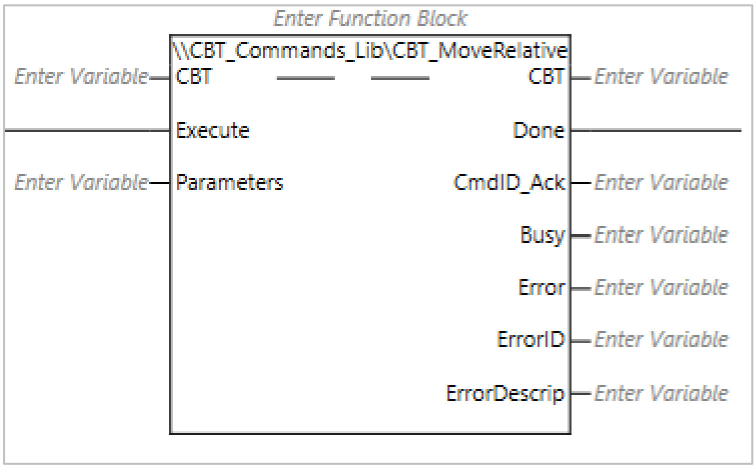


Figure 5.3.3.1 – CBT_MoveRelative Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In/Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
Parameters	Input	CBT_Commands_Lib\stCBT_MovRelParam	---	---	Motion parameters
Done	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is completed
CmdID_Ack	Output	UINT	0 to +65535	0	Command identification number
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.3.1 – CBT_MoveRelative variables type, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.3.2 – stCBT_Connection datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
CommandID	UINT	2 to 9	0	Identification for the return message for this command
MovementCommand	CBT_Commands_Lib\ eCBT_MovCmd_Movement	---	---	Motion type
DataFormat	CBT_Commands_Lib\ eCBT_MovCmdRel_DataFormat	---	---	Motion strategy
TargetPosition	CBT_Commands_Lib\ stCBT_Transformation	---	---	Target position
Speed	UINT	0 to 4500	0	Speed expressed as a percentage (%) or in velocity (mm/s)
AccelTime	UINT	150 to 9999	0	Time interval to accelerate to top speed (ms)
BlendingEnable	BOOL	TRUE or FALSE	FALSE	Enables blending with next motion command
BlendingValue	UINT	0 to 100	0	Blending value expressed as a percentage (%) or in radius (mm)
PrecisePositioning	BOOL	TRUE or FALSE	FALSE	Whether robot moves to the point precisely
ExitNode	BOOL	TRUE or FALSE	FALSE	Quit Listen node in robot program

Table 5.3.3.3 – stCBT_MovRelParam datatype, range, default value and description (source: own).

Name	Enum Value	Description
Line	0	Motion in Joint, speed in %, blending in %
PTP	1	Motion in Cartesian, speed in %, blending in %

Table 5.3.3.4 – eCBT_MovCmd_Movement datatype, value and description (source: own).

Name	Enum Value	Description
JPP_Rel	0	Motion in Joint, speed in %, blending in %
CPP_Rel	1	Motion in Cartesian, speed in %, blending in %
CPR_Rel	2	Motion in Cartesian, speed in %, blending in radius
CAP_Rel	3	Motion in Cartesian, speed in mm/s, blending in %
CAR_Rel	4	Motion in Cartesian, speed in mm/s, blending in radius
TPP_Rel	5	Motion in Tool, speed in %, blending in %
TPR_Rel	6	Motion in Tool, speed in %, blending in radius
TAP_Rel	7	Motion in Tool, speed in mm/s, blending in %
TAR_Rel	8	Motion in Tool, speed in mm/s, blending in radius

Table 5.3.3.5 – eCBT_MovCmdRel_DataFormat datatype, value and description (source: own).

Name	Data Type	Valid Range	Default	Description
X	REAL	---	0	X coordinates in mm
Y	REAL	---	0	Y coordinates in mm
Z	REAL	---	0	Z coordinates in mm
RX	REAL	---	0	RX coordinates in degrees
RY	REAL	---	0	RY coordinates in degrees
RZ	REAL	---	0	RZ coordinates in degrees

Table 5.3.3.6 – stCBT_Transformation datatype, range, default value and description (source: own).

Function Block Script

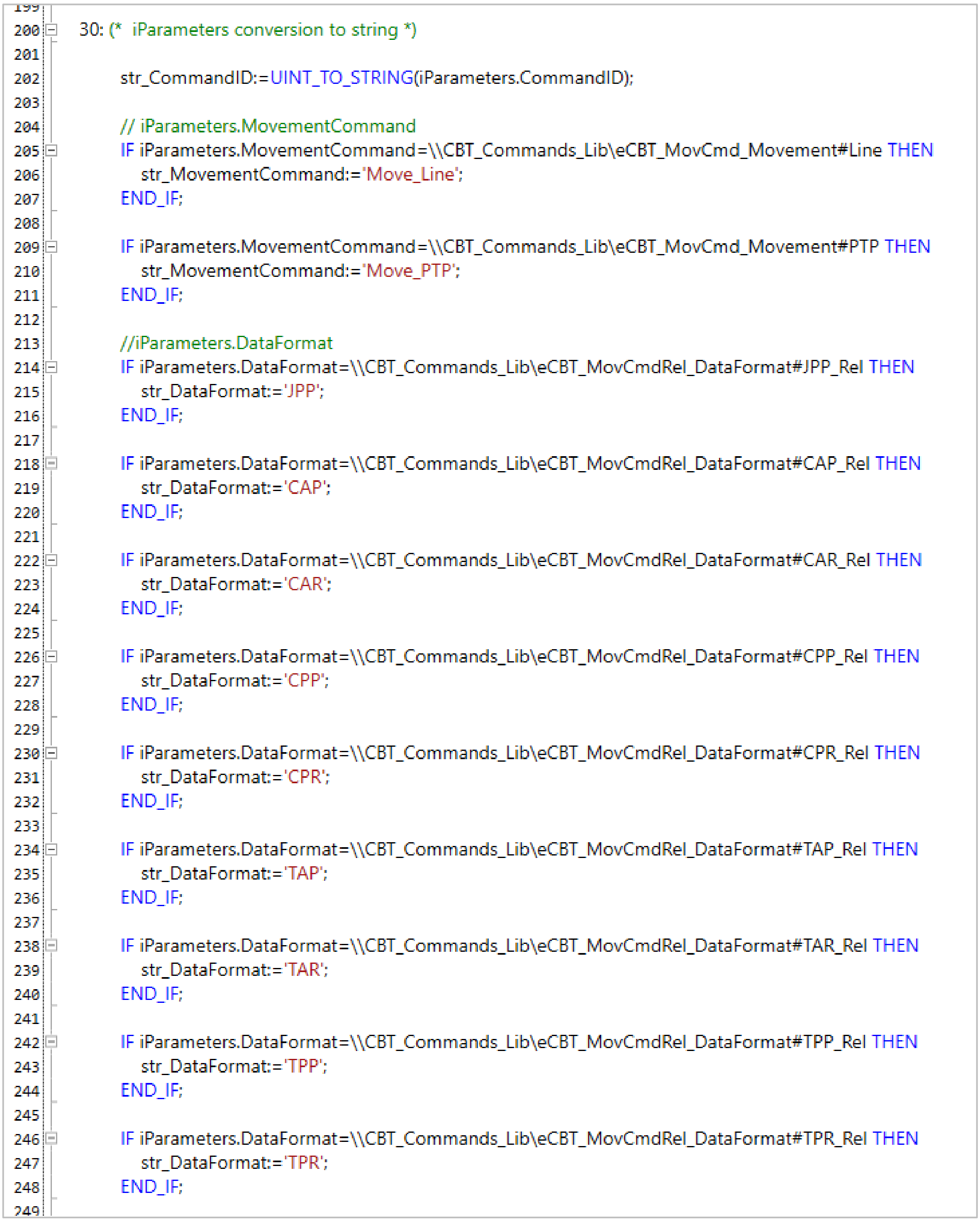
As all Function Blocks in this Library, once the upward signal differentiation of *Execute* input is detected and it is not in *Busy* state, the sequence initialization is started, otherwise an error is triggered. Sequence initialization has been described in *5.3.1.CBT_Connect* section.

As all Function Blocks in this Library, State Diagram Control is part of the script focused on output variable operation and timing. State Diagram Control has been described in *5.3.1.CBT_Connect* section.

From step 0 to 30 and from step 60 to the end, refer to the section *5.3.2. CBT_MoveAbsolute*.

Hereunder, the description of the specific script of CBT_MoveRelative FB:

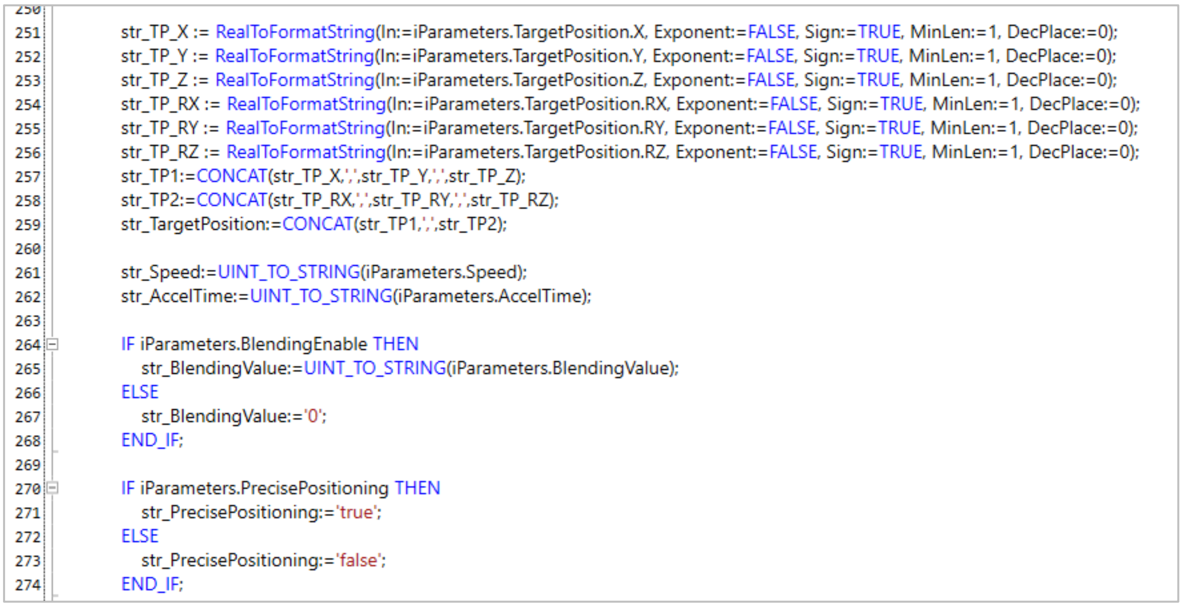
Step 30 creates the corresponding string variable (str_MovementCommand) depending on the movement command parameter and the string variable (str_DataFormat) for the data format parameter.



```
30: (*  iParameters conversion to string *)

        str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

        // iParameters.MovementCommand
        IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line THEN
            str_MovementCommand:='Move_Line';
        END_IF;

        IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP THEN
            str_MovementCommand:='Move_PTP';
        END_IF;

        //iParameters.DataFormat
        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#JPP_Rel THEN
            str_DataFormat:='JPP';
        END_IF;

        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CAP_Rel THEN
            str_DataFormat:='CAP';
        END_IF;

        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CAR_Rel THEN
            str_DataFormat:='CAR';
        END_IF;

        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CPP_Rel THEN
            str_DataFormat:='CPP';
        END_IF;

        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CPR_Rel THEN
            str_DataFormat:='CPR';
        END_IF;

        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TAP_Rel THEN
            str_DataFormat:='TAP';
        END_IF;

        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TAR_Rel THEN
            str_DataFormat:='TAR';
        END_IF;

        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TPP_Rel THEN
            str_DataFormat:='TPP';
        END_IF;

        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TPR_Rel THEN
            str_DataFormat:='TPR';
        END_IF;
```

Figure 5.3.3.2 – CBT_MoveRelative FB algorithm step 30 creating the string of the data format parameter (source: own).

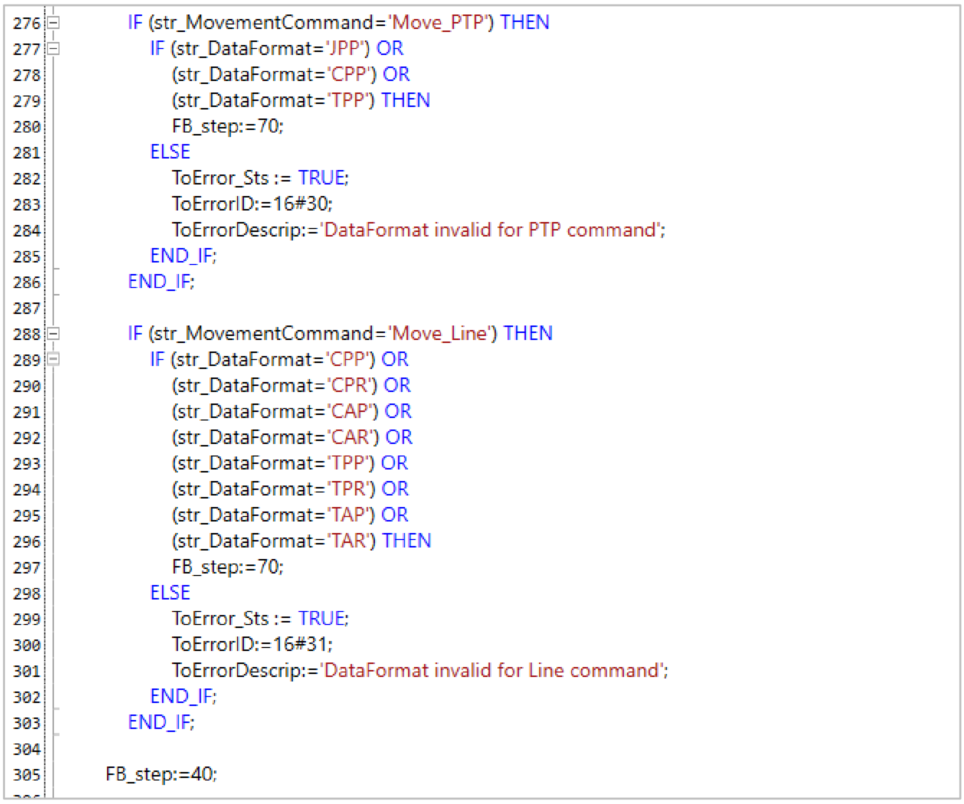
It converts target position parameter from real to string (str_TargetPosition), speed parameter to string variable (str_Speed), acceleration time to string (str_AccelTime), blending value to string (str_BlendingValue) and precise positioning to string (str_PrecisePositioning).



```
	str_TP_X := RealToFormatString(In:=iParameters.TargetPosition.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
	str_TP_Y := RealToFormatString(In:=iParameters.TargetPosition.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
	str_TP_Z := RealToFormatString(In:=iParameters.TargetPosition.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
	str_TP_RX := RealToFormatString(In:=iParameters.TargetPosition.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
	str_TP_RY := RealToFormatString(In:=iParameters.TargetPosition.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
	str_TP_RZ := RealToFormatString(In:=iParameters.TargetPosition.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
	str_TP1:=CONCAT(str_TP_X,',',str_TP_Y,',',str_TP_Z);
	str_TP2:=CONCAT(str_TP_RX,',',str_TP_RY,',',str_TP_RZ);
	str_TargetPosition:=CONCAT(str_TP1,',',str_TP2);

	str_Speed:=UINT_TO_STRING(iParameters.Speed);
	str_AccelTime:=UINT_TO_STRING(iParameters.AccelTime);

	IF iParameters.BlendingEnable THEN
		str_BlendingValue:=UINT_TO_STRING(iParameters.BlendingValue);
	ELSE
		str_BlendingValue:='0';
	END_IF;

	IF iParameters.PrecisePositioning THEN
		str_PrecisePositioning:='true';
	ELSE
		str_PrecisePositioning:='false';
	END_IF;
```

Figure 5.3.3.3 – CBT_MoveRelative FB algorithm step 30 creating the string of the target position, speed value, acceleration time value, blending value and precise positioning value (source: own).

Verifies if the data format selected is acceptable for the selected movement command.



```
		IF (str_MovementCommand='Move_PTP') THEN
			IF (str_DataFormat='JPP') OR
				(str_DataFormat='CPP') OR
				(str_DataFormat='TPP') THEN
				FB_step:=70;
			ELSE
				ToError_Sts := TRUE;
				ToErrorID:=16#30;
				ToErrorDescrip:='DataFormat invalid for PTP command';
			END_IF;
		END_IF;

		IF (str_MovementCommand='Move_Line') THEN
			IF (str_DataFormat='CPP') OR
				(str_DataFormat='CPR') OR
				(str_DataFormat='CAP') OR
				(str_DataFormat='CAR') OR
				(str_DataFormat='TPP') OR
				(str_DataFormat='TPR') OR
				(str_DataFormat='TAP') OR
				(str_DataFormat='TAR') THEN
				FB_step:=70;
			ELSE
				ToError_Sts := TRUE;
				ToErrorID:=16#31;
				ToErrorDescrip:='DataFormat invalid for Line command';
			END_IF;
		END_IF;

	FB_step:=40;
```

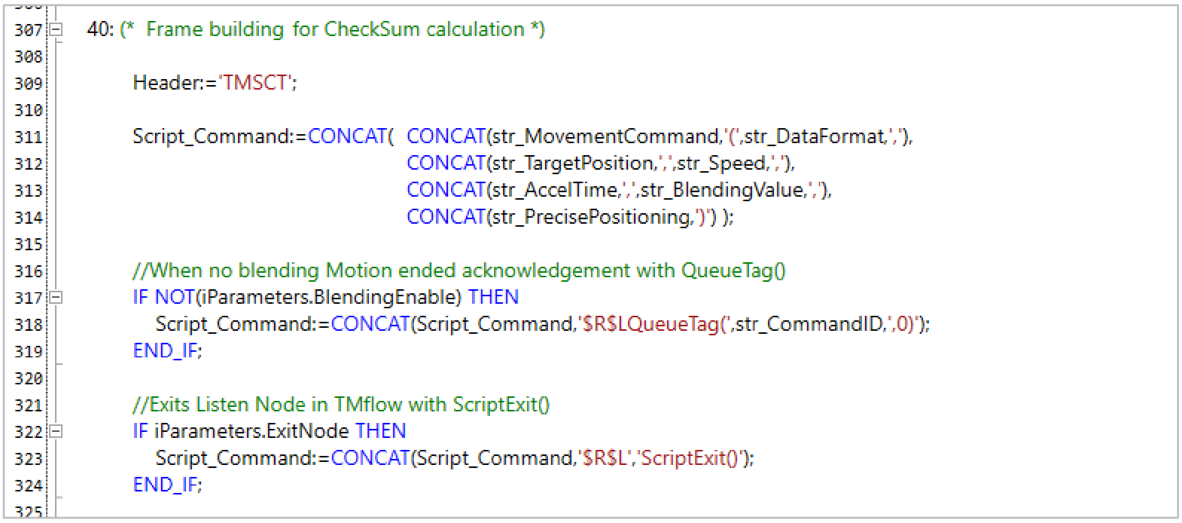
Figure 5.3.3.4 – CBT_MoveRelative FB algorithm step 30 checking if there is some incorrect data format parameter depending on the movement command (source: own).

Step 40 builds the message (script_command) to be sent to the robot including:

- Movement command
- Data format
- Target position
- Speed
- Acceleration time
- Blending value
- Precise positioning

➢ If blending is disabled, QueueTag() command is included in the message. It identifies the robot motion with the CommandID number for the acknowledge when robot motion finishes the current robot motion in process. Hence, *Done* output changes to TRUE when the robot movement has been completed.

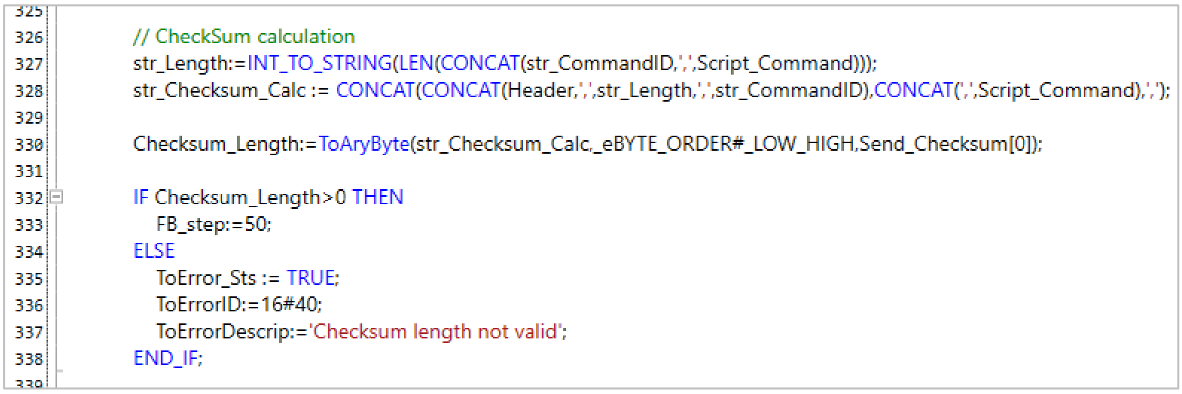
➢ If ExitNode is enabled, ScriptExit() command is included in the message. It exits external script control mode once the last motion command has been executed and finished. Listen Node is quitted, and not further external script functions commands are accepted.



```
40: (*  Frame building  for CheckSum calculation *)

        Header:='TMSCT';

        Script_Command:=CONCAT(  CONCAT(str_MovementCommand,'(',str_DataFormat,','),
                                 CONCAT(str_TargetPosition,',',str_Speed,','),
                                 CONCAT(str_AccelTime,',',str_BlendingValue,','),
                                 CONCAT(str_PrecisePositioning,')') );

        //When no blending Motion ended acknowledgement with QueueTag()
        IF NOT(iParameters.BlendingEnable) THEN
           Script_Command:=CONCAT(Script_Command,'$R$LQueueTag(',str_CommandID,',0)');
        END_IF;

        //Exits Listen Node in TMflow with ScriptExit()
        IF iParameters.ExitNode THEN
           Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
        END_IF;
```

Figure 5.3.3.5 – CBT_MoveRelative FB algorithm step 40 building the message with the complete command (source: own).

Once the message has been built, it is needed to find the number of characters of (str_CommandID + Script_Command) and to store it in the string (str_Length). Then, joining all parts (Header + str_Length + str_CommandID + Script_Command) to calculate the checksum (str_Checksum_Calc).

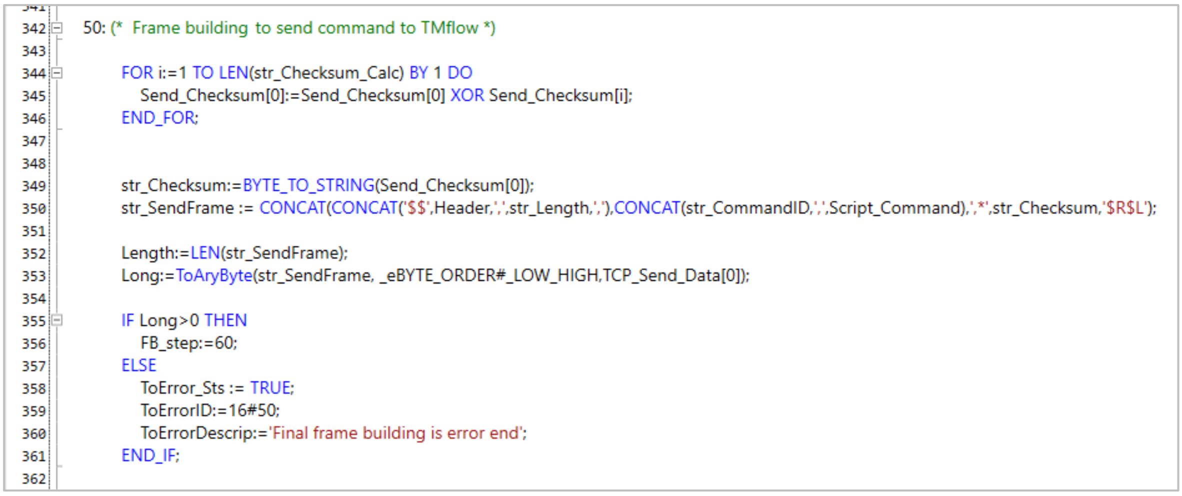


```
        // CheckSum calculation
        str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));
        str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');

        Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);

        IF Checksum_Length>0 THEN
           FB_step:=50;
        ELSE
           ToError_Sts := TRUE;
           ToErrorID:=16#40;
           ToErrorDescrip:='Checksum length not valid';
        END_IF;
```

Figure 5.3.3.6 – CBT_MoveRelative FB algorithm step 40 building the message with the complete command and finds the number of characters in the string (source: own).

Step 50 calculates, with XOR operation, the string for the checksum (str_Checksum) and created the complete message to be sent (str_SendFrame) composed by:

- Header
- Length of the message
- Command ID
- String command
- CheckSum calculation



```
    50: (*  Frame building  to send command to TMflow *)

        FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
           Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
        END_FOR;


        str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
        str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');

        Length:=LEN(str_SendFrame);
        Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

        IF Long>0 THEN
           FB_step:=60;
        ELSE
           ToError_Sts := TRUE;
           ToErrorID:=16#50;
           ToErrorDescrip:='Final frame building is error end';
        END_IF;
```

Figure 5.3.3.7 – CBT_MoveRelative FB algorithm step 50 builds the complete message (source: own).

From step 60 to the end of the algorithm is already described in previous FB explanation.

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
10	Robot not connected	Use CBT_Connect FB for connection
11	Other FB is under execution	Wait until other FB is done
20	Clear buffer error	Check Ethernet connection wiring
25	Speed range in Line: 0~4500 mm/s	Set parameter in acceptable range
26	Speed range in PTP: 0~100 %	Set parameter in acceptable range
27	Acceleration Time range is: 150~9999 ms	Set parameter in acceptable range
28	Command ID range is: 2~9	Set parameter in acceptable range
30	DataFormat invalid for PTP command	Set parameter in acceptable range
31	DataFormat invalid for Line command	Set parameter in acceptable range
40	Checksum length not valid	Re-execution is required
50	Final frame building is error end	Re-execution is required
60	TCP send error	Check Ethernet connection wiring
70	TCP receive error	Check Ethernet connection wiring
80	Command rejected	Re-execution is required
90	Clear buffer error	Check Ethernet connection wiring
100	TCP receive error	Check Ethernet connection wiring
110	Motion failed	Re-execution is required

Table 5.3.3.7 – CBT_MoveRelative error list description and action (source: own).

5.3.4. CBT_MoveCircle

This Function Block sends the command to move the robot describing a circular movement defined by the current position as initial position, an intermediate position, and an end position. The arc length must be also determined.

Among other parameters, user must set the Cobot In/out variable to identify the target robot with its IP address and communications port, the command ID to identify the command in the feedback sent by the robot, speed including units, acceleration time, if blending is required and if it is performed by percentage or by radius.

Unlike absolute movements with CBT_MoveAbsolute, arm configuration is not available for circular movements because, like in linear interpolation movements, the position of the TCP is constantly monitored, and it must follow the precalculated path. Hence, the arm configuration is kept as it was at the moment the motion started.

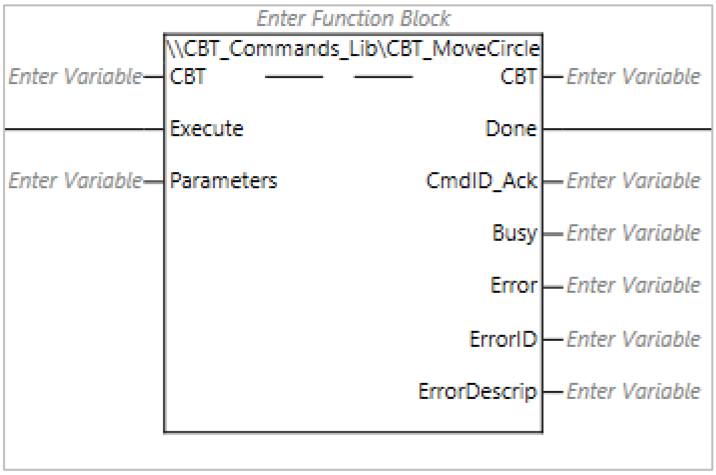


Figure 5.3.4.1 – CBT_MoveCircle Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In/Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
Parameters	Input	CBT_Commands_Lib\stCBT_MovCircleParam	---	---	Motion parameters
Done	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is completed
CmdID_Ack	Output	UINT	0 to +65535	0	Command identification number
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.4.1 – CBT_MoveCircle variables, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.4.2 – stCBT_Connection datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
CommandID	UINT	2 to 9	0	Identification for the return message for this command
DataFormat	CBT_Commands_Lib\ eCBT_MovCmdCircle_DataFormat	---	---	Motion strategy
ArcPoint	CBT_Commands_Lib\ stCBT_Transformation	---	---	A point on arc
EndPoint	CBT_Commands_Lib\ stCBT_Transformation	---	---	The end point of arc
Speed	UINT	0 to 4500	0	Speed expressed as a percentage (%) or in velocity (mm/s)
AccelTime	UINT	150 to 9999	0	Time interval to accelerate to top speed (ms)
BlendingEnable	BOOL	TRUE or FALSE	FALSE	Enables blending with next motion command
BlendingValue	UINT	0 to 100	0	Blending value expressed as a percentage (%) or in radius (mm)
ArcAngle	UINT	0 to 360	0	Length of the arc
PrecisePositioning	BOOL	TRUE or FALSE	FALSE	Whether robot moves to the point precisely
ExitNode	BOOL	TRUE or FALSE	FALSE	Quit Listen node in robot program

Table 5.3.4.3 – stCBT_MovCircleParam datatype, range, default value and description (source: own).

Name	Enum Value	Description
CPP_Circle	0	Motion in Cartesian, speed in %, blending in %
CAP_Circle	1	Motion in Cartesian, speed in mm/s, blending in %

Table 5.3.4.4 – eCBT_MovCmdCircle_DataFormat datatype, value and description (source: own).

Name	Data Type	Valid Range	Default	Description
X	REAL	---	0	X coordinates in mm
Y	REAL	---	0	Y coordinates in mm
Z	REAL	---	0	Z coordinates in mm
RX	REAL	---	0	RX coordinates in degrees
RY	REAL	---	0	RY coordinates in degrees
RZ	REAL	---	0	RZ coordinates in degrees

Table 5.3.4.5 – stCBT_Transformation datatype, range, default value and description (source: own).

Function Block Script

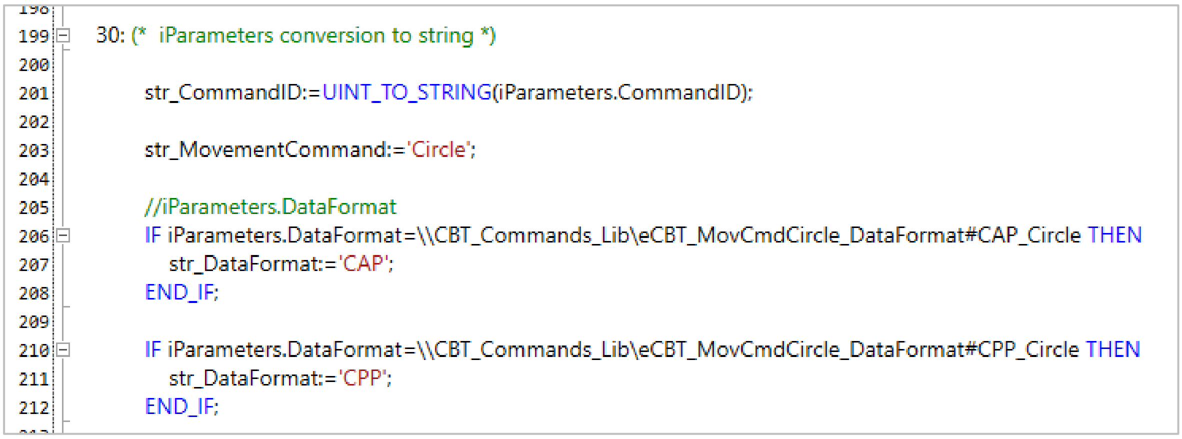
As all Function Blocks in this Library, once the upward signal differentiation of *Execute* input is detected and it is not in *Busy* state, the sequence initialization is started, otherwise an error is triggered. Sequence initialization has been described in *5.3.1.CBT_Connect* section.

As all Function Blocks in this Library, State Diagram Control is part of the script focused on output variable operation and timing. State Diagram Control has been described in *5.3.1.CBT_Connect* section.

From step 0 to 30 and from step 60 to the end, refer to the section *5.3.2. CBT_MoveAbsolute*.

Hereunder, the description of the specific script of CBT_MoveCircle FB:

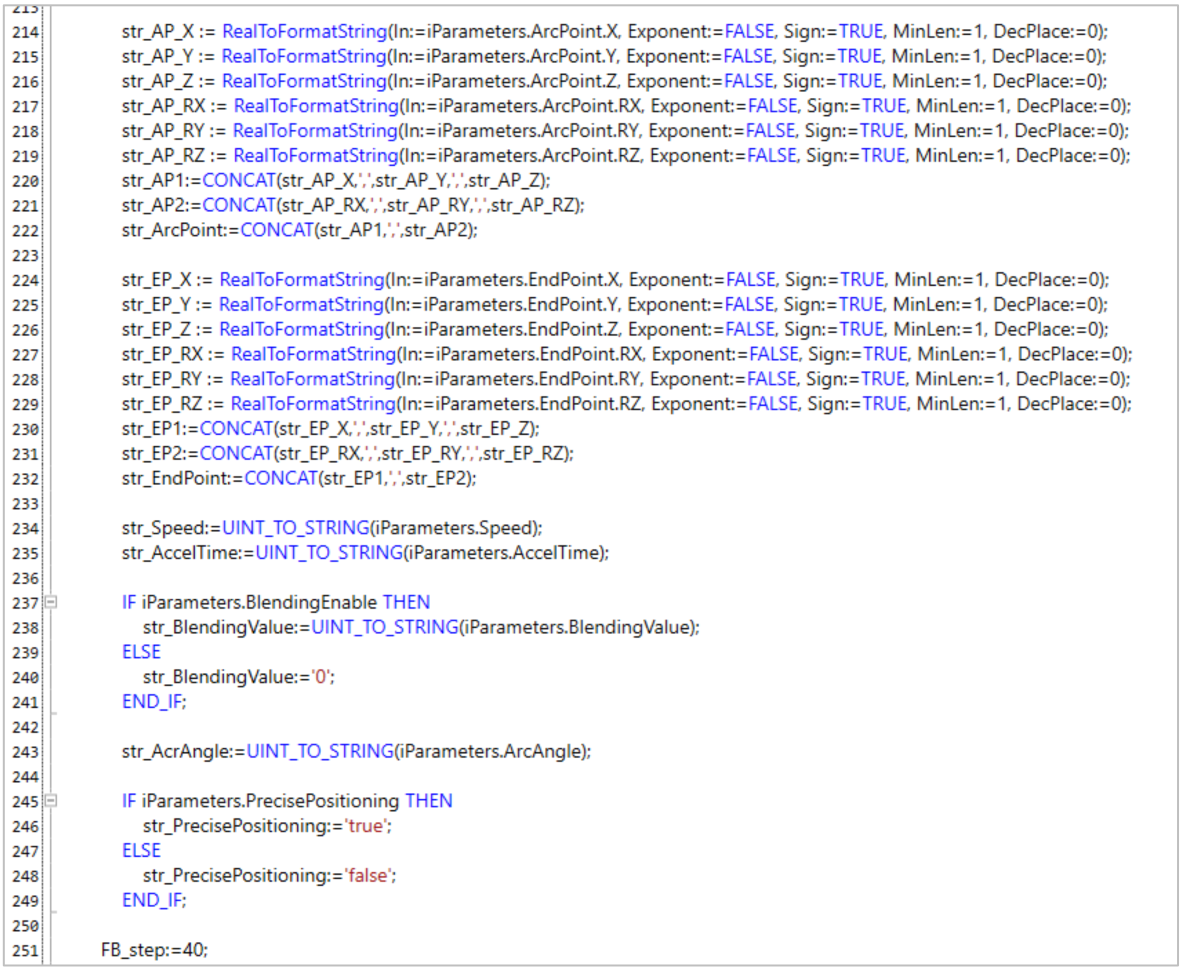
Step 30 creates the corresponding string variable (str_MovementCommand) and the string variable (str_DataFormat) for the data format parameter.



```
30: (*  iParameters conversion to string *)

        str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

        str_MovementCommand:='Circle';

        //iParameters.DataFormat
        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdCircle_DataFormat#CAP_Circle THEN
            str_DataFormat:='CAP';
        END_IF;

        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdCircle_DataFormat#CPP_Circle THEN
            str_DataFormat:='CPP';
        END_IF;
```

Figure 5.3.4.2 – CBT_MoveCircle FB algorithm step 30 creating the string of the data format parameter (source: own).

It converts arc position and end point parameters from real to string (str_ArcPosition and str_EndPoint), speed parameter to string variable (str_Speed), acceleration time to string (str_AccelTime), blending value to string (str_BlendingValue) and precise positioning to string (str_PrecisePositioning).



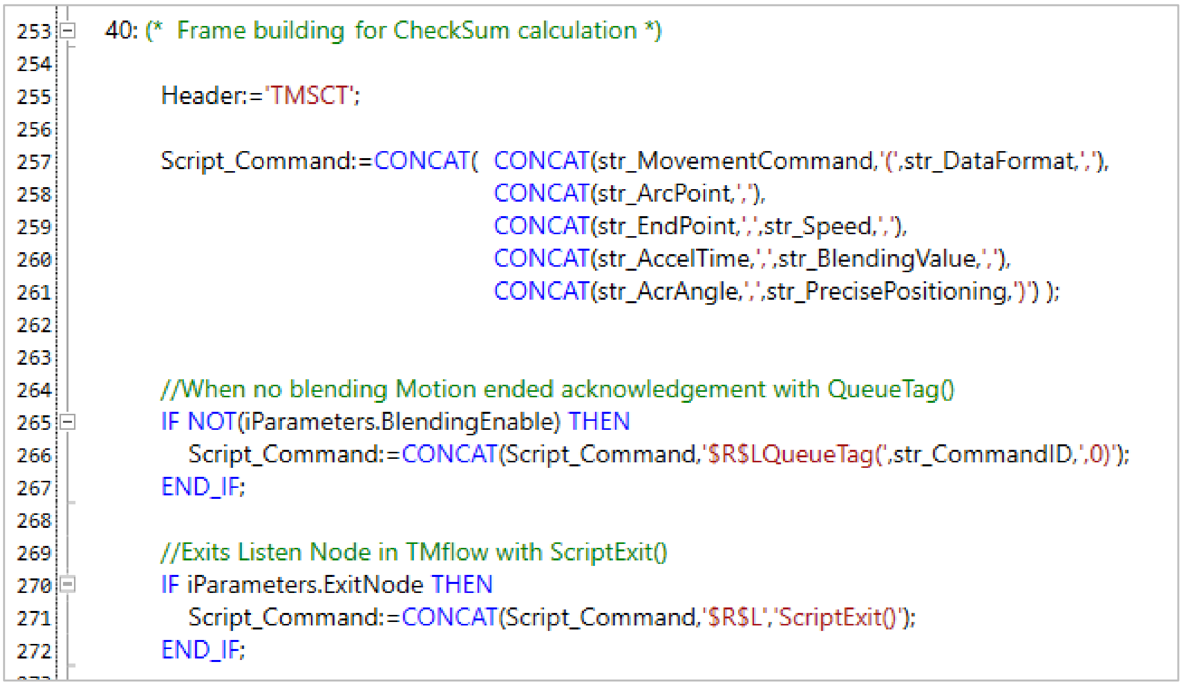
```
    str_AP_X := RealToFormatString(In:=iParameters.ArcPoint.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_AP_Y := RealToFormatString(In:=iParameters.ArcPoint.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_AP_Z := RealToFormatString(In:=iParameters.ArcPoint.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_AP_RX := RealToFormatString(In:=iParameters.ArcPoint.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_AP_RY := RealToFormatString(In:=iParameters.ArcPoint.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_AP_RZ := RealToFormatString(In:=iParameters.ArcPoint.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_AP1:=CONCAT(str_AP_X,',',str_AP_Y,',',str_AP_Z);
    str_AP2:=CONCAT(str_AP_RX,',',str_AP_RY,',',str_AP_RZ);
    str_ArcPoint:=CONCAT(str_AP1,',',str_AP2);

    str_EP_X := RealToFormatString(In:=iParameters.EndPoint.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_EP_Y := RealToFormatString(In:=iParameters.EndPoint.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_EP_Z := RealToFormatString(In:=iParameters.EndPoint.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_EP_RX := RealToFormatString(In:=iParameters.EndPoint.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_EP_RY := RealToFormatString(In:=iParameters.EndPoint.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_EP_RZ := RealToFormatString(In:=iParameters.EndPoint.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_EP1:=CONCAT(str_EP_X,',',str_EP_Y,',',str_EP_Z);
    str_EP2:=CONCAT(str_EP_RX,',',str_EP_RY,',',str_EP_RZ);
    str_EndPoint:=CONCAT(str_EP1,',',str_EP2);

    str_Speed:=UINT_TO_STRING(iParameters.Speed);
    str_AccelTime:=UINT_TO_STRING(iParameters.AccelTime);

    IF iParameters.BlendingEnable THEN
        str_BlendingValue:=UINT_TO_STRING(iParameters.BlendingValue);
    ELSE
        str_BlendingValue:='0';
    END_IF;

    str_AcrAngle:=UINT_TO_STRING(iParameters.ArcAngle);

    IF iParameters.PrecisePositioning THEN
        str_PrecisePositioning:='true';
    ELSE
        str_PrecisePositioning:='false';
    END_IF;

FB_step:=40;
```

Figure 5.3.4.3 – CBT_MoveCircle FB algorithm step 30 creating the string of the target position, speed value, acceleration time value, blending value and precise positioning value (source: own).

Step 40 builds the message (script_command) to be sent to the robot including:

- Movement command
- Data format
- Arc point
- End point
- Speed
- Acceleration time
- Blending value
- Precise positioning

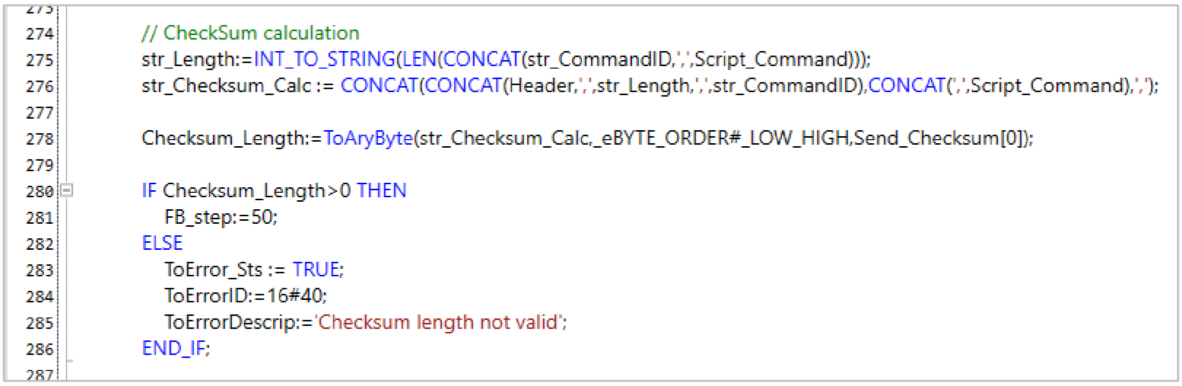
- If blending is disabled, QueueTag() command is included in the message. It identifies the robot motion with the CommandID number for the acknowlegde when robot motion finishes the current robot motion in process. Hence, *Done* output changes to TRUE when the robot movement has been completed.
- If ExitNode is enabled, ScriptExit() command is included in the message. It exits external script control mode once the last motion command has been executed and finished. Listen Node is quitted, and not further external script functions commands are accepted.



```
40: (*  Frame building  for CheckSum calculation *)

        Header:='TMSCT';

        Script_Command:=CONCAT(  CONCAT(str_MovementCommand,'(',str_DataFormat,','),
                                 CONCAT(str_ArcPoint,','),
                                 CONCAT(str_EndPoint,',',str_Speed,','),
                                 CONCAT(str_AccelTime,',',str_BlendingValue,','),
                                 CONCAT(str_AcrAngle,',',str_PrecisePositioning,')') );


        //When no blending Motion ended acknowledgement with QueueTag()
        IF NOT(iParameters.BlendingEnable) THEN
           Script_Command:=CONCAT(Script_Command,'$R$LQueueTag(',str_CommandID,',0)');
        END_IF;

        //Exits Listen Node in TMflow with ScriptExit()
        IF iParameters.ExitNode THEN
           Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
        END_IF;
```

Figure 5.3.4.4 – CBT_MoveCircle FB algorithm step 40 building the message with the complete command (source: own).

Once the message has been built, it is needed to find the number of characters of (str_CommandID + Script_Command) and to store it in the string (str_Length). Then, joining all parts (Header + str_Length + str_CommandID + Script_Command) to calculate the checksum (str_Checksum_Calc).

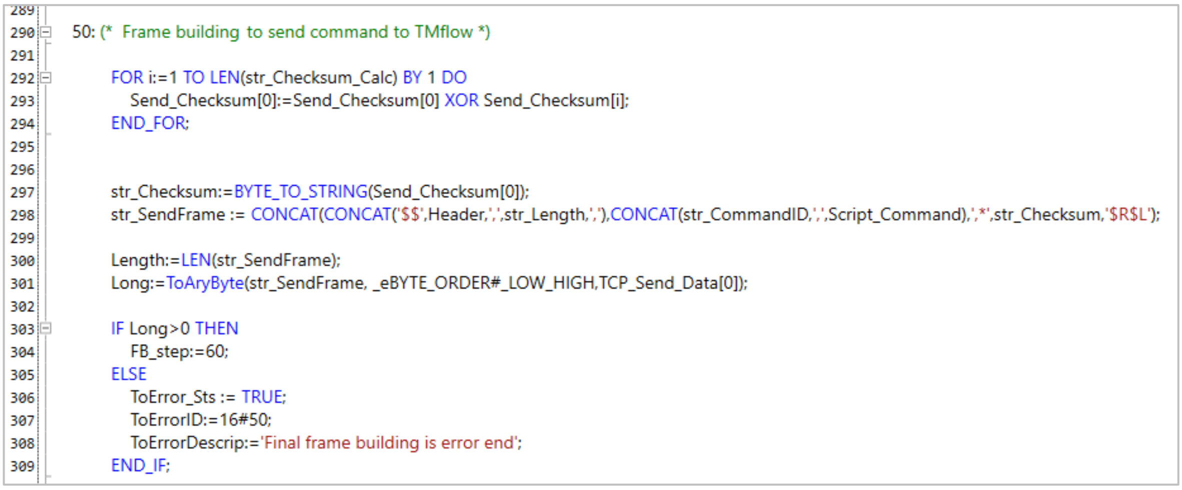


```
    // CheckSum calculation
    str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));
    str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');

    Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);

    IF Checksum_Length>0 THEN
      FB_step:=50;
    ELSE
      ToError_Sts := TRUE;
      ToErrorID:=16#40;
      ToErrorDescrip:='Checksum length not valid';
    END_IF;
```

Figure 5.3.4.5 – CBT_MoveCircle FB algorithm step 40 building the message with the complete command and finds the number of characters in the string (source: own).

Step 50 calculates with XOR operation the string for the checksum (str_Checksum) and created the complete message to be sent (str_SendFrame) composed by:

- Header
- Length of the message
- Command ID
- String command
- CheckSum calculation



```
50: (*  Frame building  to send command to TMflow *)

    FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
      Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
    END_FOR;


    str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
    str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');

    Length:=LEN(str_SendFrame);
    Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

    IF Long>0 THEN
      FB_step:=60;
    ELSE
      ToError_Sts := TRUE;
      ToErrorID:=16#50;
      ToErrorDescrip:='Final frame building is error end';
    END_IF;
```

Figure 5.3.4.6 – CBT_MoveCircle FB algorithm step 50 building the complete message (source: own).

From step 60 to the end of the algorithm, refer to the section *5.3.2. CBT_MoveAbsolute*.

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
10	Robot not connected	Use CBT_Connect FB for connection
11	Other FB is under execution	Wait until other FB is done
20	Clear buffer error	Check Ethernet connection wiring
25	Speed range in Line: 0~4500 mm/s	Set parameter in acceptable range
26	Speed range in PTP: 0~100 %	Set parameter in acceptable range
27	Acceleration Time range is: 150~9999 ms	Set parameter in acceptable range
28	Command ID range is: 2~9	Set parameter in acceptable range
40	Checksum length not valid	Re-execution is required
50	Final frame building is error end	Re-execution is required
60	TCP send error	Check Ethernet connection wiring
70	TCP receive error	Check Ethernet connection wiring
80	Command rejected	Re-execution is required
90	Clear buffer error	Check Ethernet connection wiring
100	TCP receive error	Check Ethernet connection wiring
110	Motion failed	Re-execution is required

Table 5.3.4.6 – CBT_MoveCircle error list description and action (source: own).

5.3.5. CBT_ChangeBase

This Function Block sends the command for changing the base coordinates of the follow-up motions into buffer for execution.

Among other parameters, user must set the Cobot In/out variable to identify the target robot with its IP address and communications port, the command ID to identify the command in the feedback sent by the robot and the base coordinates: X, Y and Z in millimetres for positioning, and RX, RY and RZ in degrees for orientation.

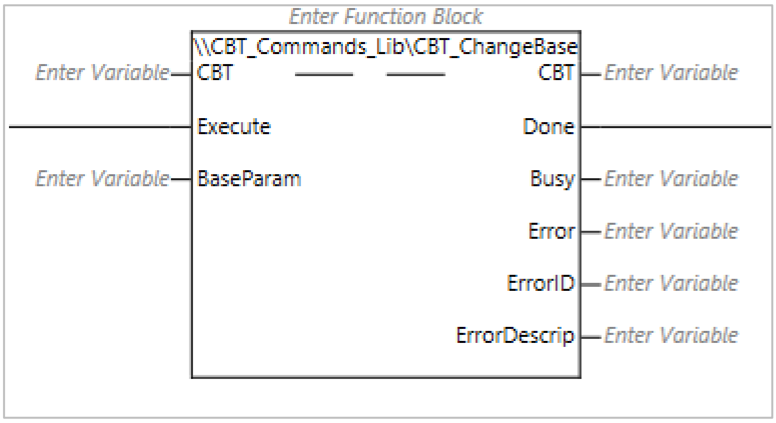


Figure 5.3.5.1 – CBT_ChangeBase Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In/Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
Parameters	Input	CBT_Commands_Lib\stCBT_BaseParam	---	---	Base Parameters
Done	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is completed
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.5.1 – CBT_ChangeBase variables type, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.5.2 – stCBT_Connection datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
CommandID	UINT	2 to 9	0	Identification for the return message for this command
Transformation	CBT_Commands_Lib\stCBT_Transformation	---	---	Base coordinates
ExitNode	BOOL	TRUE or FALSE	FALSE	Quit Listen node in robot program

Table 5.3.5.3 – stCBT_BaseParam datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
X	REAL	---	0	X coordinates in mm
Y	REAL	---	0	Y coordinates in mm
Z	REAL	---	0	Z coordinates in mm
RX	REAL	---	0	RX coordinates in degrees
RY	REAL	---	0	RY coordinates in degrees
RZ	REAL	---	0	RZ coordinates in degrees

Table 5.3.5.4 – stCBT_Transformation datatype, range, default value and description (source: own).

Function Block Script

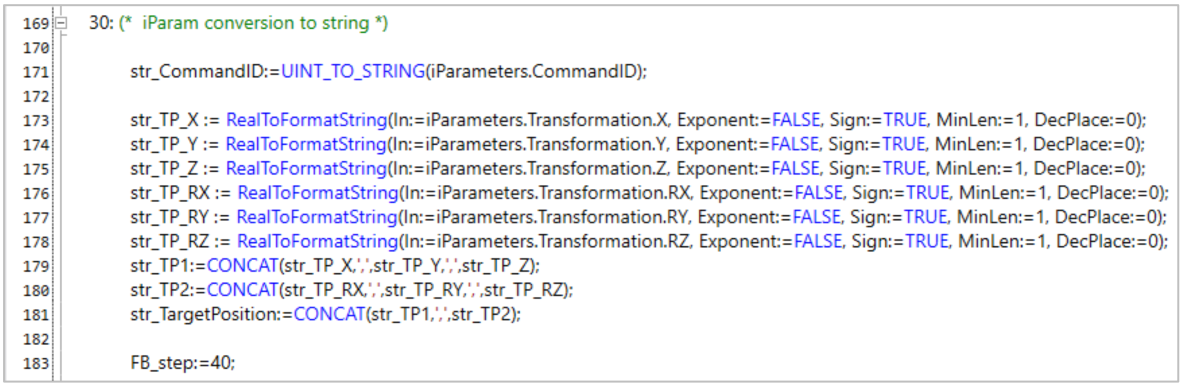
As all Function Blocks in this Library, once the upward signal differentiation of *Execute* input is detected and it is not in *Busy* state, the sequence initialization is started, otherwise an error is triggered. Sequence initialization has been described in *5.3.1.CBT_Connect* section.

As all Function Blocks in this Library, State Diagram Control is part of the script focused on output variable operation and timing. State Diagram Control has been described in *5.3.1.CBT_Connect* section.

From step 0 to 30 and from step 60 to the end, refer to the section *5.3.2. CBT_MoveAbsolute*.

Hereunder, the description of the specific script of CBT_ChangeBase FB:

Step 30 converts base transformation parameter from real to string (str_TargetPosition).



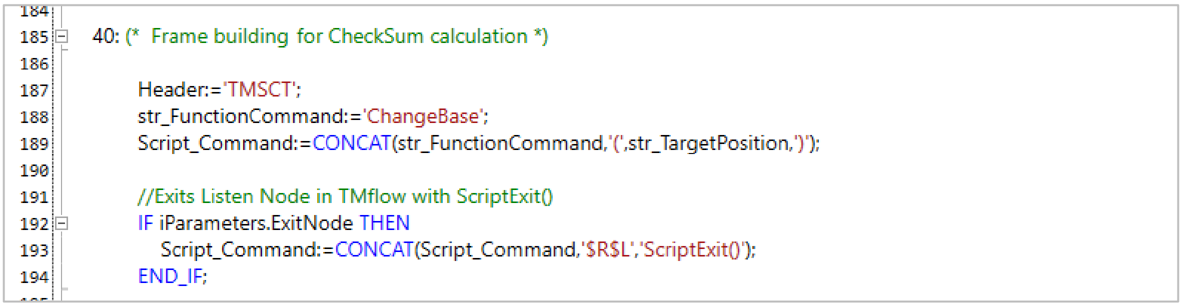
```
30: (*  iParam conversion to string *)

        str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

        str_TP_X := RealToFormatString(In:=iParameters.Transformation.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TP_Y := RealToFormatString(In:=iParameters.Transformation.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TP_Z := RealToFormatString(In:=iParameters.Transformation.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TP_RX := RealToFormatString(In:=iParameters.Transformation.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TP_RY := RealToFormatString(In:=iParameters.Transformation.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TP_RZ := RealToFormatString(In:=iParameters.Transformation.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TP1:=CONCAT(str_TP_X,',',str_TP_Y,',',str_TP_Z);
        str_TP2:=CONCAT(str_TP_RX,',',str_TP_RY,',',str_TP_RZ);
        str_TargetPosition:=CONCAT(str_TP1,',',str_TP2);

        FB_step:=40;
```

Figure 5.3.5.2 – CBT_ChangeBase FB algorithm step 30 creating the string of the base transformation (source: own).

Step 40 builds the message (script_command) to be sent to the robot including:

- Function command
- Base transformation

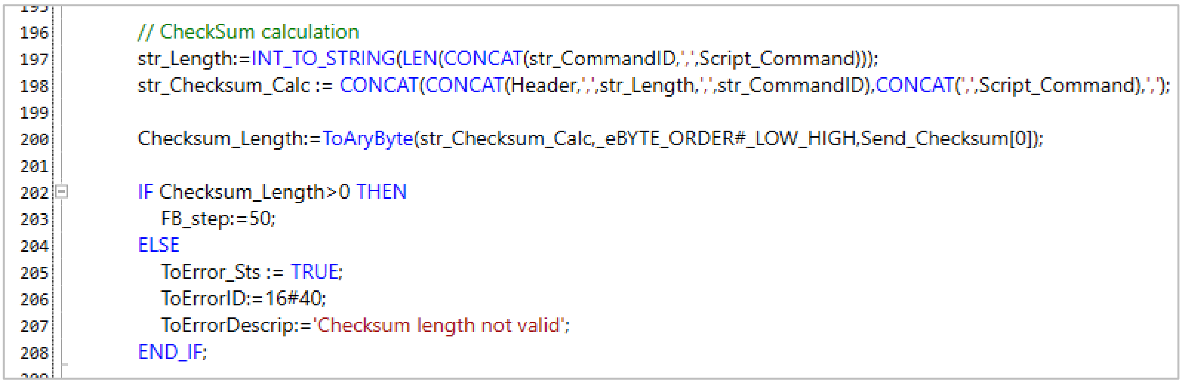
➢ If ExitNode is enabled, ScriptExit() command is included in the message. It exits external script control mode once the last motion command has been executed and finished. Listen Node is quitted, and not further external script functions commands are accepted.



```
40: (*  Frame building for CheckSum calculation *)

        Header:='TMSCT';
        str_FunctionCommand:='ChangeBase';
        Script_Command:=CONCAT(str_FunctionCommand,'(',str_TargetPosition,')');

        //Exits Listen Node in TMflow with ScriptExit()
        IF iParameters.ExitNode THEN
            Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
        END_IF;
```

Figure 5.3.5.3 – CBT_ChangeBase FB algorithm step 40 building the message with the complete command (source: own).

Once the message has been built, it is needed to find the number of characters of (str_CommandID + Script_Command) and to store it in the string (str_Length). Then, joining all parts (Header + str_Length + str_CommandID + Script_Command) to calculate the checksum (str_Checksum_Calc).

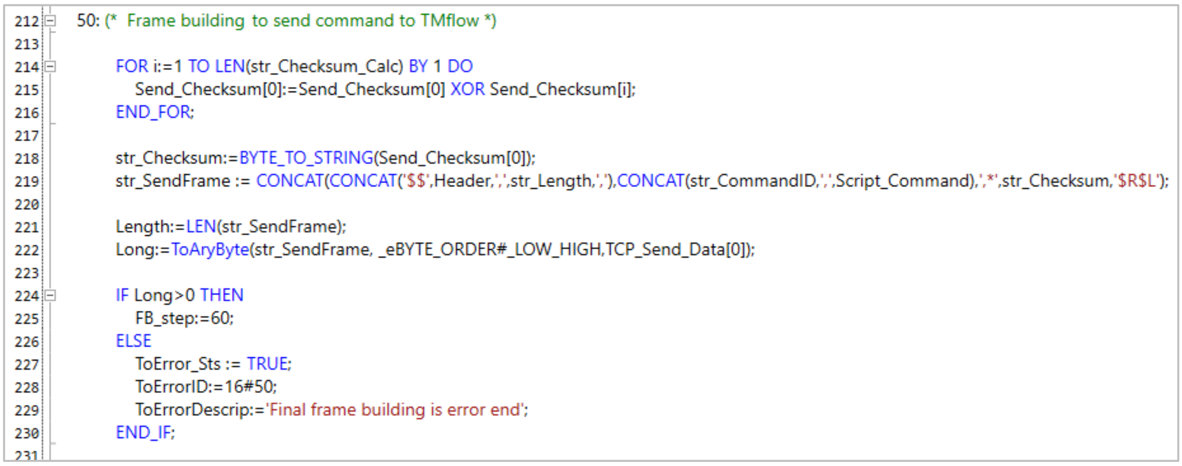


```
        // CheckSum calculation
        str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));
        str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');

        Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);

        IF Checksum_Length>0 THEN
            FB_step:=50;
        ELSE
            ToError_Sts := TRUE;
            ToErrorID:=16#40;
            ToErrorDescrip:='Checksum length not valid';
        END_IF;
```

Figure 5.3.5.4 – CBT_ChangeBase FB algorithm step 40 building the message with the complete command and finds the number of characters in the string (source: own).

Step 50 calculates with XOR operation the string for the checksum (str_Checksum) and created the complete message to be sent (str_SendFrame) composed by:

- Header
- Length of the message
- Command ID
- String command
- CheckSum calculation



```
50: (*  Frame building  to send command to TMflow *)

    FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
      Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
    END_FOR;

    str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
    str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');

    Length:=LEN(str_SendFrame);
    Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

    IF Long>0 THEN
      FB_step:=60;
    ELSE
      ToError_Sts := TRUE;
      ToErrorID:=16#50;
      ToErrorDescrip:='Final frame building is error end';
    END_IF;
```

Figure 5.3.5.5 – CBT_ChangeBase FB algorithm step 50 building the complete message (source: own).

From step 60 to the end of the algorithm is already described in previous FB explanation.

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
10	Robot not connected	Use CBT_Connect FB for connection
11	Other FB is under execution	Wait until other FB is done
20	Clear buffer error	Check Ethernet connection wiring
40	Checksum length not valid	Re-execution is required
50	Final frame building is error end	Re-execution is required
60	TCP send error	Check Ethernet connection wiring
70	TCP receive error	Check Ethernet connection wiring
80	Command rejected	Re-execution is required

Table 5.3.5.5 – CBT_ChangeBase error list description and action (source: own).

5.3.6. CBT_ChangeTCP

This Function Block sends the command for changing the TCP offset value of the follow-up motions into buffer for execution.

Among other parameters, user must set the Cobot In/out variable to identify the target robot with its IP address and communications port, the command ID to identify the command in the feedback sent by the robot, the base coordinates (X, Y and Z in millimetres for positioning, and RS, RY and RZ in degrees for orientation), the weight (in kilograms), the moment of inertia (Ixx, Iyy and Izz in kg·mm²), and the location of the mass center (X, Y and Z in millimetres for positioning, and RX, RY and RZ in degrees for orientation).

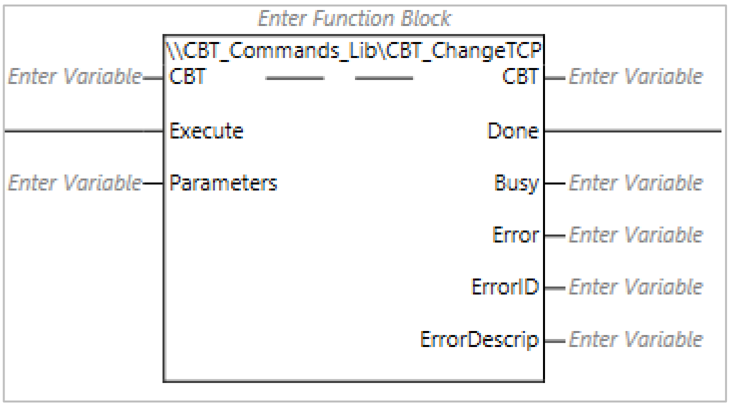


Figure 5.3.6.1 – CBT_ChangeTCP Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In/Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
Parameters	Input	CBT_Commands_Lib\stCBT_TCPParam	---	---	TCP parameters
Done	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is completed
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.6.1 – CBT_ChangeTCP variables type, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.6.2 – stCBT_Connection datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
CommandID	UINT	2 to 9	0	Identification for the return message for this command
TCPOffset	CBT_Commands_Lib\stCBT_Transformation	---	---	Tool Offset coordinates
Weight	UINT	0 to 14	0	Weight of the tool
MomentOfInertia	CBT_Commands_Lib\stCBT_MomentOfInertia	---	---	Tool’s moment of inertia
MassCenter	CBT_Commands_Lib\stCBT_Transformation	---	---	Frame reference of mass center
ExitNode	BOOL	TRUE or FALSE	FALSE	Quit Listen node in robot program

Table 5.3.6.3 – stCBT_ TCPParam datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
X	REAL	---	0	X coordinates in mm
Y	REAL	---	0	Y coordinates in mm
Z	REAL	---	0	Z coordinates in mm
RX	REAL	---	0	RX coordinates in degrees
RY	REAL	---	0	RY coordinates in degrees
RZ	REAL	---	0	RZ coordinates in degrees

Table 5.3.6.4 – stCBT_Transformation datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Ixx	REAL	---	0	Moment of inertia around X in kg·mm²
Iyy	REAL	---	0	Moment of inertia around Y in kg·mm²
Izz	REAL	---	0	Moment of inertia around Z in kg·mm²

Table 5.3.6.5 – stCBT_ MomentOfInertia datatype, range, default value and description (source: own).

Function Block Script

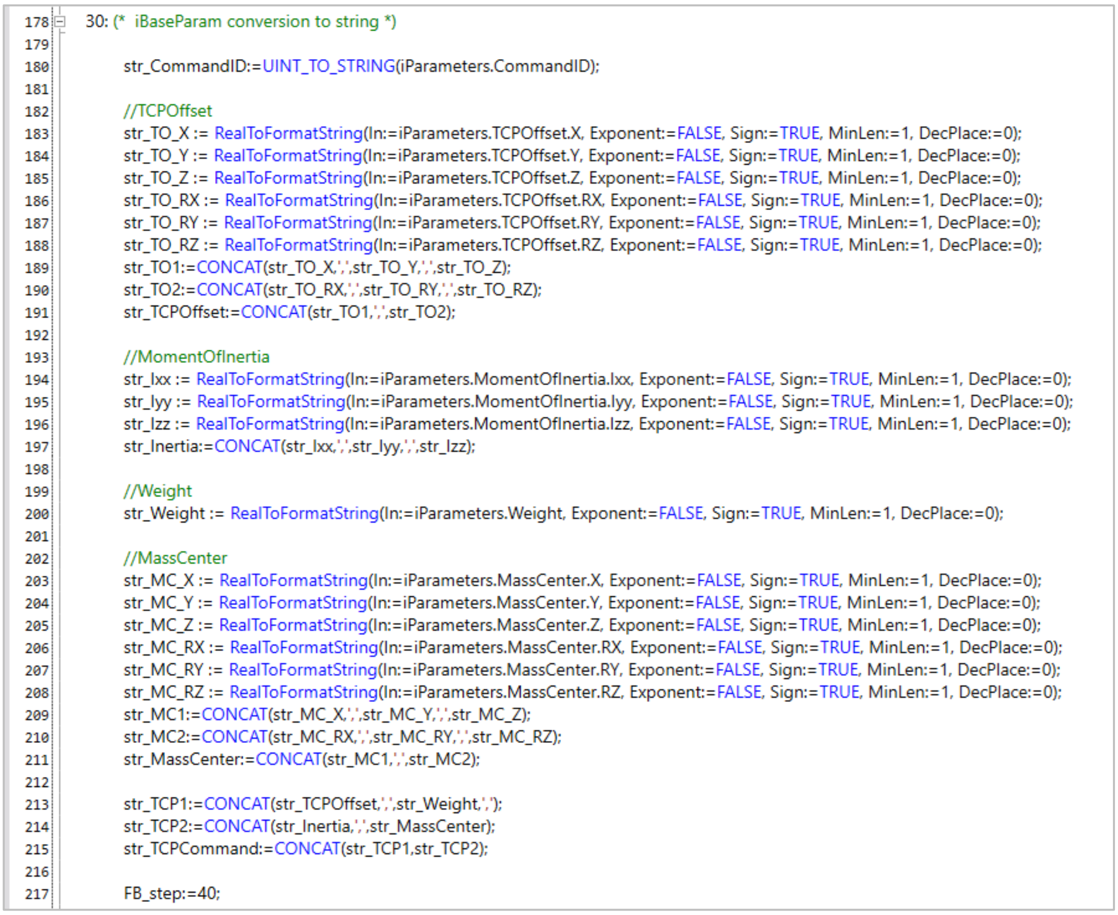
As all Function Blocks in this Library, once the upward signal differentiation of *Execute* input is detected and it is not in *Busy* state, the sequence initialization is started, otherwise an error is triggered. Sequence initialization has been described in *5.3.1.CBT_Connect* section.

As all Function Blocks in this Library, State Diagram Control is part of the script focused on output variable operation and timing. State Diagram Control has been described in *5.3.1.CBT_Connect* section.

From step 0 to 30 and from step 60 to the end, refer to the section *5.3.2. CBT_MoveAbsolute*.

Hereunder, the description of the specific script of CBT_ChangeTCP FB:

Step 30 converts TCP transformation offset parameter from real to string (str_TCPOffset), Inertia parameter from real to string (str_Inertia), Weight parameter from real to string (str_Weight), Mass Center parameter from real to string (str_MassCenter).



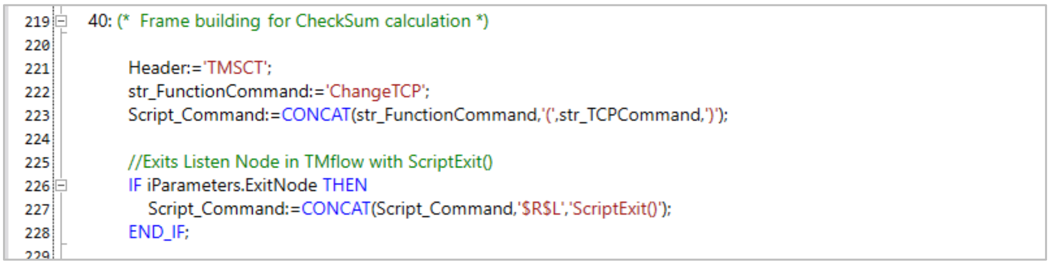
```
30: (*  iBaseParam conversion to string *)

        str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

        //TCPOffset
        str_TO_X := RealToFormatString(In:=iParameters.TCPOffset.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TO_Y := RealToFormatString(In:=iParameters.TCPOffset.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TO_Z := RealToFormatString(In:=iParameters.TCPOffset.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TO_RX := RealToFormatString(In:=iParameters.TCPOffset.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TO_RY := RealToFormatString(In:=iParameters.TCPOffset.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TO_RZ := RealToFormatString(In:=iParameters.TCPOffset.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TO1:=CONCAT(str_TO_X,',',str_TO_Y,',',str_TO_Z);
        str_TO2:=CONCAT(str_TO_RX,',',str_TO_RY,',',str_TO_RZ);
        str_TCPOffset:=CONCAT(str_TO1,',',str_TO2);

        //MomentOfInertia
        str_Ixx := RealToFormatString(In:=iParameters.MomentOfInertia.Ixx, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_Iyy := RealToFormatString(In:=iParameters.MomentOfInertia.Iyy, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_Izz := RealToFormatString(In:=iParameters.MomentOfInertia.Izz, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_Inertia:=CONCAT(str_Ixx,',',str_Iyy,',',str_Izz);

        //Weight
        str_Weight := RealToFormatString(In:=iParameters.Weight, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);

        //MassCenter
        str_MC_X := RealToFormatString(In:=iParameters.MassCenter.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_MC_Y := RealToFormatString(In:=iParameters.MassCenter.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_MC_Z := RealToFormatString(In:=iParameters.MassCenter.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_MC_RX := RealToFormatString(In:=iParameters.MassCenter.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_MC_RY := RealToFormatString(In:=iParameters.MassCenter.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_MC_RZ := RealToFormatString(In:=iParameters.MassCenter.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_MC1:=CONCAT(str_MC_X,',',str_MC_Y,',',str_MC_Z);
        str_MC2:=CONCAT(str_MC_RX,',',str_MC_RY,',',str_MC_RZ);
        str_MassCenter:=CONCAT(str_MC1,',',str_MC2);

        str_TCP1:=CONCAT(str_TCPOffset,',',str_Weight,',');
        str_TCP2:=CONCAT(str_Inertia,',',str_MassCenter);
        str_TCPCommand:=CONCAT(str_TCP1,str_TCP2);

        FB_step:=40;
```

Figure 5.3.6.2 – CBT_ChangeTCP FB algorithm step 30 creating the string of the complete command (source: own).

Step 40 builds the message (script_command) to be sent to the robot including:

- Function command
- TCP offset

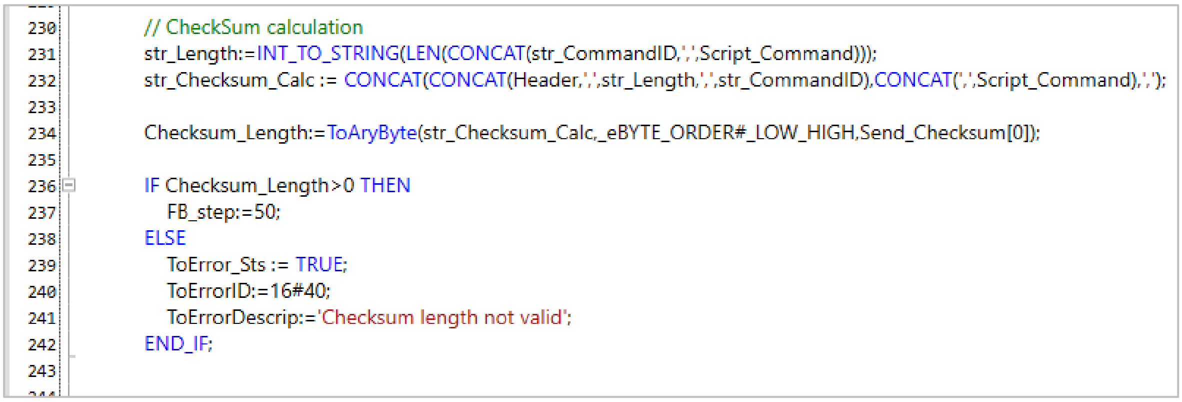
- ➢ If ExitNode is enabled, ScriptExit() command is included in the message. It exits external script control mode once the last motion command has been executed and finished. Listen Node is quitted, and not further external script functions commands are accepted.



```
40: (*  Frame building  for CheckSum calculation *)

        Header:='TMSCT';
        str_FunctionCommand:='ChangeTCP';
        Script_Command:=CONCAT(str_FunctionCommand,'(',str_TCPCommand,')');

        //Exits Listen Node in TMflow with ScriptExit()
        IF iParameters.ExitNode THEN
           Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
        END_IF;
```

Figure 5.3.6.3 – CBT_ChangeTCP FB algorithm step 40 building the message with the complete command (source: own).

Once the message has been built, it is needed to find the number of characters of (str_CommandID + Script_Command) and to store it in the string (str_Length). Then, joining all parts (Header + str_Length + str_CommandID + Script_Command) to calculate the checksum (str_Checksum_Calc).

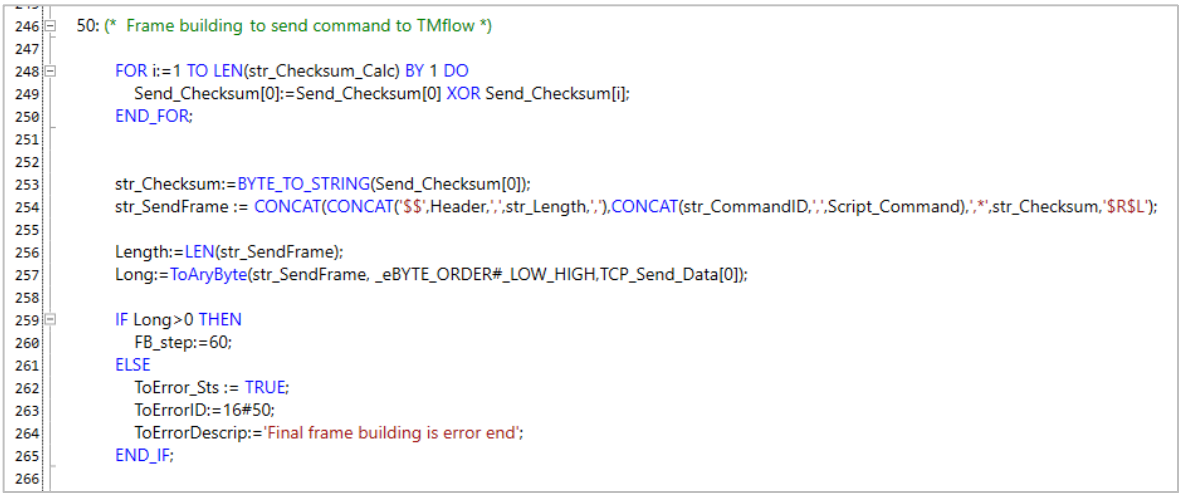


```
        // CheckSum calculation
        str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));
        str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');

        Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);

        IF Checksum_Length>0 THEN
           FB_step:=50;
        ELSE
           ToError_Sts := TRUE;
           ToErrorID:=16#40;
           ToErrorDescrip:='Checksum length not valid';
        END_IF;
```

Figure 5.3.6.4 – CBT_ChangeTCP FB algorithm step 40 building the message with the complete command and finds the number of characters in the string (source: own).

Step 50 calculates with XOR operation the string for the checksum (str_Checksum) and created the complete message to be sent (str_SendFrame) composed by:

- Header
- Length of the message
- Command ID
- String command
- CheckSum calculation



```
50: (*  Frame building  to send command to TMflow *)

    FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
       Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
    END_FOR;


    str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
    str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');

    Length:=LEN(str_SendFrame);
    Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

    IF Long>0 THEN
       FB_step:=60;
    ELSE
       ToError_Sts := TRUE;
       ToErrorID:=16#50;
       ToErrorDescrip:='Final frame building is error end';
    END_IF;

```

Figure 5.3.6.5 – CBT_ChangeTCP FB algorithm step 50 building the complete message (source: own).

From step 60 to the end of the algorithm, refer to the section *5.3.2. CBT_MoveAbsolute*.

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
10	Robot not connected	Use CBT_Connect FB for connection
11	Other FB is under execution	Wait until other FB is done
20	Clear buffer error	Check Ethernet connection wiring
25	Weight range: 0~14 kg	Set parameter in acceptable range
40	Checksum length not valid	Re-execution is required
50	Final frame building is error end	Re-execution is required
60	TCP send error	Check Ethernet connection wiring
70	TCP receive error	Check Ethernet connection wiring
80	Command rejected	Re-execution is required

Table 5.3.6.6 – CBT_ChangeTCP error list description and action (source: own).

5.3.7. CBT_ProgramControl

This Function Block sends the command to the robot to pause or resume the program execution. If the program is paused during robot movement, the robot will complete the interrupted movement once the program execution is restored.

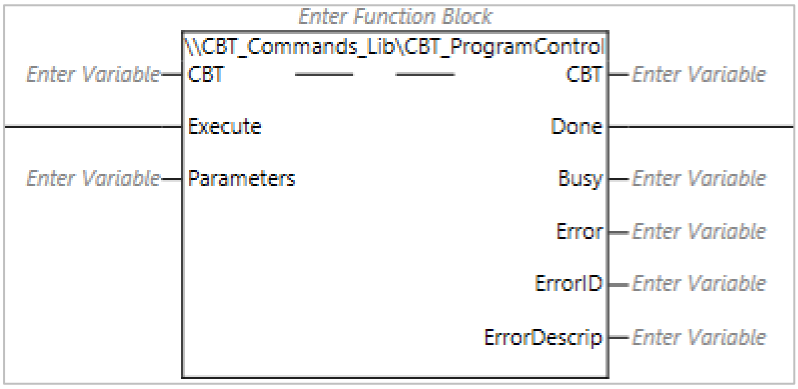


Figure 5.3.7.1 – CBT_ProgramControl Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In/Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
Parameters	Input	CBT_Commands_Lib\stCBT_PrgControl	---	---	Program control parameters
Done	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is completed
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.7.1 – CBT_ProgramControl variables, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.7.2 – stCBT_Connection datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
CommandID	UINT	2 to 9	0	Identification for the return message for this command
Command	CBT_Commands_Lib\eCBT_PrgControlCmd	---	---	Pause or resume program

Table 5.3.7.3 – stCBT_PrgControl datatype, range, default value and description (source: own).

Name	Enum Value	Description
Pause	0	Pause the project and the motion of the robot
Resume	1	Resume the project and the motion of the robot

Table 5.3.7.4 – eCBT_PrgControlCmd datatype description (source: own).

Function Block Script

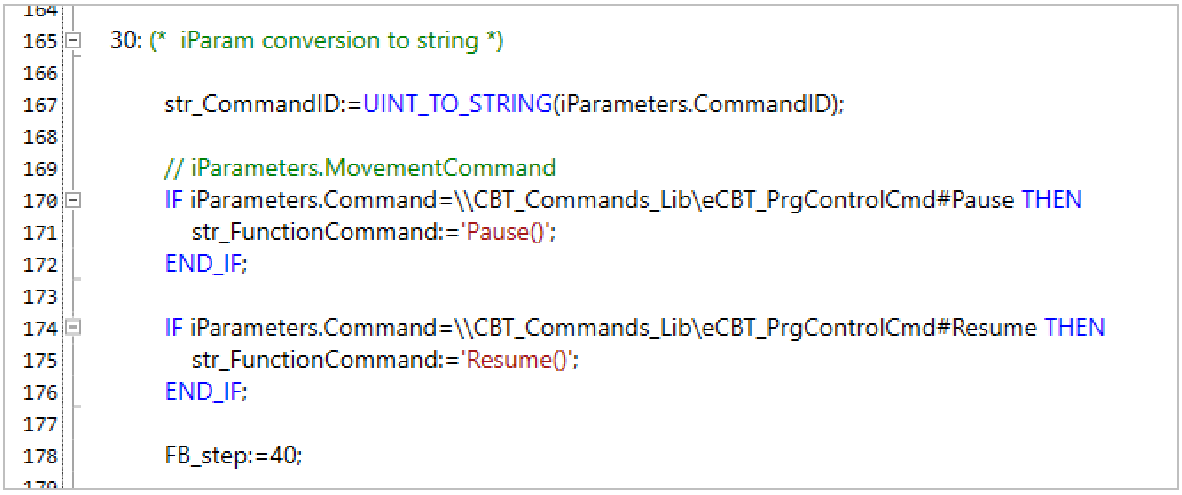
As all Function Blocks in this Library, once the upward signal differentiation of *Execute* input is detected and it is not in *Busy* state, the sequence initialization is started, otherwise an error is triggered. Sequence initialization has been described in *5.3.1.CBT_Connect* section.

As all Function Blocks in this Library, State Diagram Control is part of the script focused on output variable operation and timing. State Diagram Control has been described in *5.3.1.CBT_Connect* section.

From step 0 to 30 and from step 60 to the end, refer to the section *5.3.2. CBT_MoveAbsolute*.

Hereunder, the description of the specific script of CBT_ProgramControl FB:

Step 30 converts Program Control Command parameter from enumerated to string (str_FunctionCommand).

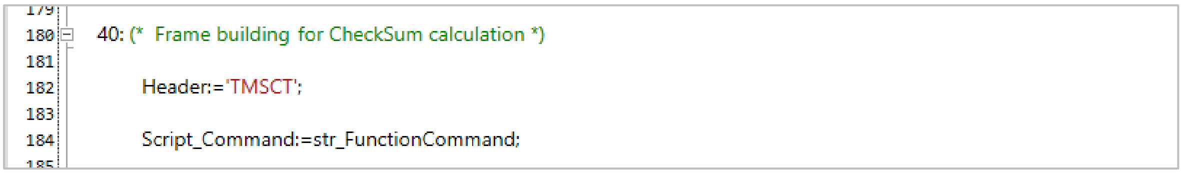


```
30: (*  iParam conversion to string *)

        str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

        // iParameters.MovementCommand
        IF iParameters.Command=\\CBT_Commands_Lib\eCBT_PrgControlCmd#Pause THEN
            str_FunctionCommand:='Pause()';
        END_IF;

        IF iParameters.Command=\\CBT_Commands_Lib\eCBT_PrgControlCmd#Resume THEN
            str_FunctionCommand:='Resume()';
        END_IF;

        FB_step:=40;
```

Figure 5.3.7.2 – CBT_ProgramControl FB algorithm step 30 creating the string of the complete command (source: own).

Step 40 builds the message (script_command) to be sent to the robot including:

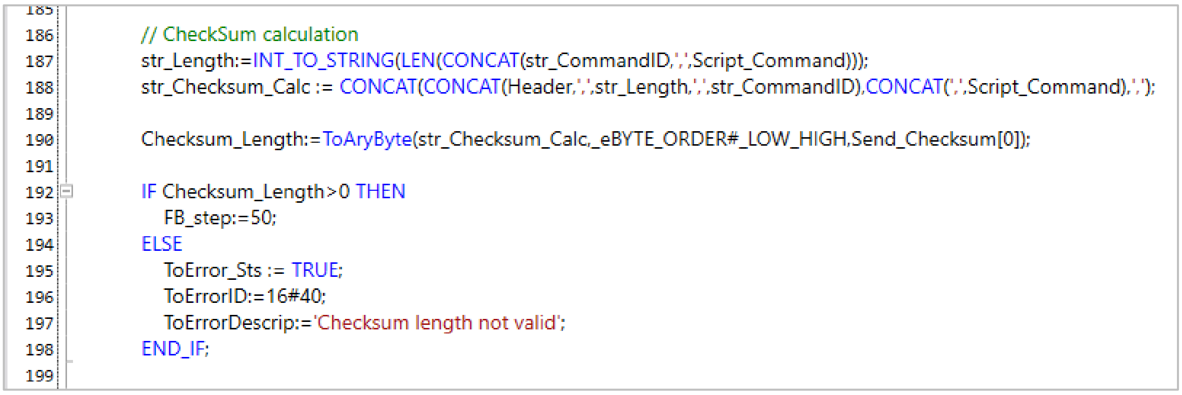
- Function command



```
40: (*  Frame building  for CheckSum calculation *)

        Header:='TMSCT';

        Script_Command:=str_FunctionCommand;
```

Figure 5.3.7.3 – CBT_ProgramControl FB algorithm step 40 building the message with the complete command (source: own).

Once the message has been built, it is needed to find the number of characters of (str_CommandID + Script_Command), and to store it in the string (str_Length). Then, joining all parts (Header + str_Length + str_CommandID + Script_Command) to calculate the checksum (str_Checksum_Calc).

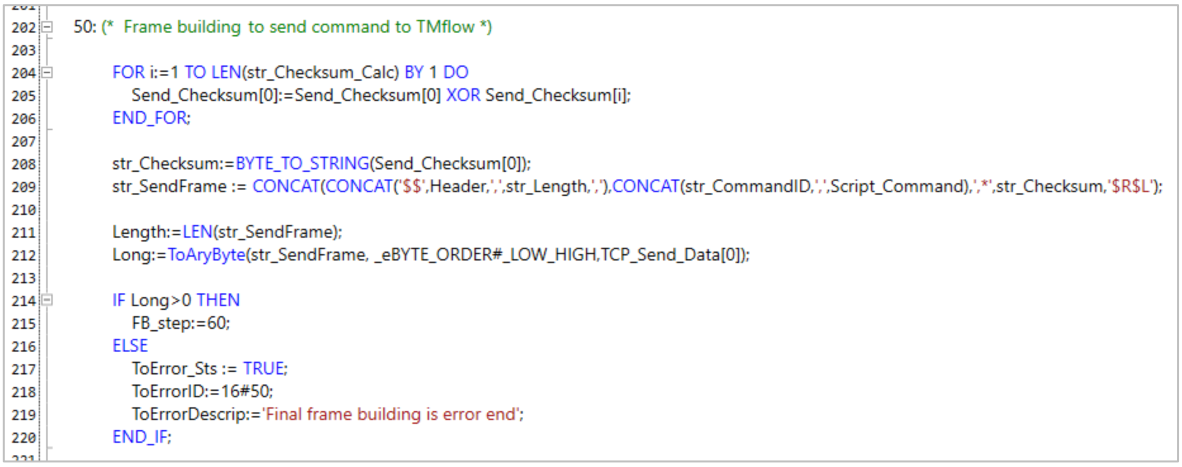


```
    // CheckSum calculation
    str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));
    str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');

    Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);

    IF Checksum_Length>0 THEN
        FB_step:=50;
    ELSE
        ToError_Sts := TRUE;
        ToErrorID:=16#40;
        ToErrorDescrip:='Checksum length not valid';
    END_IF;
```

Figure 5.3.7.4 – CBT_ProgramControl FB algorithm step 40 building the message with the complete command and finds the number of characters in the string (source: own).

Step 50 calculates with XOR operation the string for the checksum (str_Checksum) and created the complete message to be sent (str_SendFrame) composed by:

- Header
- Length of the message
- Command ID
- String command
- CheckSum calculation



```
50: (*  Frame building  to send command to TMflow *)

    FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
        Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
    END_FOR;

    str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
    str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');

    Length:=LEN(str_SendFrame);
    Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

    IF Long>0 THEN
        FB_step:=60;
    ELSE
        ToError_Sts := TRUE;
        ToErrorID:=16#50;
        ToErrorDescrip:='Final frame building is error end';
    END_IF;
```

Figure 5.3.7.5 – CBT_ProgramControl FB algorithm step 50 building the complete message (source: own).

From step 60 to the end of the algorithm, refer to the section *5.3.2. CBT_MoveAbsolute*.

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
10	Robot not connected	Use CBT_Connect FB for connection
20	Clear buffer error	Check Ethernet connection wiring
40	Checksum length not valid	Re-execution is required
50	Final frame building is error end	Re-execution is required
60	TCP send error	Check Ethernet connection wiring
70	TCP receive error	Check Ethernet connection wiring
80	Command rejected	Re-execution is required

Table 5.3.7.5 – CBT_ProgramControl error list description and action (source: own).

5.4. Program example

A program example in Ladder language (IEC 61131-3 language) has been created to execute each Function Block independently. Each rung contains: Execute input contact, FB instance and Done output relay. Almost all variables have been declared as Global Variables to be reachable from and HMI.

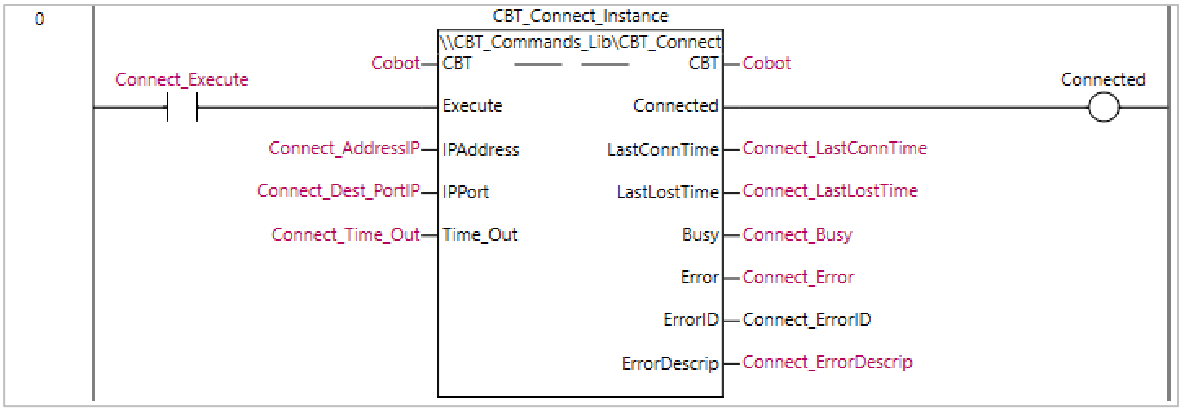


Figure 5.4.1 – CBT_Connect FB in the program example (source: own).

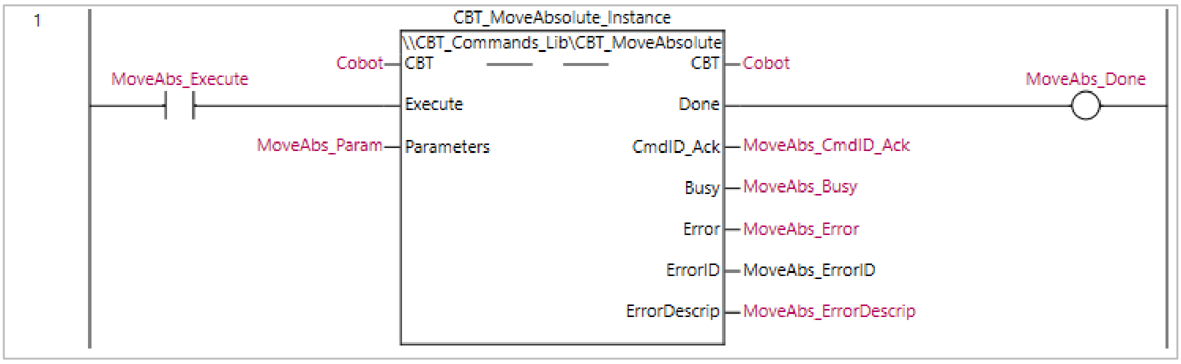


Figure 5.4.2 – CBT_MoveAbsolute FB in the program example (source: own).

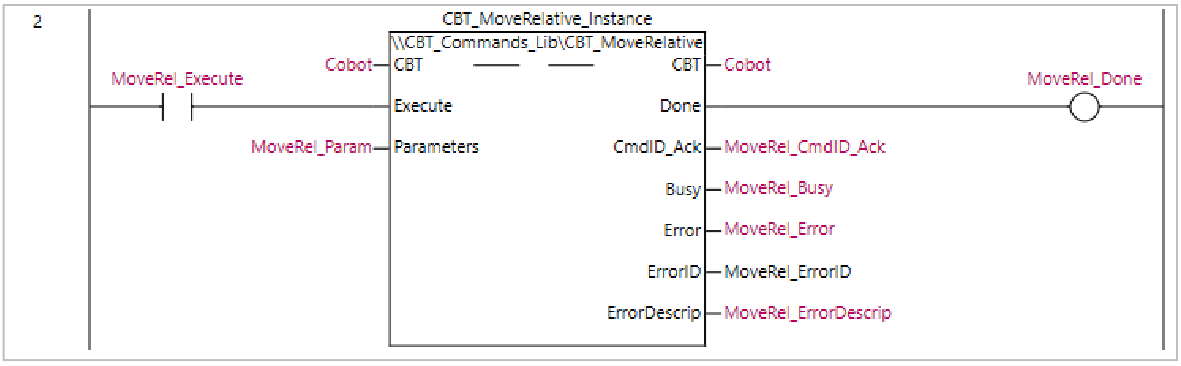


Figure 5.4.3 – CBT_MoveRelative FB in the program example (source: own).

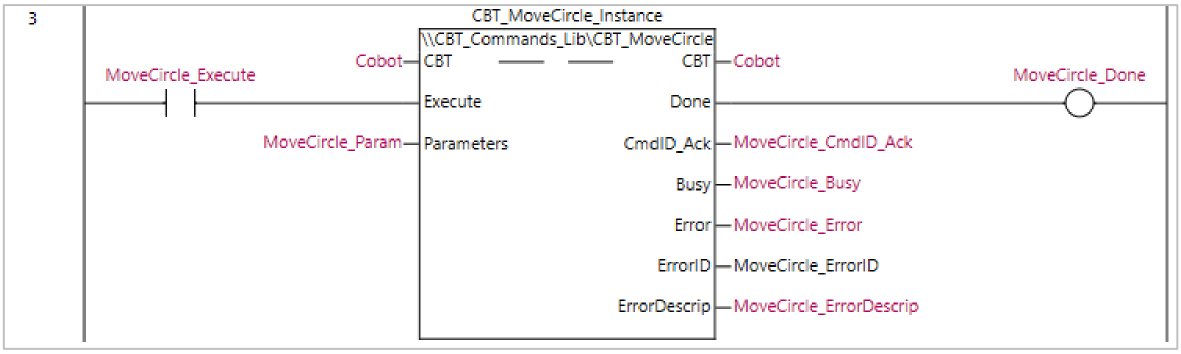


Figure 5.4.4 – CBT_MoveCircle FB in the program example (source: own).

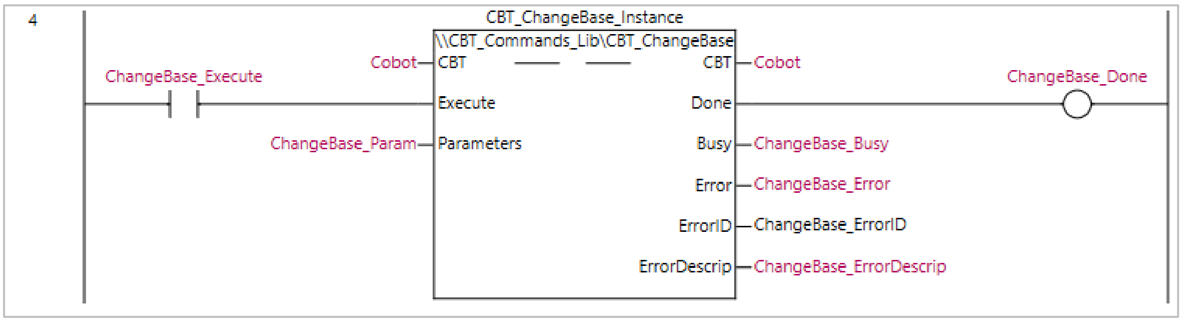


Figure 5.4.5 – CBT_ChangeBase FB in the program example (source: own).

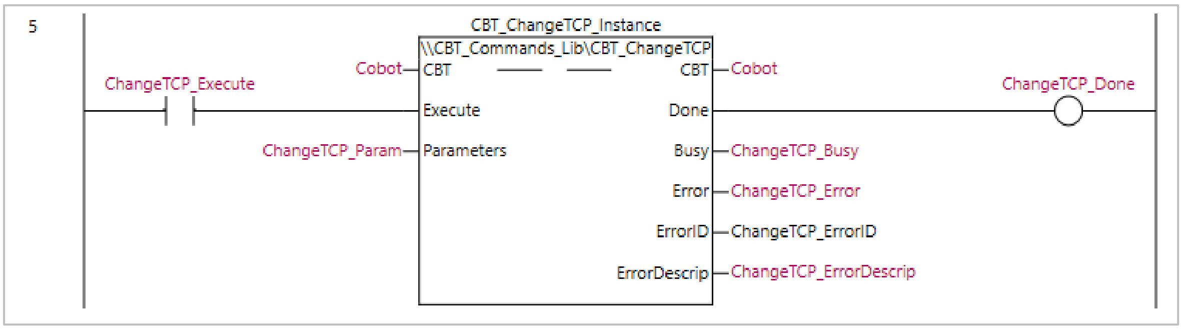


Figure 5.4.6 – CBT_ChangeTCP FB in the program example (source: own).

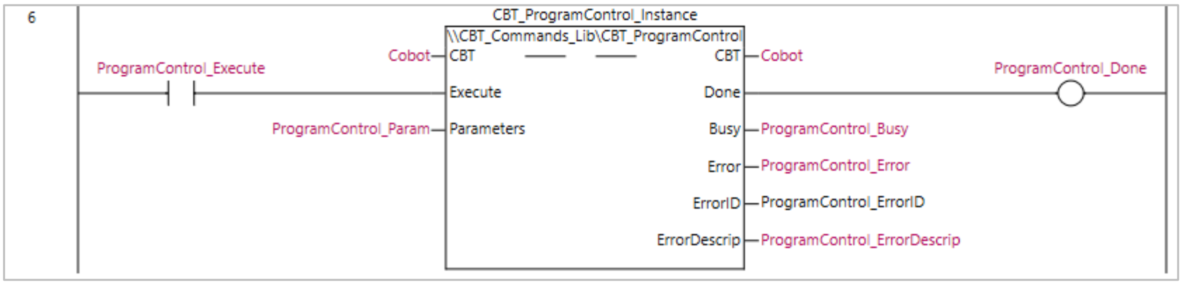


Figure 5.4.7 – CBT_ProgramControl FB in the program example (source: own).

A second program has been created for a PnP (Pick and Place) application in which only the CBT_MoveAbsolute has been used. For this, the program is composed by 6 steps, each one corresponds to one of the 6 segments of the typical PnP trajectory cycle (see the whole program in the Annex *B1. Pick and Place sequence*).

On this program example, Cobot in/out variable identifying the target robot with its IP address and communications port must be the same one. Then, for each one of the 6 CBT_MoveAbsolute instances execution, it is possible to set the command ID to identify the command in the feedback sent by the robot in case there is no blending, speed including units, acceleration time, if blending is required and if it is performed by percentage or by radius and the robot arm configuration for PTP movements.

In the image below it can be seen the trajectory of the robot TCP (Tool Center Point) description for a typical PnP cycle.

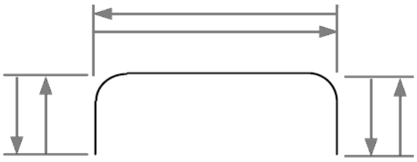


Figure 5.4.8 – PnP cycle (source: own).

6. HMI

6.1. Description

The target of developing this GUI (Graphical User Interface) is providing to the user a simple graphical environment to easily use the Function Blocks and send commands to the robot. The project has been designed defining a Main page as primary menu giving direct access to secondary pages with different purposes:

- Get connection with robot.
- Usage of all function and motion command Function Blocks.
- PnP application example.

Sysmac Studio is the software used to do the programming of the HMI. It uses VB.NET (Visual Basic .NET) object-oriented programming language implemented in .NET Framework. Most of the graphical components are configured using element properties like appearance (text font, margins, colours, visibility), behaviour (variable associated with maximum and minimum values, display format, availability, etc), layout (position and size) and security (access level or visibility level). Events and action can be set when pressing or releasing a button (buzzer, call a routine of VB programming, open a page, reset a variable, etc.).

6.2. Mapping Variables

Mapping variables refers to assigning global variables in the PLC connected to the HMI to global variables in the HMI. Therefore, mapping variables is required to link input value from user in the HMI with the input parameters in the FBs.

PLC Variables	DataType	HMI Variable
ChangeBase_Busy	BOOL	HMI_ChangeBase_Busy
ChangeBase_Done	BOOL	HMI_ChangeBase_Done
ChangeBase_Error	BOOL	HMI_ChangeBase_Error
ChangeBase_ErrorDescrip	STRING[50]	HMI_ChangeBase_ErrorDescrip
ChangeBase_Execute	BOOL	HMI_ChangeBase_Execute
ChangeBase_Param	CBT_Commands_Lib\stCBT_BaseParam	HMI_ChangeBase_Parameters
ChangeTCP_Busy	BOOL	HMI_ChangeTCP_Busy

ChangeTCP_Done	BOOL	HMI_ChangeTCP_Done
ChangeTCP_Error	BOOL	HMI_ChangeTCP_Error
ChangeTCP_ErrorDescrip	STRING[50]	HMI_ChangeTCP_ErrorDescrip
ChangeTCP_Execute	BOOL	HMI_ChangeTCP_Execute
ChangeTCP_Param	CBT_Commands_Lib\stCBT_TCPParam	HMI_ChangeTCP_Parameters
Cobot	CBT_Commands_Lib\stCBT_Connection	HMI_Cobot
Connect_AddressIP	STRING[50]	HMI_Connect_AddressIP
Connect_Busy	BOOL	HMI_Connect_Busy
Connect_Dest_PortIP	INT	HMI_Connect_Dest_PortIP
Connect_Error	BOOL	HMI_Connect_Error
Connect_ErrorDescrip	STRING[50]	HMI_Connect_ErrorDescrip
Connect_Execute	BOOL	HMI_Connect_Execute
Connect_LastConnTime	DATE_AND_TIME	HMI_Connect_LastConnTime
Connect_LastLostTime	DATE_AND_TIME	HMI_Connect_LastLostTime
Connect_Time_Out	UINT	HMI_Connect_Time_Out
MoveAbs_Busy	BOOL	HMI_MoveAbs_Busy
MoveAbs_CmdID_Ack	UINT	HMI_MoveAbs_CmdID_Ack
MoveAbs_Done	BOOL	HMI_MoveAbs_Done
MoveAbs_Error	BOOL	HMI_MoveAbs_Error
MoveAbs_ErrorDescrip	STRING[50]	HMI_MoveAbs_ErrorDescrip
MoveAbs_Execute	BOOL	HMI_MoveAbs_Execute
MoveAbs_Param	CBT_Commands_Lib\stCBT_MovAbsParam	HMI_MoveAbs_Parameters
MoveCircle_Busy	BOOL	HMI_MoveCircle_Busy
MoveCircle_CmdID_Ack	UINT	HMI_MoveCircle_CmdID_Ack
MoveCircle_Done	BOOL	HMI_MoveCircle_Done
MoveCircle_Error	BOOL	HMI_MoveCircle_Error
MoveCircle_ErrorDescrip	STRING[50]	HMI_MoveCircle_ErrorDescrip
MoveCircle_Execute	BOOL	HMI_MoveCircle_Execute
MoveCircle_Param	CBT_Commands_Lib\stCBT_MovCircleParam	HMI_MoveCircle_Parameters
MoveRel_Busy	BOOL	HMI_MoveRel_Busy
MoveRel_CmdID_Ack	UINT	HMI_MoveRel_CmdID_Ack
MoveRel_Done	BOOL	HMI_MoveRel_Done
MoveRel_Error	BOOL	HMI_MoveRel_Error
MoveRel_ErrorDescrip	STRING[50]	HMI_MoveRel_ErrorDescrip
MoveRel_Execute	BOOL	HMI_MoveRel_Execute
MoveRel_Param	CBT_Commands_Lib\stCBT_MovRelParam	HMI_MoveRel_Parameters
PnP_Example_Execute	BOOL	HMI_PnP_Example_Execute
PnP_Move_CmdID_Ack	UINT	HMI_PnP_Move_CmdID_Ack
PnP_MoveBusy	BOOL	HMI_PnP_MoveBusy
PnP_MoveDone	BOOL	HMI_PnP_MoveDone
PnP_MoveError	BOOL	HMI_PnP_MoveError
PnP_MoveErrorDescrip	STRING[50]	HMI_PnP_MoveErrorDescrip
PnP_MoveParameters	ARRAY[0..5] OF CBT_Commands_Lib\stCBT_MovAbsParam	HMI_PnP_MoveParameters
ProgramControl_Busy	BOOL	HMI_ProgramControl_Busy
ProgramControl_Done	BOOL	HMI_ProgramControl_Done
ProgramControl_Error	BOOL	HMI_ProgramControl_Error
ProgramControl_ErrorDescrip	STRING[50]	HMI_ProgramControl_ErrorDescrip
ProgramControl_Execute	BOOL	HMI_ProgramControl_Execute
ProgramControl_Param	CBT_Commands_Lib\stCBT_PrgControl	HMI_ProgramControl_Parameters

Table 6.2.1 – PLC – HMI variables mapping (source: own).

6.3. Pages

Pages are described in detail in the following sections, but here a picture of the whole navigation tree is shown.

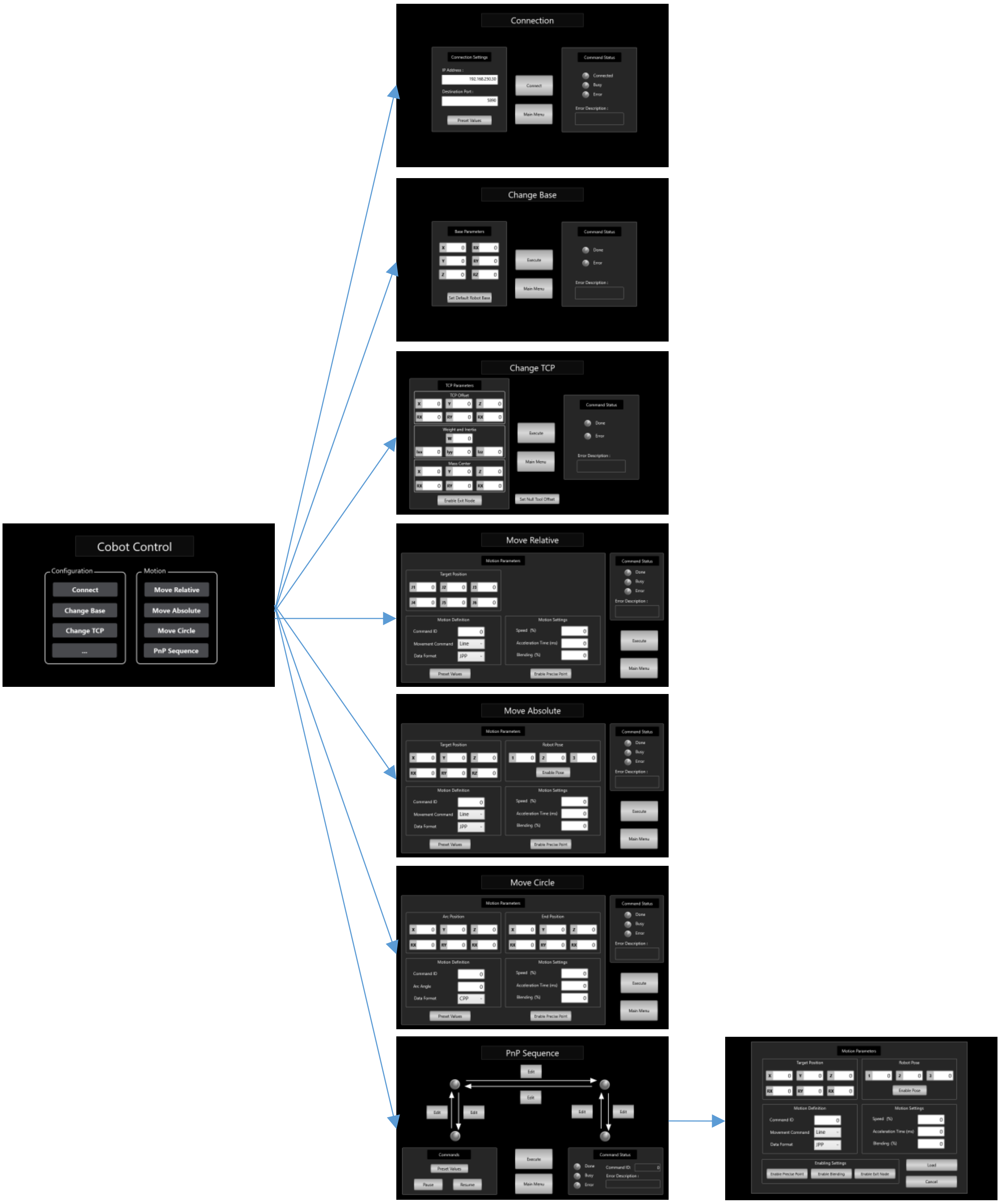


Figure 6.3.0 – HMI navigation tree (source: own).

6.3.1. Main

Once the touchscreen boots up, this is the initial page. It gives access to all the secondary pages which independently handles each FB in the library. As shown in the picture below, it is composed by 2 areas: Configuration and Motion.

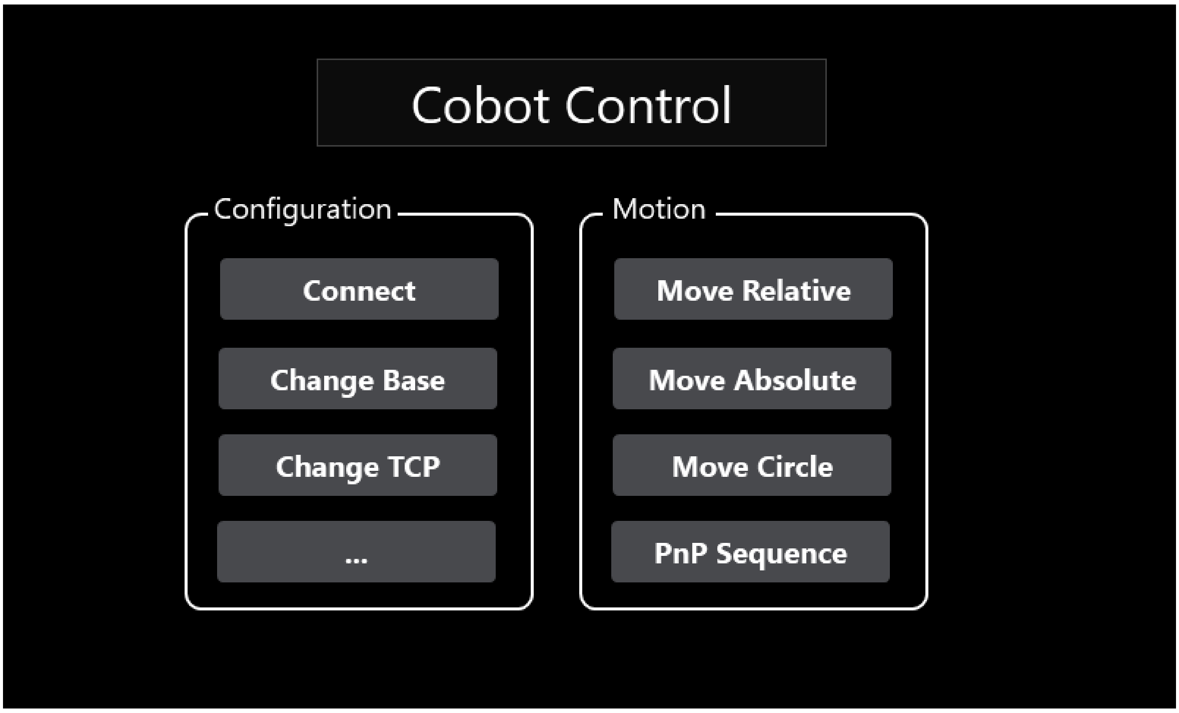


Figure 6.3.1 – Main page of HMI (source: own).

Configuration area includes the actions most needed to start moving the robot, connection with controller, base coordinates, and TCP settings:

- *Connect* button. Opens the page in which the connection with the robot can be done. Variables are mapped to the *CBT_Connect* Function Block in the program example (*Figure 5.4.1*).
- *Change Base* button. Opens the page in which the base coordinate can be set for the follow-up motions into buffer for execution. Variables are mapped to the *CBT_ChangeBase* Function Block in the program example (*Figure 5.4.5*).
- *Change TCP* button. Opens the page in which the TCP coordinates, weight, inertia, and mass center can be configured for the follow-up motions into buffer for execution. Variables are mapped to the *CBT_ChangeTCP* Function Block in the program example (*Figure 5.4.6*).
- A fourth button is reserved for future implementations.

Motion area includes the different motion commands and a Pick and Place application program sample.

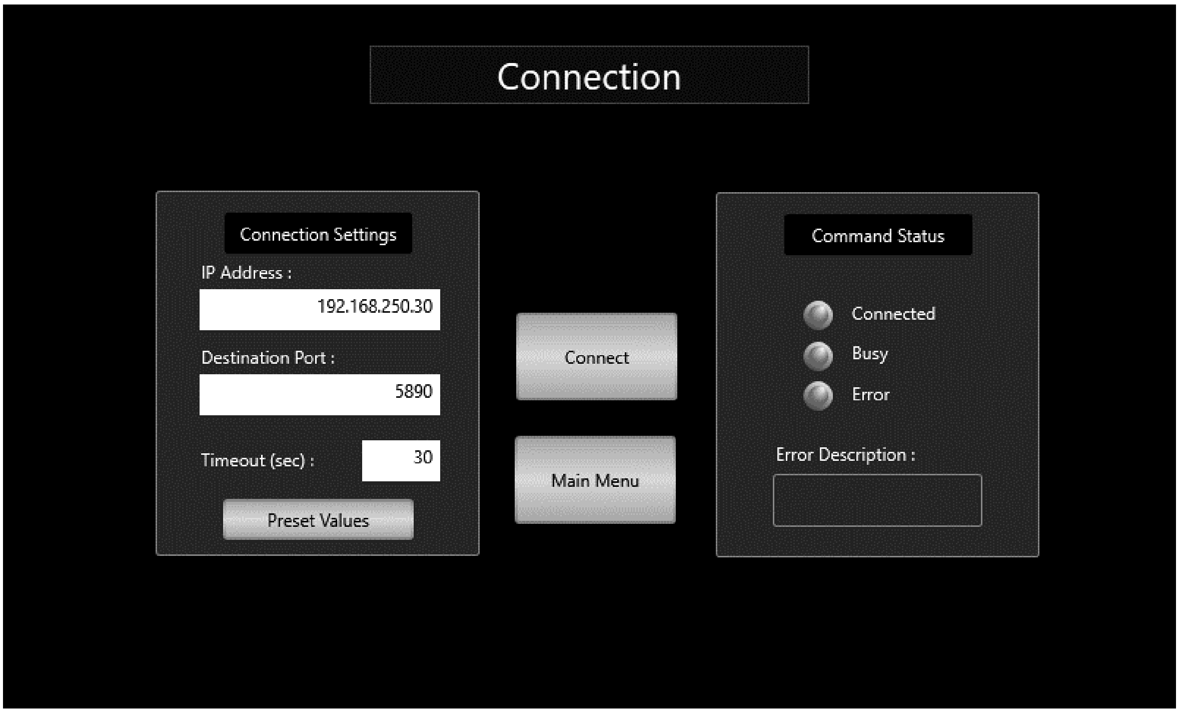
- *Move Relative* button. Opens the page in which the robot can be commanded to a relative position defining the motion parameters. Variables are mapped to the *CBT_MoveRelative* Function Block in the program example (*Figure 5.4.3*).
- *Move Absolute* button. Opens the page in which the robot can be commanded to a, absolute position defining the motion parameters. Variables are mapped to the *CBT_MoveAbsolute* Function Block in the program example (*Figure 5.4.2*).
- *Move Circle* button. Opens the page in which the robot can be commanded for a circular movement defining the motion parameters. Variables are mapped to the *CBT_MoveCircular* Function Block in the program example (*Figure 5.4.4*).
- *PnP Sequence* Button. Opens the page in which the robot can be commanded to perform a PnP application. Each of the 6 motion segments can be configured. Variables are mapped to the *CBT_ProgramControl* Function Block in the program example (*Figure 5.4.7*).

6.3.2. pgConnect

This page is used as interface for *CBT_Connect* Function Block. Inputs can be configured and outputs are monitored. It is composed by 2 areas:

- *Connection Settings*, where Robot IP address, connection port and the timeout (before the FB triggers an error if connection cannot be done) must be defined. *Preset Values* button is used to set predefined values configured in the page script for input parameters.
- *Command Status*, region where the FB outputs are displayed to monitor FB status.

Connect button executes *CBT_Connect* Function Block (*Figure 5.4.1*) and *Main Menu* button returns to *Main* page.



IP Address : 192.168.250.30

Destination Port : 5890

Timeout (sec) : 30

Error Description :

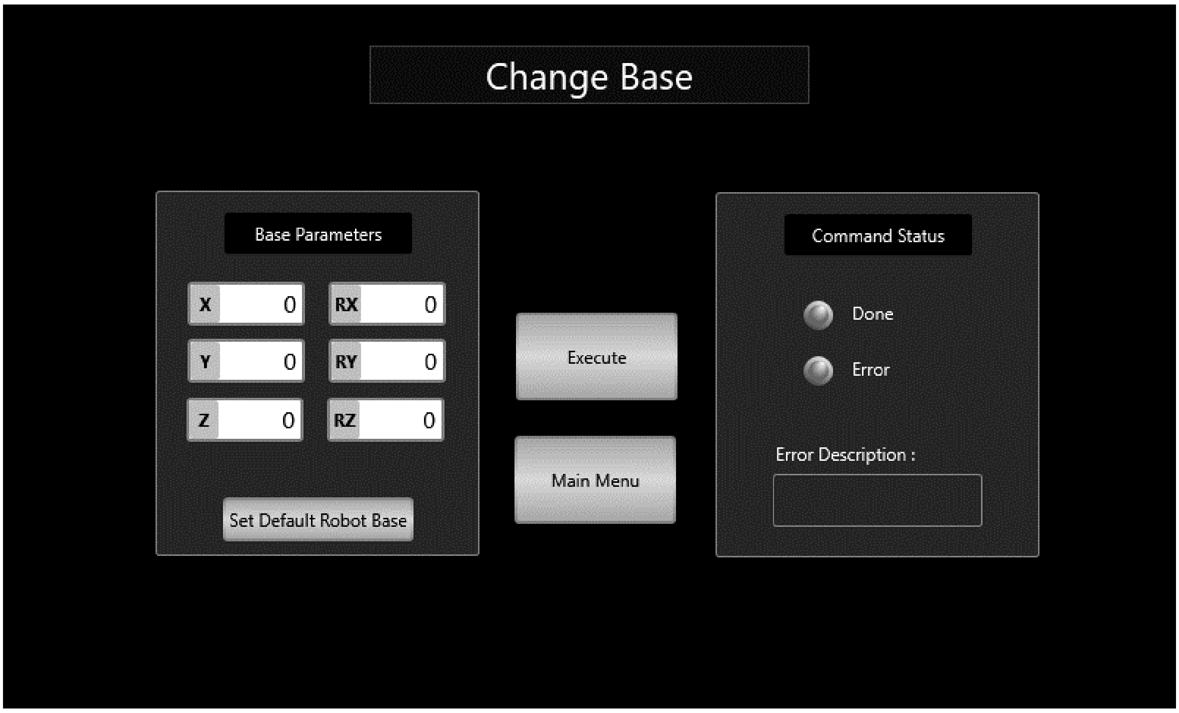
Figure 6.3.2 – pgConnect page of HMI (source: own).

6.3.3. pgChangeBase

This page is used as interface for *CBT_ChangeBase* Function Block. Inputs can be configured and outputs are monitored. It is composed by 2 areas:

- *Base Parameters,* where the base coordinate (mm, ◦) must be set. *Set Default Robot Base* button is used to set predefined values configured in the page script for input parameters. In this case, to reset the base coordinates.
- *Command Status,* region where the FB outputs are displayed to monitor FB status.

Execute button executes *CBT_ChangeBase* Function Block (*Figure 5.4.5*) and *Main Menu* button returns to *Main* page.



Change Base

Base Parameters

X 0

RX 0

Y 0

RY 0

Z 0

RZ 0

Command Status

Error Description :

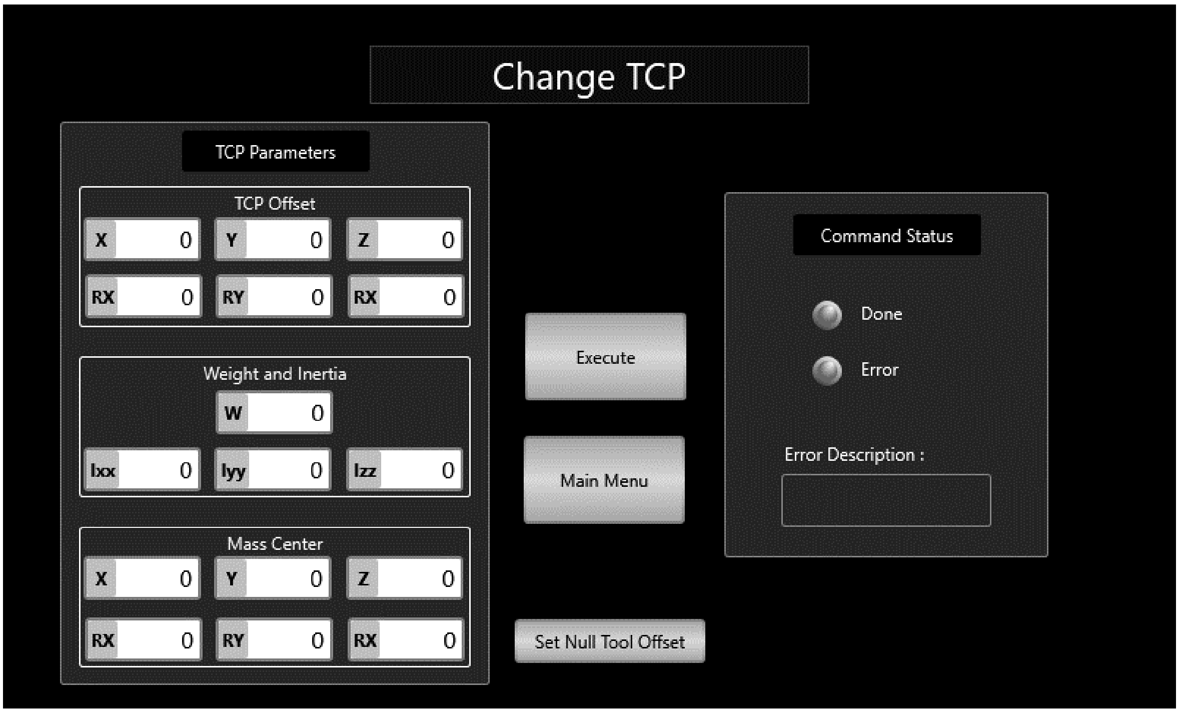
Figure 6.3.3 – pgChangeBase page of HMI (source: own).

6.3.4. pgChangeTCP

This page is used as interface for *CBT_ChangeTCP* Function Block. Inputs can be configured and outputs are monitored. It is composed by 2 areas:

- *TCP Parameters,* where the TCP offset (mm, ◦), Weight (kg), Inertia (kg·mm^2) and Mass Center (mm, ◦) coordinates must be set. *Set Null Tool Offset* button is used to set predefined values configured in the page script for input parameters. In this case, null values for input parameters like no tool was selected.
- *Command Status,* region where the FB outputs are displayed to monitor FB status.

Execute button executes *CBT_ChangeTCP* Function Block (*Figure 5.4.6*) and *Main Menu* button returns to *Main* page.



Change TCP

TCP Parameters

TCP Offset

X 0 Y 0 Z 0

RX 0 RY 0 RX 0

Weight and Inertia

W 0

Ixx 0 Iyy 0 Izz 0

Mass Center

X 0 Y 0 Z 0

RX 0 RY 0 RX 0

Command Status

Error Description :

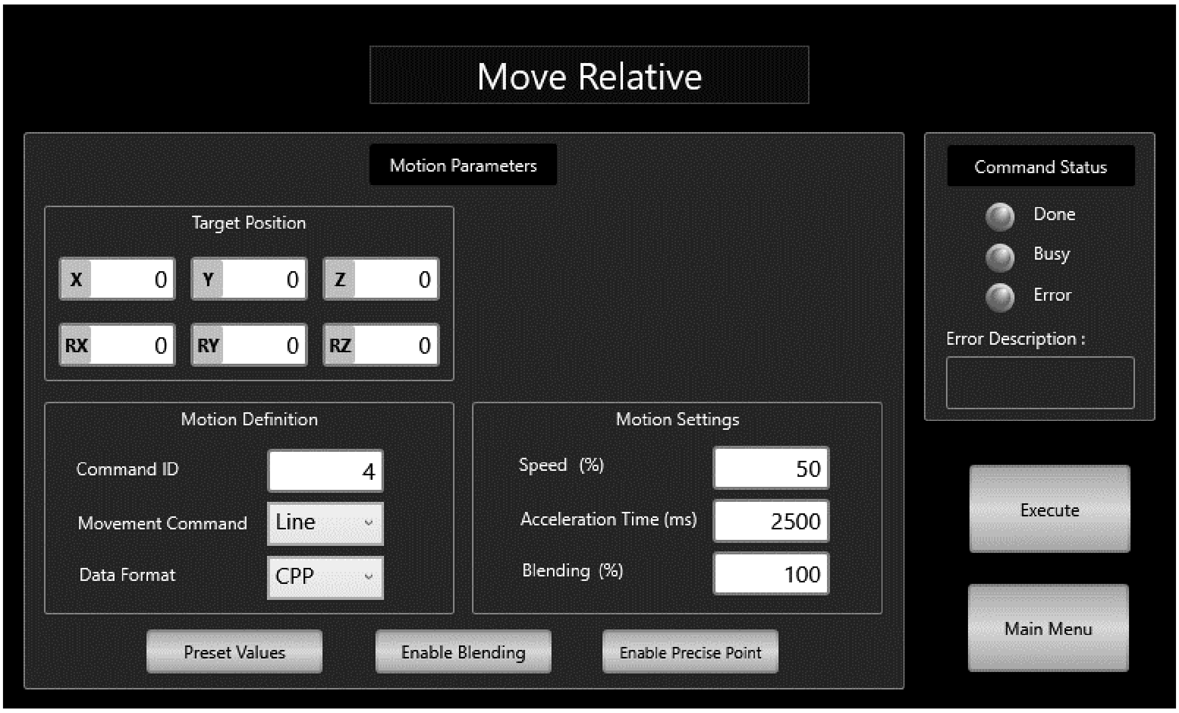
Figure 6.3.4 – pgChangeTCP page of HMI (source: own).

6.3.5. pgMoveRelative

This page is used as interface for *CBT_MoveRelative* Function Block. Inputs can be configured and outputs are monitored. It is composed by 2 areas:

- *Motion Parameters,* where the Target Position (mm, ◦), Motion Command (Line or PTP) and Motion Settings must be set. *Preset Values* button is used to set predefined values configured in the page script for input parameters. In this case, null values. *Enable Blending* and *Enable Precise Point* are also optional settings.
- *Command Status,* region where the FB outputs are displayed to monitor FB status.

Execute button executes *CBT_MoveRelative* Function Block (*Figure 5.4.3*) and *Main Menu* button returns to *Main* page.



Move Relative

Target Position

X 0 Y 0 Z 0

RX 0 RY 0 RZ 0

Command ID 4

Movement Command Line

Data Format CPP

Speed (%) 50

Acceleration Time (ms) 2500

Blending (%) 100

Error Description :

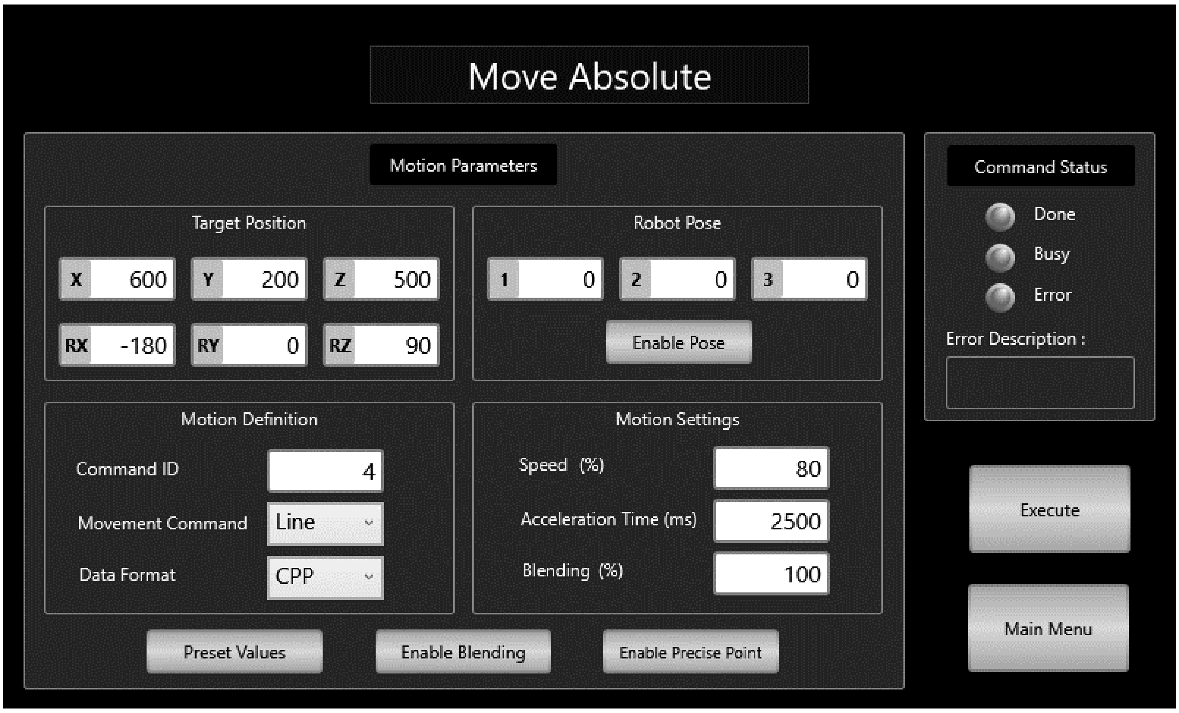
Figure 6.3.5 – pgMoveRelative page of HMI (source: own).

6.3.6. pgMoveAbsolute

This page is used as interface for *CBT_MoveAbsolute* Function Block. Inputs can be configured and outputs are monitored. It is composed by 2 areas:

- *Motion Parameters,* where the Target Position (mm, ◦), Robot Pose (Righty/Lefty, Above/Below and Flip/Noflip), Motion Command (Line or PTP) and Motion Settings must be set. *Preset Values* button is used to set predefined values configured in the page script for input parameters. *Enable Blending* and *Enable Precise Point* are also optional settings.
- *Command Status,* region where the FB outputs are displayed to monitor FB status.

Execute button executes *CBT_MoveAbsolute* Function Block (*Figure 5.4.2*) and *Main Menu* button returns to *Main* page.



Move Absolute

Motion Parameters

Target Position

X 600 | Y 200 | Z 500

RX -180 | RY 0 | RZ 90

Robot Pose

1 0 | 2 0 | 3 0

Motion Definition

Command ID: 4

Movement Command: Line

Data Format: CPP

Motion Settings

Speed (%): 80

Acceleration Time (ms): 2500

Blending (%): 100

Command Status

○ Done

○ Busy

○ Error

Error Description :

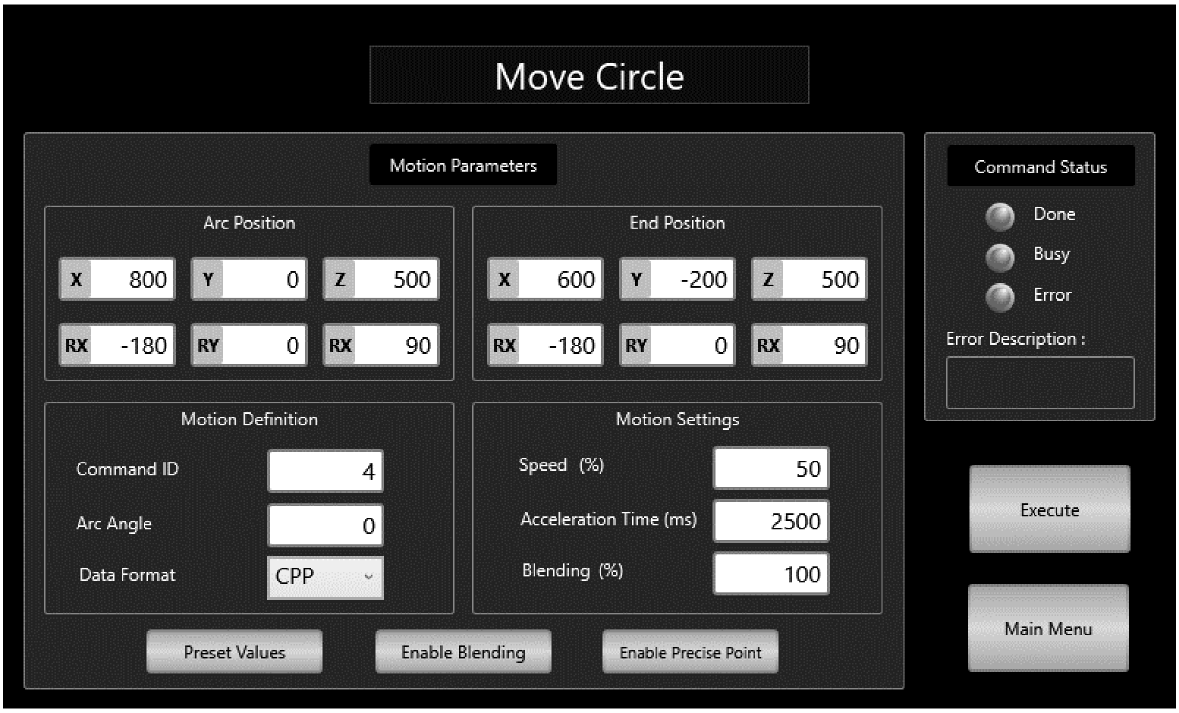
Figure 6.3.6 – pgMoveAbsolute page of HMI (source: own).

6.3.7. pgMoveCircle

This page is used as interface for *CBT_MoveCircle* Function Block. Inputs can be configured and outputs are monitored. It is composed by 2 areas:

- *Motion Parameters,* where the Arc Position and End Position (mm, ◦), Motion Command (Line or PTP) and Motion Settings must be set. *Preset Values* button is used to set predefined values configured in the page script for input parameters. *Enable Blending* and *Enable Precise Point* are also optional settings.
- *Command Status,* region where the FB outputs are displayed to monitor FB status.

Execute button executes *CBT_MoveCircle* Function Block (*Figure 5.4.4*) and *Main Menu* button returns to *Main* page.



Arc Position

X	800	Y	0	Z	500
RX	-180	RY	0	RX	90

End Position

X	600	Y	-200	Z	500
RX	-180	RY	0	RX	90

Motion Definition

Command ID 4

Arc Angle 0

Data Format CPP

Motion Settings

Speed (%) 50

Acceleration Time (ms) 2500

Blending (%) 100

Figure 6.3.7 – pgMoveCircle page of HMI (source: own).

6.3.8. pgPnP_sequence

Page used as interface for Pick and Place application program in which each of the 6 motion segments are done by *CBT_MoveAbsolute* FB. It is composed by 3 areas:

- *Main* area where the trajectory is shown in 6 discrete movements which parameters can be modified by pressing *Edit* buttons.
- *Commands,* where *Preset Values* button is used to set predefined values configured in the page script for input parameters. In this case, coordinates, and motion strategy (speed, acceleration, PTP or line motion, etc.) for each one of the 6 movements. Additionally, this page includes *Pause* and *Resume* buttons to execute *CBT_ProgramControl* Function Block (*Figure 5.4.7*). If the program is paused during robot movement, the robot will complete the interrupted movement once the program execution is restored.
- *Command Status,* region where the FB outputs are displayed to monitor FB status.

Execute button executes *CBT_MoveAbsolute* FB and *Main Menu* button returns to *Main* page.

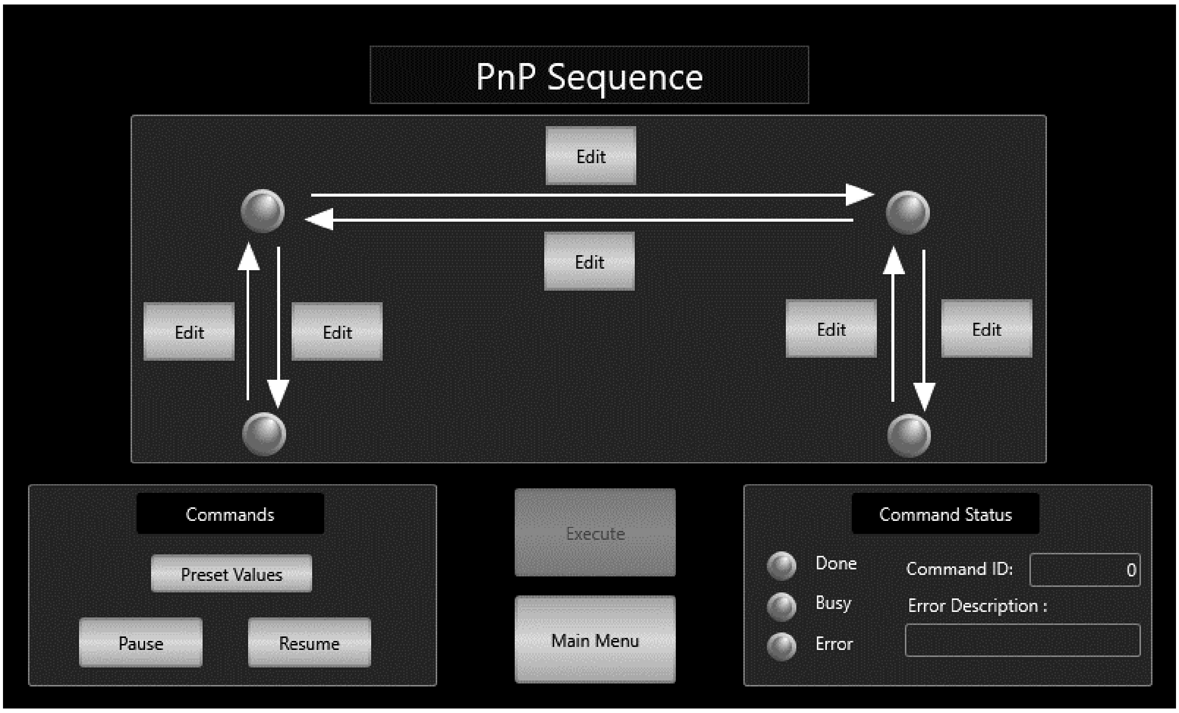
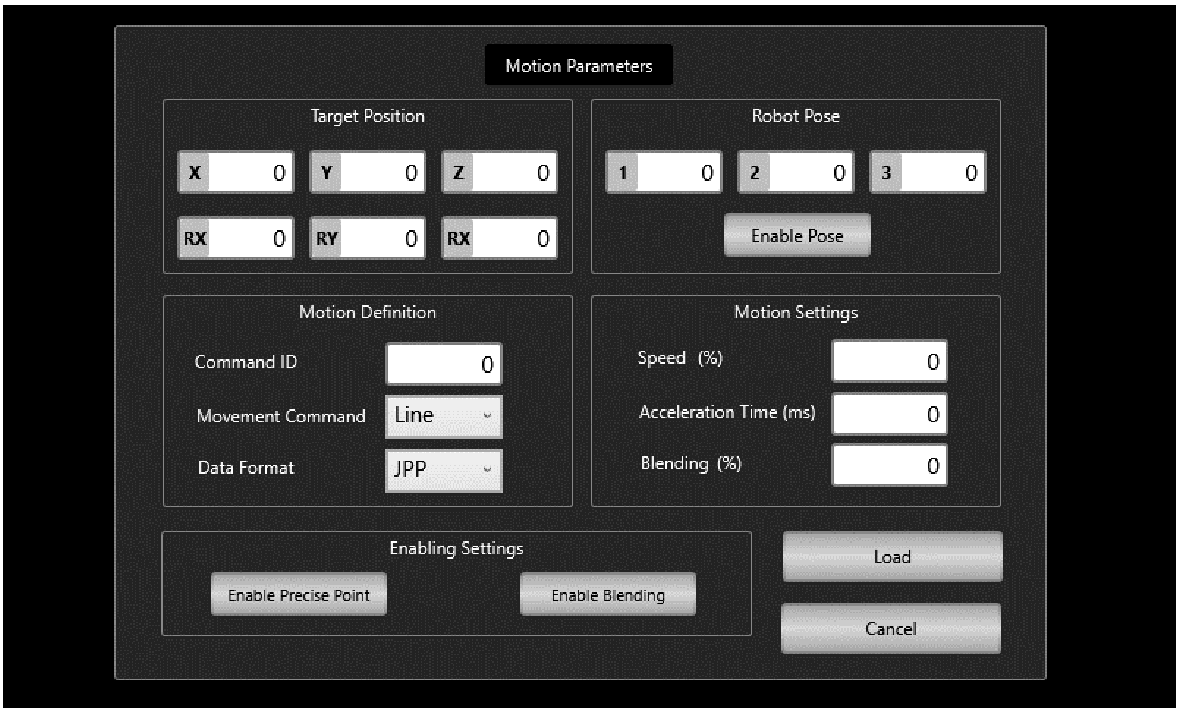


Figure 6.3.8 – pgPnP_Sequence main page of HMI (source: own).

Whether any *Edit* button is pressed, following window popups to provide to the usr the *Motion Parameters* of *CBT_MoveAbsolute*. Each one of the 6 movements has the same input parameters shown in the picture below. To select which movement the user wants to configure, just click on the *Edit* button close to the arrow representing the movement.



Motion Parameters

Target Position

X 0
Y 0
Z 0
RX 0
RY 0
RX 0

Robot Pose

1 0
2 0
3 0

Motion Definition

Command ID 0
Movement Command Line
Data Format JPP

Motion Settings

Speed (%) 0
Acceleration Time (ms) 0
Blending (%) 0

Enabling Settings

Figure 6.3.9 – pgPnP_Sequence settings page of HMI (source: own).

7. Environmental impact analysis

The product developed in this thesis is a set of Function Blocks, which usage would not directly affect to the environment. But the complete system will require electrical power thus the fact of producing that electricity would affect to the environment depending on which technology has been used by the company providing the service.

Conclusions

The first step of this project was defining the specifications of the product it was decided to create. Some of those specifications have been slightly modified during the development of the project due to the fact that some issue was found that forced to update some of those initial specifications.

The thesis is based on Function Blocks development, therefore the very first step was the search of regulations or guidelines to define the minimum requirements of a FB acceptance. Thus, a first FB sample was developed, tested, and debugged; achieving the expected appearance and behaviour. At that time, since the created FB sample was very robust in behaviour, it was decided to use that sample as template for all FBs in the library. Then, that template reduced the development time because its reuse.

Another topic was the way in which the FB identifies the robot to which the command is sent. This is done with the IP address of the robot. The idea was that the user introduces the IP address into the FB in a string variable, but there were possibilities of mistaken this. Therefore, it was required to find an intuitive and elegant solution, so it was conceived the in-out variable. It is common for all the FB and the IP address would be set only in the in/out variable definition, then just assigning this variable to the parameter, the robot is linked with the FB. As result, an intuitive and elegant way for the user to set the robot object.

Next point was the way in which the user introduces the input parameters in the FB. Since the message sent to the robot is a string variable, the first idea was to allow to the user to set the parameters as string format, what reduces the programming complexity, but this would imply a huge probability of wrong input parameters if the user does not know the proper syntax of the commands. Avoiding this undesired situation and considering as a must that the robot must receive the command message in the correct syntax, it was decided to define a datatype for each parameter. Reducing the chances of inputting wrong values and the possibility of having the robot rejecting commands because of wrong syntax. For those parameters which the value is string type, enumerated variables were created so the user just needs to select the item from a list and the FB will internally convert all input parameters in string variables and joining them in just one message. These all decisions helped to get robust, efficient, and elegant Function Blocks.

Regarding the capabilities of this Library in terms of number of robots that can be controlled, even though the development and testing have been done with just one robot connected to the communications network, the estimation is that 1 PLC would be able to control as many robots as the maximum number of TCP connections are allowed by the Omron controller used in the testing setup. Therefore, expectation is 32 cobots maximum.

In case of a second version would be release many improvements can be applied. Some of the new specifications could be:

- Include the missing functions available for cobot control from external devices.
- CommandID input parameter would be hidden for the user, and it would be managed internally by the FBs.
- In the CBT_MoveAbsolute Function Block, the RobotPose input parameter (*stCBT_RobotPose, table 5.3.2.7*) is integer datatype with only 2 possible values each one. The improvement is to define enumerated datatype to be clearer for the user:
    - 0 value would have “Righty” name.
    - 1 value would have “Lefty” name.
    - 2 value would have “Above” name.
    - 3 value would have “Below” name.
    - 4 value would have “NoFlip” name.
    - 5 value would have “Flip” name.

In general terms, it can be concluded that the Function Block Library created achieves the main target proposed at the beginning of the project: ***Provide to customers a solution to command Omron Cobots from external Omron controllers in order to make easier the integration with industrial machines.***

Budget

There are two types of expenses: Hardware and Engineering. Hardware considers the devices required for the development and for running an application where Omron cobot is controller from Omron PLC. Engineering considers the development and documentation of the Library and HMI.

The table below splits the overall cost in that two main groups.

HARDWARE

#	Reference	Description	Price
1	TM5-700	Omron TM-series Collaborative Robot (3 weeks loan)	1,406.94 €
2	NJ501-1300	Omron NJ-series Programmable Logic Controllers (PLC)	4,812.48 €
3	NA5-9W001S-V1	Omron NA-Series Touch screen HMI, 9 inch, 800x480 resolution	2,898.65 €
4	W4S1-05B	Omron Industrial Ethernet Switching Hub	320.46 €
5	S8VK-G01524	Omron Power Supply, 15 W, 24VDC	44.21 €
6	XS6W-6LSZH8SS100CM-G	Omron Ethernet cables 1 meter (4 units)	64.56 €
7	SYSMAC-SA401L-64	Omron Sysmac Studio programming software License	2,135.95 €
8	Tmflow	Omron Collaborative robot programming software	- €
			11,683.25 €
		IVA (21%)	2,453.48 €
		TOTAL HARDWARE	**14,136.73 €**

ENGINEERING

#	Reference	Description	Price
1	Dev-Prj	Development (150h x 28.95€)	4,342.50 €
2	Doc-Prj	Documentation (30h x 16.52€)	496.50 €
			4,839.00 €
		IVA (21%)	1,016.19 €
		TOTAL ENGINEERING	**5,855.19 €**

TOTAL PROJECT	**19,991.92 €**

Total cost estimation of the project is 19,991.92 €.

Bibliography

[1] PLCopen Organization. < https://plcopen.org/ >

[2] Omron Europe. *Product website* .< https://industrial.omron.eu/en/home >

[3] PLCopen, *Software Construction Guidelines. Creating PLCopen Compliant Libraries*. <https://plcopen.org/sites/default/files/downloads/creating_plcopen_compliant_function_block_libraries.pdf >

[4] International Electrotechnical Commission IEC Webstore. IEC 61131-3:2013. *Programmable controllers - Part 3: Programming languages*. <https://webstore.iec.ch/publication/4552 >

[5] AKSIT, Mehmet; TEKINERDOGAN, Bedir; BERGMANS, Lodewijk. *The six concerns for separation of concerns*. En 15th European Conference on Object-Oriented Programming, ECOOP 2001. 2001.

[6] Omron Europe. NJ_NX-series CPU Unit Built-in EtherNetIP Port User's Manual (W506-E1-24).

[7] Omron Europe. NJ_NX-series Instructions Reference Manual (W502-E1-32)

[8] Omron Europe. NJ-series CPU Unit Hardware User's Manual (W500-E1-27)

[9] Omron Europe. Sysmac Studio Version 1 Operation Manual (W504-E1-36)

[10] Omron Europe. Software Manual TMflow SW1.82 (I626-E-09)

[11] Omron Europe. TM Expression Editor and Listen Node Reference Guide (I848-E-03)

[12] Omron Europe. NA-series Connection (V119-E1-09)

[13] Omron Europe. NA-series hardware (V117-E1-10)

[14] Omron Europe. NA-series Software (V118-E1-18)

Annex A: Function Block scripts

A1. CBT_Connect

```
(* ***********************************************************************          *)
(*                              Robot StartUp FB                                    *)
(* ***********************************************************************          *)
(* --------------------------------------------------------------------------------
FB name:                                CBT_Connect
(* --------------------------------------------------------------------------------
FB version:                 V01
Library:                    CBT_Commands
SS version:                 V1.44
Date:                       June 2021
Author:                     Ruben Sanchez Boli
Description:                Establishes TCP socket connection
(* -------------------------------------------------------------------------------- *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
    IF FB_status <> 2 THEN
            //initates secuences
            FB_status:=1;
            FB_step:=10;

            //Save input at FB Execute to avoid input values modification during FB execution
            Local_PortIP        := 0;                   // Local TCP port number: Automatically assigned.
            AddressIP           := IPAddress;           // Cobot IP address
            Dest_PortIP         := IPPort;              // Cobot IP port
            ConnectionTimeOut   := Time_Out;            // Connection Timeout
    END_IF;
END_IF;

//Close connection
F_TRIG_Execute(Clk:=Execute, Q=>FalseExecute);
IF FalseExecute THEN
    FB_step:=100;
END_IF;


(* FB State Diagram control                                                         *)
(* -------------------------------------------------------------------------------- *)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive


//Sequence
CASE FB_status OF

0: (* Idle          *)
            FB_status:=0;

1: (*Reset          *)
            ToError_Sts:=FALSE;
            ToDone_Sts:=FALSE;

            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';
            FB_status:=2;

2: (*       Busy state  *)
            Done := FALSE;
            Busy := TRUE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';

            IF ToError_Sts THEN
                    FB_status:=3;                   //Error flag
            END_IF;

            IF ToDone_Sts THEN
                    FB_status:=5;                   //Done flag
            END_IF;

3: (*       Error state *)
            Done := FALSE;
            Busy := FALSE;
            Error := TRUE;
            ErrorID:=ToErrorID;
            ErrorDescrip:=ToErrorDescrip;

            IF NOT Execute THEN
                    FB_status:= 4;                                  //Returns to idle
            END_IF;

4: (*       Error Deactive if not execute       *)
            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=ToErrorID;
            ErrorDescrip:=ToErrorDescrip;

5: (*       Done state *)
            Done := TRUE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';

            IF NOT Execute THEN
                    FB_status:= 6;                                  //Returns to idle
            END_IF;

6: (*               Done deactive if not Execute            *)
            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';

END_CASE ;


(* FB Algorithm                                 *)
(* ------------------------------------------   *)

IF Busy THEN

    CASE FB_step OF
    0: (*   Idle                *)
                    FB_step:=0;

    10: (*  Initialization              *)
                    //InternalStep variables
                    TCP_Connect_Exe                 :=FALSE;
```

```
                TCP_Clear_Buffer_Exe        :=FALSE;
                Get_TCP_Status_Exe          :=FALSE;
                TCP_Send_Exe                                :=FALSE;
                TCP_Rcv_Exe                                 :=FALSE;
                TCP_Close_Exe                               :=FALSE;
                TON_TimeOut_Exe             :=FALSE;
                ToErrorID                                           :=0;
                ToErrorDescrip                              :=' ';

                //FB Outputs
                CBT.WaitingReturn:=FALSE;
                CBT.Connected:=FALSE;
                Connected:=CBT.Connected;
                LastConnTime        := SecToDt(0);
                LastLostTime        := SecToDt(0);
                FirstConnection:=FALSE;

                FB_step:=15;

    15:(*       Check Timeout value             *)

                IF (ConnectionTimeOut<10) OR (ConnectionTimeOut>300) THEN
                        ToErrorID:=16#15;
                        ToErrorDescrip:='Timeout range: 10~300 seconds';
                        TCP_Connect_Exe:=FALSE;
                        ToError_Sts :=TRUE;
                else
                        FB_step:=20;
                END_IF;



    20:(*       Request a connection            *)
                TCP_Connect_Exe := TRUE;
                TON_TimeOut_Exe:=TRUE;

                IF TCP_Connect.Done THEN
                        TCP_Connect_Exe:=FALSE;
                        TON_TimeOut_Exe:=FALSE;
                        FB_step:=30;
                END_IF;

                IF TCP_Connect.Error THEN
                        ToErrorID:=16#20;
                        ToErrorDescrip:='Connection Error';
                        TCP_Connect_Exe:=FALSE;
                        ToError_Sts :=TRUE;
                END_IF;

                IF TimeOut THEN
                        ToErrorID:=16#21;
                        ToErrorDescrip:='TimeOut Error';
                        TCP_Connect_Exe:=FALSE;
                        TON_TimeOut_Exe:=FALSE;
                        ToError_Sts :=TRUE;
                END_IF;

    30: (*      Clear receive buffer            *)
                TCP_Clear_Buffer_Exe:=TRUE;

                IF TCP_Clear_Buffer.Done THEN
                        TCP_Clear_Buffer_Exe:=FALSE;
                        FB_step:=40;
                END_IF;

                IF TCP_Clear_Buffer.Error THEN
                        ToErrorID:=16#30;
                        ToErrorDescrip:='Clear Buffer Error';
                        TCP_Clear_Buffer_Exe:=FALSE;
                        ToError_Sts :=TRUE;
                END_IF;


    40: (*    Request reading status            *)
                Get_TCP_Status_Exe:=TRUE;

                IF Get_TCP_Status.Done THEN
                    Get_TCP_Status_Exe:=FALSE;
                    FB_step:=50;
                END_IF;

                IF Get_TCP_Status.Error THEN
                    ToErrorID:=16#40;
                    ToErrorDescrip:='Get TCP Status Error';
                    Get_TCP_Status_Exe:=FALSE;
                    ToError_Sts := TRUE;
                END_IF;


    50: (*    Request sending data              *)
                // Converts command to byte array

                str_CheckListen:='$$TMSTA,2,00,*41$R$L';
                // $$TMSTA       --> Communication package acquiring status // Sysmac requires $$, the 1st is sysmac systax to recognize the 2nd $ as character
                // 2         --> Indicates the length of 00 is 2 bytes
                // 00        --> Indicates if cobot is in external script control mode or not (is Cobot in Listen node or not)
                // 41        --> Checksum
                //$R         --> $R in sysmac syntax is \R in ASCII (carriage)
                //$L         --> $L in sysmac syntax is \L in ASCII (enter)

                TCP_Send_Size:=LEN(str_CheckListen);
                ToAryByte(In:=str_CheckListen, Order:=_eBYTE_ORDER#_LOW_HIGH, AryOut:=TCP_Send_Data[0]);


                TCP_Send_Exe      :=TRUE;

                IF TCP_Send.Done THEN
                    TCP_Send_Exe:=FALSE;
                    FB_step:=60;
                END_IF;

                IF TCP_Send.Error THEN
                    ToErrorID:=16#50;
                    ToErrorDescrip:='TCP Send Error';
                    TCP_Send_Exe:=FALSE;
                    ToError_Sts := TRUE;
                END_IF;

    60: (*    Request receiving data                    *)

                TCP_Rcv_TimeOut:=0;           //0: No timeouts
                TCP_Rcv_Size:=256;            //Set number of bytes to read from the receive buffer
                StringOfReceivedData:='';     //Clear the variable where Receive data array is compiled

                TCP_Rcv_Exe       :=TRUE;

                IF TCP_Rcv.Done THEN
                    StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_RcvSize);
                    //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
                    TCP_Rcv_Exe:=FALSE;
                    FB_step:=70;
                END_IF;

                IF TCP_Rcv.Error THEN
                    ToErrorID:=16#60;
                    ToErrorDescrip:='TCP Receive Error';
                    TCP_Rcv_Exe:=FALSE;
                    ToError_Sts := TRUE;
                END_IF;


    70: (*      Check received data                 *)

                    IF FIND(StringOfReceivedData,'true') <> 0 THEN
                        FB_step:=80;
                    ELSIF FIND(StringOfReceivedData,'false') <> 0 THEN
                        ToErrorID:=16#70;
                        ToErrorDescrip:='Device is not a Cobot';
                        ToError_Sts := TRUE;
                    END_IF;


    80: (*      Continuously checking status                *)

                    Get_TCP_Status_Exe:=TRUE;

                    IF Get_TCP_Status.Done THEN
                        CBT.Connected:=TRUE;
                        Connected:=CBT.Connected;
                        Get_TCP_Status_Exe:=FALSE;
                        FB_step:=90;
                    END_IF;

                    IF Get_TCP_Status.Error THEN
                        ToErrorID:=16#80;
                        ToErrorDescrip:='Get TCP Status Error';
                        Get_TCP_Status_Exe:=FALSE;
                        ToError_Sts := TRUE;
                    END_IF;


    90: (*      Check status each time                      *)

                    IF NOT(FirstConnection) THEN //Updates output just once
                        LastConnTime:=GetTime();
                        FirstConnection:=TRUE;
                    END_IF;

                    Timer_1_Enable := TRUE ;

                    IF Timer_1_Done THEN
                        Timer_1_Enable:=FALSE;
                        FB_step:=FB_step-10;                    //Continuously checking status
                    END_IF;

                    IF SocketStatus =_CLOSE_WAIT THEN
                        LastLostTime:=GetTime();
                        ToErrorID:=16#90;
                        ToErrorDescrip := 'Socket disconnected';
                        CBT.WaitingReturn:=FALSE;
                        CBT.Connected:=FALSE;
                        Connected:=CBT.Connected;
                        Get_TCP_Status_Exe:=FALSE;
                        ToError_Sts := TRUE;
                    END_IF;


    100: (*     Request closing                     *)

                    CBT.WaitingReturn:=FALSE;
                    CBT.Connected:=FALSE;
                    Connected:=CBT.Connected;
                    TCP_Close_Exe:=TRUE;
                    LastLostTime:=GetTime();

                    IF TCP_Close.Done THEN
                        TCP_Close_Exe:=FALSE;
                        FB_step:=110;
                    END_IF;

```

```
                        IF TCP_Close.Error THEN
                            ToErrorID:=16#100;
                            ToErrorDescrip:='TCP Close Error';
                            TCP_Close_Exe:=FALSE;
                            ToError_Sts := TRUE;
                        END_IF;


    110: (*     End Execution                   *)
            ToDone_Sts:=TRUE;

    END_CASE ;

END_IF;


(* Function Bolcks                                              *)
(* ---------------------------------------------------------------- *)

TCP_Connect(
    Execute:=TCP_Connect_Exe,
    SrcTcpPort:=Local_PortIP,
    DstAdr:=AddressIP,
    DstTcpPort:=Dest_PortIP,
    //Done=>, Busy=>, Error=>, ErrorID=>,
    Socket=>CBT.Socket);

TCP_Clear_Buffer(
    Execute:=TCP_Clear_Buffer_Exe,
    Socket:=CBT.Socket
    //Done=>, Busy=>, Error=>, ErrorID=>
    );

Get_TCP_Status(
    Execute:=Get_TCP_Status_Exe,
    Socket:=CBT.Socket,
    //Done=>, Busy=>, Error=>, ErrorID=>,
    TcpStatus=>SocketStatus
    //DatRcvFlag=>
    );

TCP_Send(
    Execute:=TCP_Send_Exe,
    Socket:=CBT.Socket,
    SendDat:=TCP_Send_Data[0],
    Size:=TCP_Send_Size
    //Done=>, Busy=>, Error=>, ErrorID=>
    );

TCP_Rcv(
    Execute:=TCP_Rcv_Exe,
    Socket:=CBT.Socket,
    TimeOut:=TCP_Rcv_TimeOut,
    Size:=TCP_Rcv_Size,
    RcvDat:=TCP_Rcv_Data[0],
    //Done=>, Busy=>, Error=>, ErrorID=>,
    RcvSize=>TCP_Rcv_RcvSize);

TCP_Close(
    Execute:=TCP_Close_Exe,
    Socket:=CBT.Socket
    //Done=>, Busy=>, Error=>, ErrorID=>
    );

TON_TimeOut(In:=TON_TimeOut_Exe, PT:=NanoSecToTime(ConnectionTimeOut*1000000000), Q=>TimeOut);
Timer_1(In:=Timer_1_Enable, PT:=T#1000ms, Q=>Timer_1_Done);
```

A2. CBT_MoveAbsolute

```
(* ********************************************************************************      *)
(*                                          Robot StartUp FB                              *)
(* ********************************************************************************      *)
(* ------------------------------------------------------------------------------
FB name:                                    CBT_MoveAbsolute
(* ------------------------------------------------------------------------------
FB version:                   V01
Library:                      CBT_Commands
SS version:                   V1.44
Date:                         June 2021
Author:                       Ruben Sanchez Boli
Description:                  Sends motion commands to absolute positions
                              - PTP() -> Syntax 4 in expression editor
                              - Line() -> Syntax 2 in expression editor
(* ------------------------------------------------------------------------------ *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
     IF FB_status <> 2 THEN                          //FBstatus_Step different than 2 - Busy (avoids re-execution)
                  iParameters:=Parameters;           //Copy internal variable
                  FB_status:=1;                      //Initates State Diagram control
                  FB_step:=10;                       //Initates algorithm sequence
     END_IF;
END_IF;

//If robot is not conneted when FB is executed or robot suddenly disconnects (e.g. EStop is triggered during FB execution)
IF Execute AND NOT(CBT.Connected) THEN
     ToError_Sts := TRUE;
     ToErrorID:=16#99;
     ToErrorDescrip:='Robot not connected';
END_IF;

(* FB State Diagram control                                                       *)
(* ------------------------------------------------------------------------------ *)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive


//Sequence
CASE FB_status OF

0: (*                         Idle                                  *)
                  FB_status:=0;

1: (*                         Reset                    *)
                  ToError_Sts:=FALSE;
                  ToDone_Sts:=FALSE;

                  Done := FALSE;
                  Busy := FALSE;
                  Error := FALSE;
                  ErrorID:=0;
                  ErrorDescrip:=' ';
                  FB_status:=2;

2: (*             Busy state  *)
                  Done := FALSE;
                  Busy := TRUE;
                  Error := FALSE;
```

```
                ErrorID:=0;
                ErrorDescrip:=' ';

                IF ToError_Sts THEN
                        FB_status:=3;                   //"Error" flag
                END_IF;

                IF ToDone_Sts THEN
                        FB_status:=5;                   //"Done" flag
                END_IF;


3: (*           Error state *)
                Done := FALSE;
                Busy := FALSE;
                Error := TRUE;
                ErrorID:=ToErrorID;
                ErrorDescrip:=ToErrorDescrip;

                IF NOT Execute THEN
                        FB_status:= 4;                  //Returns to idle
                END_IF;

4: (*           Error Deactive if not execute       *)
                Done := FALSE;
                Busy := FALSE;
                Error := FALSE;
                ErrorID:=ToErrorID;
                ErrorDescrip:=ToErrorDescrip;


5: (*           Done state *)
                Done := TRUE;
                Busy := FALSE;
                Error := FALSE;
                ErrorID:=0;
                ErrorDescrip:=' ';

                IF NOT Execute THEN
                        FB_status:= 6;                  //Returns to idle
                END_IF;

6: (*           Done deactive if not Execute            *)
                Done := FALSE;
                Busy := FALSE;
                Error := FALSE;
                ErrorID:=0;
                ErrorDescrip:=' ';

END_CASE ;



(* FB Algorithm                                         *)
(* ---------------------------------------------------------------------  *)

IF Busy THEN                    //FB State Diagram control      in "Busy" state

    CASE FB_step OF
    0: (*       Idle                    *)
                        FB_step:=0;

    10: (*      Initialization                  *)

                        TCP_Clear_Buffer_Exe            :=FALSE;
                        TCP_Send_Exe                                    :=FALSE;
                        TCP_Rcv_Exe                                     :=FALSE;
                        ToErrorID                                               :=0;
                        ToErrorDescrip                                  :=' ';
                        CmdID_Ack                                               :=Parameters.CommandID;

```

```
        IF CBT.Connected and NOT(CBT.WaitingReturn) THEN                    //If robot is connected
and "released" (not occupied by other FB)
            FB_step:=20;
        ELSIF NOT(CBT.Connected) THEN
            //Robot not connected
            ToError_Sts := TRUE;
            ToErrorID:=16#10;
            ToErrorDescrip:='Robot not connected';
        ELSIF (CBT.WaitingReturn) THEN
                //Robot is "occupied" by other FB
            ToError_Sts := TRUE;
            ToErrorID:=16#11;
            ToErrorDescrip:='Other FB is under execution';
        END_IF;


    20: (*    Clear receive buffer            *)
        TCP_Clear_Buffer_Exe:=TRUE;

        IF TCP_Clear_Buffer.Done THEN
            TCP_Clear_Buffer_Exe:=FALSE;
            FB_step:=25;
        END_IF;

        IF TCP_Clear_Buffer.Error THEN
            TCP_Clear_Buffer_Exe:=FALSE;
            ToError_Sts := TRUE;
            ToErrorID:=16#20;
            ToErrorDescrip:='Clear buffer error';
        END_IF;


    25: (*    Do not exceed the seetable range *)

        IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line
THEN
            IF iParameters.Speed > 4500 THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#25;
                ToErrorDescrip:='Speed range in Line: 0~4500 mm/s';
            END_IF;
        END_IF;

        IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP
THEN
            IF iParameters.Speed > 100 THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#26;
                ToErrorDescrip:='Speed range in PTP: 0~100 %';
            END_IF;
        END_IF;

        IF (iParameters.AccelTime < 150) OR (iParameters.AccelTime > 9999) THEN
            ToError_Sts := TRUE;
            ToErrorID:=16#27;
            ToErrorDescrip:='Acceleration Time range is: 150~9999 ms';
        END_IF;

        IF (iParameters.CommandID < 2) OR (iParameters.CommandID > 9) THEN
            ToError_Sts := TRUE;
            ToErrorID:=16#28;
            ToErrorDescrip:='Command ID range is: 2~9';
        END_IF;

        IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line
THEN
            IF iParameters.RobotPoseEnable = true THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#29;
                ToErrorDescrip:='Robot Pose must be disable for Line command';
                END_IF;
            END_IF;

            FB_step:=30;


    30: (*  Do not exceed the seetable range *)

            IF iParameters.RobotPoseEnable THEN
                IF (iParameters.RobotPose.Pose1 < 0) OR (iParameters.RobotPose.Pose1 > 1) THEN
                    ToError_Sts := TRUE;
                    ToErrorID:=16#30;
                    ToErrorDescrip:='Pose1 range is 0~1';
                END_IF;
            END_IF;

            IF iParameters.RobotPoseEnable THEN
                IF (iParameters.RobotPose.Pose2 < 2) OR (iParameters.RobotPose.Pose2 > 3) THEN
                    ToError_Sts := TRUE;
                    ToErrorID:=16#31;
                    ToErrorDescrip:='Pose2 range is 2~3';
                END_IF;
            END_IF;

            IF iParameters.RobotPoseEnable THEN
                IF (iParameters.RobotPose.Pose3 < 4) OR (iParameters.RobotPose.Pose3 > 5) THEN
                    ToError_Sts := TRUE;
                    ToErrorID:=16#32;
                    ToErrorDescrip:='Pose3 range is 4~5';
                END_IF;
            END_IF;

            FB_step:=35;

    35: (*  iParameters conversion to string *)

            str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

            // iParameters.MovementCommand
            IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line THEN
                str_MovementCommand:='Line';
            END_IF;

            IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP THEN
                str_MovementCommand:='PTP';
            END_IF;

            //iParameters.DataFormat
            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CAP_Abs THEN
                str_DataFormat:='CAP';
            END_IF;

            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CAR_Abs THEN
                str_DataFormat:='CAR';
            END_IF;

            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CPP_Abs THEN
                str_DataFormat:='CPP';
            END_IF;

            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CPR_Abs THEN
                str_DataFormat:='CPR';
            END_IF;

```

```
IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#JPP_Abs THEN
    str_DataFormat:='JPP';
END_IF;

//Conversion of REAL variables to a text string with the specified format
str_TP_X := RealToFormatString(In:=iParameters.TargetPosition.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
str_TP_Y := RealToFormatString(In:=iParameters.TargetPosition.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
str_TP_Z := RealToFormatString(In:=iParameters.TargetPosition.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
str_TP_RX := RealToFormatString(In:=iParameters.TargetPosition.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
str_TP_RY := RealToFormatString(In:=iParameters.TargetPosition.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
str_TP_RZ := RealToFormatString(In:=iParameters.TargetPosition.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
str_TP1:=CONCAT(str_TP_X,',',str_TP_Y,',',str_TP_Z);
str_TP2:=CONCAT(str_TP_RX,',',str_TP_RY,',',str_TP_RZ);
str_TargetPosition:=CONCAT(str_TP1,',',str_TP2);

//Conversion of integer to text string
str_Speed:=UINT_TO_STRING(iParameters.Speed);
str_AccelTime:=UINT_TO_STRING(iParameters.AccelTime);

IF iParameters.BlendingEnable THEN
    str_BlendingValue:=UINT_TO_STRING(iParameters.BlendingValue);
ELSE
    str_BlendingValue:='0';
END_IF;

//Conversion of bool to text string
IF iParameters.PrecisePositioning THEN
    str_PrecisePositioning:='true';
ELSE
    str_PrecisePositioning:='false';
END_IF;

//iParameters.RobotPose
str_RobotPose:=CONCAT(     UINT_TO_STRING(iParameters.RobotPose.Pose1),',',
                           UINT_TO_STRING(iParameters.RobotPose.Pose2),',',
                           UINT_TO_STRING(iParameters.RobotPose.Pose3));


//Verify if dataformat input is valid for MovementCommand selection
IF (str_MovementCommand='PTP') THEN
    IF (str_DataFormat='JPP') OR (str_DataFormat='CPP') THEN
        FB_step:=70;
    ELSE
        ToError_Sts := TRUE;
        ToErrorID:=16#35;
        ToErrorDescrip:='DataFormat invalid for PTP command';
    END_IF;
END_IF;

IF (str_MovementCommand='Line') THEN
    IF (str_DataFormat='CPP') OR
        (str_DataFormat='CPR') OR
        (str_DataFormat='CAP') OR
        (str_DataFormat='CAR') THEN
        FB_step:=70;
    ELSE
        ToError_Sts := TRUE;
        ToErrorID:=16#36;
        ToErrorDescrip:='DataFormat invalid for Line command';
    END_IF;
END_IF;

```

```
                FB_step:=40;

        40: (*  Frame buildingfor CheckSum calculation *)

                        Header:='TMSCT';            //Header required by robot controller to receive external scripts

                        //Creates the script command with all the input parameters
                        IF NOT(iParameters.RobotPoseEnable) then
                                Script_Command:=CONCAT(   CONCAT(str_MovementCommand,'(',str_DataFormat,','),

                                CONCAT(str_TargetPosition,',',str_Speed,','),

                                CONCAT(str_AccelTime,',',str_BlendingValue,','),

                                CONCAT(str_PrecisePositioning,')') );
                        ELSE
                                Script_Command:=CONCAT(   CONCAT(str_MovementCommand,'(',str_DataFormat,','),

                                CONCAT(str_TargetPosition,',',str_Speed,','),

                                CONCAT(str_AccelTime,',',str_BlendingValue,','),

                                CONCAT(str_PrecisePositioning,',',str_RobotPose,')') );
                        END_IF;


                        //When no blending motion, acknowledgement with QueueTag()
                        IF NOT(iParameters.BlendingEnable) THEN
                                Script_Command:=CONCAT(Script_Command,'$R$LQueueTag(',str_CommandID,',0)');
                        END_IF;

                        //Exits Listen Node in TMflow with ScriptExit()
                        IF iParameters.ExitNode THEN
                                Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
                        END_IF;

                        // CheckSum calculation
                        str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));

                //Length for DATA command
                        str_Checksum_Calc :=
  CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');             //Checksum
  includes: Header, Length for DATA command, str_CommandID and Script_Command


        Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);

                        IF Checksum_Length>0 THEN
                                FB_step:=50;
                        ELSE
                                ToError_Sts := TRUE;
                                ToErrorID:=16#40;
                                ToErrorDescrip:='Checksum length not valid';
                        END_IF;



        50: (*  Frame buildingto send command to TMflow *)

                        //CheckSum calculation by XOR operation
                        FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
                                Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
                        END_FOR;

                        //Converts checksum byte value to text string
                        str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
                        str_SendFrame :=
  CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');

                        //Finds the number of characters in the string to be sent in the frame
                        Length:=LEN(str_SendFrame);
```

```
            Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

            IF Long>0 THEN
                FB_step:=60;
            ELSE
                ToError_Sts := TRUE;
                ToErrorID:=16#50;
                ToErrorDescrip:='Final frame building is error end';
            END_IF;


    60: (*      Send command                    *)

            //Finds the number of characters in the string to be sent in the frame
            TCP_Send_Size:=LEN(str_SendFrame);
            ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH, AryOut:=TCP_Send_Data[0]);

            TCP_Send_Exe        :=TRUE;

            IF TCP_Send.Done THEN
                CBT.WaitingReturn:=TRUE;                //Flag to set the robot in "busy" state to avoid other FB to be executed
                TCP_Send_Exe:=FALSE;
                FB_step:=70;
            END_IF;

            IF TCP_Send.Error THEN
                TCP_Send_Exe:=FALSE;
                ToError_Sts := TRUE;
                ToErrorID:=16#60;
                ToErrorDescrip:='TCP send error';
            END_IF;


    70: (*      Request receiving data                    *)

            TCP_Rcv_TimeOut:=0;                                     //0: No timeouts
            TCP_Rcv_Size:=256;                                      //Set number of bytes to read from the receive buffer
            StringOfReceivedData:='';                   //Clear the variable where Receive data array is compiled

            TCP_Rcv_Exe :=TRUE;

            IF TCP_Rcv.Done THEN
                StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
                                                        //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
                TCP_Rcv_Exe:=FALSE;
                FB_step:=80;
            END_IF;

            IF TCP_Rcv.Error THEN
                TCP_Rcv_Exe:=FALSE;
                ToError_Sts := TRUE;
                ToErrorID:=16#70;
                ToErrorDescrip:='TCP receive error';
            END_IF;


    80: (*      Check acknowledgement Command accepted                    *)

        IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN                 //Message no vaild
            FB_step:=70;
        END_IF;

        IF FIND(StringOfReceivedData,'TMSCT') <> 0 THEN
            IF FIND(StringOfReceivedData,'OK') <> 0 THEN                        //Command accepted
                IF NOT(iParameters.BlendingEnable) THEN
```

```
                    FB_step:=90;
                    //"Done" signal waits until motion is finished (no blending)
                ELSE
                    CBT.WaitingReturn:=FALSE;
                    //Flag to set the robot in "released" state to avoid other FB to be executed
                    CmdID_Ack:=STRING_TO_UINT(str_CommandID);  //Output the Command ID when ack is done
                    FB_step:=200;
                    //"Done" signal does not wait until  motion ends (allows blending)
                END_IF;
            ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
                CBT.WaitingReturn:=FALSE;
                    //Flag to set the robot in "released" state to avoid other FB to be executed
                ToError_Sts := TRUE;
                ToErrorID:=16#80;
                ToErrorDescrip:=' Command rejected';
            END_IF;
        END_IF;

    90: (*  Clear receive buffer            *)
            TCP_Clear_Buffer_Exe:=TRUE;

            IF TCP_Clear_Buffer.Done THEN
                TCP_Clear_Buffer_Exe:=FALSE;
                FB_step:=100;
            END_IF;

            IF TCP_Clear_Buffer.Error THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#90;
                ToErrorDescrip:='Clear buffer error';
            END_IF;


    100: (*  Request receiving data                *)

            CmdID_Ack:=STRING_TO_UINT(str_CommandID);           //Output the Command ID when ack is done

            TCP_Rcv_TimeOut:=0;                                 //0: No timeouts
            TCP_Rcv_Size:=256;                                  //Set number of bytes to read from the receive buffer
            StringOfReceivedData:='';               //Clear the variable where Receive data array is compiled

            TCP_Rcv_Exe :=TRUE;

            IF TCP_Rcv.Done THEN
                StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);

            //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
                TCP_Rcv_Exe:=FALSE;
                FB_step:=110;
            END_IF;

            IF TCP_Rcv.Error THEN
                TCP_Rcv_Exe:=FALSE;
                ToError_Sts := TRUE;
                ToErrorID:=16#100;
                ToErrorDescrip:='TCP receive error';
            END_IF;


    110: (*  Check acknowledgement Motion Completed          *)

            IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN
    //QueueTag acknowledgement
                IF FIND(StringOfReceivedData,str_CommandID) <> 0 THEN             //QueueTag for the last motion command
                    IF FIND(StringOfReceivedData,'true') <> 0 THEN
        //Motion finished
```

```
                        CBT.WaitingReturn:=FALSE;                               //Flag to set the robot in "released" state to avoid other FB to be executed
                        CmdID_Ack:=STRING_TO_UINT(str_CommandID);               //Output the Command ID when ack is done
                        FB_step:=200;
                    ELSIF FIND(StringOfReceivedData,'false') <> 0 THEN
                        CBT.WaitingReturn:=FALSE;                               //Flag to set the robot in "released" state to avoid other FB to be executed
                        ToError_Sts := TRUE;
                        ToErrorID:=16#110;
                        ToErrorDescrip:=' Motion failed';
                    END_IF;
                ELSE    //no str_CommandID
                    FB_step:=100;
                END_IF;
            ELSE            //no TMSTA
                FB_step:=100;
            END_IF;


  200: (*       End Execution       *)
        ToDone_Sts:=TRUE;

  END_CASE ;

END_IF;




(* Function Bolcks                                                                  *)
(* -------------------------------------------------------------------  *)


TCP_Clear_Buffer(
  Execute:=TCP_Clear_Buffer_Exe,
  Socket:=CBT.Socket
  //Done=>, Busy=>, Error=>, ErrorID=>
  );

TCP_Send(
  Execute:=TCP_Send_Exe,
  Socket:=CBT.Socket,
  SendDat:=TCP_Send_Data[0],
  Size:=TCP_Send_Size
  //Done=>, Busy=>, Error=>, ErrorID=>
  );

TCP_Rcv(
  Execute:=TCP_Rcv_Exe,
  Socket:=CBT.Socket,
  TimeOut:=TCP_Rcv_TimeOut,
  Size:=TCP_Rcv_Size,
  RcvDat:=TCP_Rcv_Data[0],
  //Done=>, Busy=>, Error=>, ErrorID=>,
  RcvSize=>TCP_Rcv_RcvSize);
```

A3. CBT_MoveRelative

```
(* ***************************************************************************     *)
(*                      Robot StartUp FB                                     *)
(* ***************************************************************************     *)
(* -----------------------------------------------------------------------------
FB name:                          CBT_MoveRelative
(* -----------------------------------------------------------------------------
FB version:             V01
Library:                          CBT_Commands
SS version:             V1.44
Date:                             June 2021
Author:                           Ruben Sanchez Boli
Description:            Sends motion commands to relative positions
                        - move_PTP() -> Syntax 2 in expression editor
                        - move_Line() -> Syntax 2 in expression editor
(* ----------------------------------------------------------------------------- *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
    IF FB_status <> 2 THEN              //FBstatus_Step different than 2 - Busy (avoids re-execution)
            iParameters:=Parameters;    //Copy internal variable
            FB_status:=1;               //initates secuences
            FB_step:=10;                //Initates algorithm sequence
    END_IF;
END_IF;

//In case of EStop is triggered during FB execution
IF Execute AND NOT(CBT.Connected) THEN
    ToError_Sts := TRUE;
    ToErrorID:=16#99;
    ToErrorDescrip:='Robot not connected';
END_IF;

(* FB State Diagram control                                          *)
(* ------------------------------------------------------------------------------  *)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive


//Sequence
CASE FB_status OF

0: (*                   Idle                        *)
            FB_status:=0;

1: (*                   Reset               *)
            ToError_Sts:=FALSE;
            ToDone_Sts:=FALSE;

            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';
            FB_status:=2;

2: (*       Busy state  *)
            Done := FALSE;
            Busy := TRUE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';

            IF ToError_Sts THEN
                    FB_status:=3;               //Error flag
            END_IF;

            IF ToDone_Sts THEN
                    FB_status:=5;               //Done flag
            END_IF;

3: (*       Error state *)
            Done := FALSE;
            Busy := FALSE;
            Error := TRUE;
            ErrorID:=ToErrorID;
            ErrorDescrip:=ToErrorDescrip;

            IF NOT Execute THEN
                    FB_status:= 4;                              //Returns to idle
            END_IF;

4: (*       Error Deactive if not execute    *)
            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=ToErrorID;
            ErrorDescrip:=ToErrorDescrip;

5: (*       Done state *)
            Done := TRUE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';

            IF NOT Execute THEN
                    FB_status:= 6;                              //Returns to idle
            END_IF;

6: (*               Done deactive if not Execute           *)
            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';

END_CASE ;



(* FB Algorithm                                                  *)
(* ------------------------------------------------------------------   *)

IF Busy THEN

    CASE FB_step OF
    0: (*   Idle            *)
                    FB_step:=0;

    10: (*  Initialization          *)
                    //InternalStep variables
                    TCP_Clear_Buffer_Exe        :=FALSE;
                    TCP_Send_Exe                                :=FALSE;
                    TCP_Rcv_Exe                                 :=FALSE;
                    ToErrorID                                           :=0;
                    ToErrorDescrip                              :=' ';
                    CmdID_Ack                                           :=Parameters.CommandID;

                    IF CBT.Connected and NOT(CBT.WaitingReturn) THEN
                            FB_step:=20;
            ELSIF NOT(CBT.Connected) THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#10;
                ToErrorDescrip:='Robot not connected';
            ELSIF (CBT.WaitingReturn) THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#11;
                ToErrorDescrip:='Other FB is under execution';
            END_IF;


    20: (*      Clear receive buffer            *)
            TCP_Clear_Buffer_Exe:=TRUE;

            IF TCP_Clear_Buffer.Done THEN
                TCP_Clear_Buffer_Exe:=FALSE;
                FB_step:=25;
            END_IF;

            IF TCP_Clear_Buffer.Error THEN
                TCP_Clear_Buffer_Exe:=FALSE;
                ToError_Sts := TRUE;
                ToErrorID:=16#20;
                ToErrorDescrip:='Clear buffer error';
            END_IF;


    25: (*      Do not exceed the seetable range *)

            IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line THEN
                IF iParameters.Speed > 4500 THEN
                    ToError_Sts := TRUE;
                    ToErrorID:=16#25;
                    ToErrorDescrip:='Speed range in Line: 0~4500 mm/s';
                END_IF;
            END_IF;

            IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP THEN
                IF iParameters.Speed > 100 THEN
                    ToError_Sts := TRUE;
                    ToErrorID:=16#26;
                    ToErrorDescrip:='Speed range in PTP: 0~100 %';
                END_IF;
            END_IF;

            IF (iParameters.AccelTime < 150) OR (iParameters.AccelTime > 9999) THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#27;
                ToErrorDescrip:='Acceleration Time range is: 150~9999 ms';
            END_IF;

            IF (iParameters.CommandID < 2) OR (iParameters.CommandID > 9) THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#28;
                ToErrorDescrip:='Command ID range is: 2~9';
            END_IF;

            FB_step:=30;


    30: (*  iParameters conversion to string *)

            str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

            // iParameters.MovementCommand
            IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line THEN
                str_MovementCommand:='Move_Line';
            END_IF;

            IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP THEN
                str_MovementCommand:='Move_PTP';
            END_IF;

            //iParameters.DataFormat
            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#JPP_Rel THEN
                str_DataFormat:='JPP';
            END_IF;

            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CAP_Rel THEN
                str_DataFormat:='CAP';
            END_IF;

            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CAR_Rel THEN
                str_DataFormat:='CAR';
            END_IF;

            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CPP_Rel THEN
                str_DataFormat:='CPP';
            END_IF;

            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CPR_Rel THEN
                str_DataFormat:='CPR';
            END_IF;

            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TAP_Rel THEN
                str_DataFormat:='TAP';
            END_IF;

            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TAR_Rel THEN
                str_DataFormat:='TAR';
            END_IF;

            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TPP_Rel THEN
                str_DataFormat:='TPP';
            END_IF;

            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TPR_Rel THEN
                str_DataFormat:='TPR';
            END_IF;


            str_TP_X := RealToFormatString(In:=iParameters.TargetPosition.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_TP_Y := RealToFormatString(In:=iParameters.TargetPosition.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_TP_Z := RealToFormatString(In:=iParameters.TargetPosition.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_TP_RX := RealToFormatString(In:=iParameters.TargetPosition.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_TP_RY := RealToFormatString(In:=iParameters.TargetPosition.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_TP_RZ := RealToFormatString(In:=iParameters.TargetPosition.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_TP1:=CONCAT(str_TP_X,',',str_TP_Y,',',str_TP_Z);
            str_TP2:=CONCAT(str_TP_RX,',',str_TP_RY,',',str_TP_RZ);
            str_TargetPosition:=CONCAT(str_TP1,',',str_TP2);

            str_Speed:=UINT_TO_STRING(iParameters.Speed);
            str_AccelTime:=UINT_TO_STRING(iParameters.AccelTime);

            IF iParameters.BlendingEnable THEN
                str_BlendingValue:=UINT_TO_STRING(iParameters.BlendingValue);
            ELSE
                str_BlendingValue:='0';
            END_IF;

            IF iParameters.PrecisePositioning THEN
                str_PrecisePositioning:='true';
            ELSE
                str_PrecisePositioning:='false';
            END_IF;

            IF (str_MovementCommand='Move_PTP') THEN
                IF (str_DataFormat='JPP') OR
                    (str_DataFormat='CPP') OR
                    (str_DataFormat='TPP') THEN
                    FB_step:=70;
                ELSE
                    ToError_Sts := TRUE;
                    ToErrorID:=16#30;
                    ToErrorDescrip:='DataFormat invalid for PTP command';
                END_IF;
            END_IF;

            IF (str_MovementCommand='Move_Line') THEN
                IF (str_DataFormat='CPP') OR
                    (str_DataFormat='CPR') OR
                    (str_DataFormat='CAP') OR
                    (str_DataFormat='CAR') OR
                    (str_DataFormat='TPP') OR
                    (str_DataFormat='TPR') OR
                    (str_DataFormat='TAP') OR
                    (str_DataFormat='TAR') THEN
                    FB_step:=70;
                ELSE
                    ToError_Sts := TRUE;
                    ToErrorID:=16#31;
                    ToErrorDescrip:='DataFormat invalid for Line command';
                END_IF;
            END_IF;

        FB_step:=40;

    40: (*  Frame buildingfor CheckSum calculation *)

            Header:='TMSCT';

            Script_Command:=CONCAT(    CONCAT(str_MovementCommand,'(',str_DataFormat,','),

            CONCAT(str_TargetPosition,',',str_Speed,','),

            CONCAT(str_AccelTime,',',str_BlendingValue,','),

            CONCAT(str_PrecisePositioning,')') );

            //When no blending Motion ended acknowledgement with QueueTag()
            IF NOT(iParameters.BlendingEnable) THEN
                Script_Command:=CONCAT(Script_Command,'$R$LQueueTag(',str_CommandID,',0)');
            END_IF;

            //Exits Listen Node in TMflow with ScriptExit()
            IF iParameters.ExitNode THEN
                Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
            END_IF;

            // CheckSum calculation
            str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));

        //Length for DATA command
```

```
                str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');          //Checksum includes: Header, Length for DATA command, str_CommandID and Script_Command

    Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);

                IF Checksum_Length>0 THEN
                    FB_step:=50;
                ELSE
                    ToError_Sts := TRUE;
                    ToErrorID:=16#40;
                    ToErrorDescrip:='Checksum length not valid';
                END_IF;



    50: (*  Frame buildingto send command to TMflow *)

                FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
                    Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
                END_FOR;


                str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
                str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');

                Length:=LEN(str_SendFrame);
                Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

                IF Long>0 THEN
                    FB_step:=60;
                ELSE
                    ToError_Sts := TRUE;
                    ToErrorID:=16#50;
                    ToErrorDescrip:='Final frame building is error end';
                END_IF;


    60: (*      Send command                    *)

                TCP_Send_Size:=LEN(str_SendFrame);
                ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH, AryOut:=TCP_Send_Data[0]);

                TCP_Send_Exe        :=TRUE;

                IF TCP_Send.Done THEN
                    CBT.WaitingReturn:=TRUE;            //Flag to set the robot in "busy" state to avoid other FB to be executed
                    TCP_Send_Exe:=FALSE;
                    FB_step:=70;
                END_IF;

                IF TCP_Send.Error THEN
                    TCP_Send_Exe:=FALSE;
                    ToError_Sts := TRUE;
                    ToErrorID:=16#60;
                    ToErrorDescrip:='TCP send error';
                END_IF;


    70: (*      Request receiving data                  *)

                TCP_Rcv_TimeOut:=0;                             //0: No timeouts
                TCP_Rcv_Size:=256;                              //Set number of bytes to read from the receive buffer
                StringOfReceivedData:='';               //Clear the variable where Receive data array is compiled

```

```
            TCP_Rcv_Exe :=TRUE;

            IF TCP_Rcv.Done THEN
                StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);    //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
                TCP_Rcv_Exe:=FALSE;
                FB_step:=80;
            END_IF;

            IF TCP_Rcv.Error THEN
                TCP_Rcv_Exe:=FALSE;
                ToError_Sts := TRUE;
                ToErrorID:=16#70;
                ToErrorDescrip:='TCP receive error';
            END_IF;


    80: (*  Check acknowledgement Command accepted                   *)

        IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN                     //Message no vaild
            FB_step:=70;
        END_IF;

        IF FIND(StringOfReceivedData,'TMSCT') <> 0 THEN
            IF FIND(StringOfReceivedData,'OK') <> 0 THEN                            //Command accepted
                IF NOT(iParameters.BlendingEnable) THEN
                    FB_step:=90;                                        //"Done" signal waits until motion is finished (no blending)
                ELSE
                    CBT.WaitingReturn:=FALSE;
                    //Flag to set the robot in "released" state to avoid other FB to be executed
                    CmdID_Ack:=STRING_TO_UINT(str_CommandID);  //Output the Command ID when ack is done
                    FB_step:=200;                                       //"Done" signal does not wait until motion ends (allows blending)
                END_IF;
            ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
                CBT.WaitingReturn:=FALSE;
                    //Flag to set the robot in "released" state to avoid other FB to be executed
                ToError_Sts := TRUE;
                ToErrorID:=16#80;
                ToErrorDescrip:=' Command rejected';
            END_IF;
        END_IF;


    90: (*  Clear receive buffer            *)
            TCP_Clear_Buffer_Exe:=TRUE;

            IF TCP_Clear_Buffer.Done THEN
                TCP_Clear_Buffer_Exe:=FALSE;
                FB_step:=100;
            END_IF;

            IF TCP_Clear_Buffer.Error THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#90;
                ToErrorDescrip:='Clear buffer error';
            END_IF;


    100: (* Request receiving data                   *)

            CmdID_Ack:=STRING_TO_UINT(str_CommandID);                               //Output the Command ID when ack is done

            TCP_Rcv_TimeOut:=0;                                     //0: No timeouts
```

```
			TCP_Rcv_Size:=256;				//Set number of bytes to read from the receive buffer
			StringOfReceivedData:='';			//Clear the variable where Receive data array is compiled

			TCP_Rcv_Exe :=TRUE;

			IF TCP_Rcv.Done THEN
				StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);	//Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
				TCP_Rcv_Exe:=FALSE;
				FB_step:=110;
			END_IF;

			IF TCP_Rcv.Error THEN
				TCP_Rcv_Exe:=FALSE;
				ToError_Sts := TRUE;
				ToErrorID:=16#100;
				ToErrorDescrip:='TCP receive error';
			END_IF;


	110: (*	Check acknowledgement Motion Completed		*)

			IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN		//QueueTag acknowledgement
				IF FIND(StringOfReceivedData,str_CommandID) <> 0 THEN		//QueueTag for the last motion command
					IF FIND(StringOfReceivedData,'true') <> 0 THEN		//Motion finished
						CBT.WaitingReturn:=FALSE;			//Flag to set the robot in "released" state to avoid other FB to be executed
						CmdID_Ack:=STRING_TO_UINT(str_CommandID);		//Output the Command ID when ack is done
						FB_step:=200;
					ELSIF FIND(StringOfReceivedData,'false') <> 0 THEN
						CBT.WaitingReturn:=FALSE;			//Flag to set the robot in "released" state to avoid other FB to be executed
						ToError_Sts := TRUE;
						ToErrorID:=16#110;
						ToErrorDescrip:=' Motion failed';
					END_IF;
				ELSE	//no str_CommandID
					FB_step:=100;
				END_IF;
			ELSE		//no TMSTA
				FB_step:=100;
			END_IF;



	200: (*			End Execution		*)
		ToDone_Sts:=TRUE;

	END_CASE ;

END_IF;




(* Function Bolcks										*)
(* -------------------------------------------------------------------	*)


TCP_Clear_Buffer(
	Execute:=TCP_Clear_Buffer_Exe,
	Socket:=CBT.Socket
	//Done=>, Busy=>, Error=>, ErrorID=>
```

```
    );

TCP_Send(
    Execute:=TCP_Send_Exe,
    Socket:=CBT.Socket,
    SendDat:=TCP_Send_Data[0],
    Size:=TCP_Send_Size
    //Done=>, Busy=>, Error=>, ErrorID=>
    );

TCP_Rcv(
    Execute:=TCP_Rcv_Exe,
    Socket:=CBT.Socket,
    TimeOut:=TCP_Rcv_TimeOut,
    Size:=TCP_Rcv_Size,
    RcvDat:=TCP_Rcv_Data[0],
    //Done=>, Busy=>, Error=>, ErrorID=>,
    RcvSize=>TCP_Rcv_RcvSize);




```

A4. CBT_MoveCircle

```
(* ************************************************************************      *)
(*                              Robot StartUp FB                                  *)
(* ************************************************************************      *)
(* ------------------------------------------------------------------------------
FB name:                        CBT_MoveCircle
(* ------------------------------------------------------------------------------
FB version:          V01
Library:             CBT_Commands
SS version:          V1.44
Date:                June 2021
Author:              Ruben Sanchez Boli
Description:         Sends circle motion commands
                     - Circle() -> Syntax 2 in expression editor
(* ------------------------------------------------------------------------------ *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
    IF FB_status <> 2 THEN                          //FBstatus_Step different than 2 - Busy (avoids re-execution)
            iParameters:=Parameters;            //Copy internal variable
            FB_status:=1;                                       //initates secuences
            FB_step:=10;                                        //Initates algorithm sequence
    END_IF;
END_IF;

//In case of EStop is triggered during FB execution
IF Execute AND NOT(CBT.Connected) THEN
    ToError_Sts := TRUE;
    ToErrorID:=16#99;
    ToErrorDescrip:='Robot not connected';
END_IF;

```

```
(* FB State Diagram control                                           *)
(* ------------------------------------------------------------------------------ *)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive


//Sequence
CASE FB_status OF

0: (*                   Idle                        *)
            FB_status:=0;


1: (*                   Reset               *)
            ToError_Sts:=FALSE;
            ToDone_Sts:=FALSE;

            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';
            FB_status:=2;


2: (*       Busy state  *)
            Done := FALSE;
            Busy := TRUE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';

            IF ToError_Sts THEN
                    FB_status:=3;               //Error flag
            END_IF;

            IF ToDone_Sts THEN
                    FB_status:=5;               //Done flag
            END_IF;


3: (*       Error state *)
            Done := FALSE;
            Busy := FALSE;
            Error := TRUE;
            ErrorID:=ToErrorID;
            ErrorDescrip:=ToErrorDescrip;

            IF NOT Execute THEN
                    FB_status:= 4;                              //Returns to idle
            END_IF;

4: (*       Error Deactive if not execute       *)
            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=ToErrorID;
            ErrorDescrip:=ToErrorDescrip;


5: (*       Done state *)
            Done := TRUE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';

            IF NOT Execute THEN
                    FB_status:= 6;                                      //Returns to idle
            END_IF;

6: (*                   Done deactive if not Execute                *)
            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';

END_CASE ;



(* FB Algorithm                                                      *)
(* -------------------------------------------------------------------  *)

IF Busy THEN

    CASE FB_step OF
    0: (*   Idle                    *)
                    FB_step:=0;

    10: (*  Initialization                  *)
                    //InternalStep variables
                    TCP_Clear_Buffer_Exe            :=FALSE;
                    TCP_Send_Exe                                    :=FALSE;
                    TCP_Rcv_Exe                                     :=FALSE;
                    ToErrorID                                               :=0;
                    ToErrorDescrip                                  :=' ';
                    CmdID_Ack                                               :=Parameters.CommandID;

                    IF CBT.Connected and NOT(CBT.WaitingReturn) THEN
                            FB_step:=20;
                    ELSIF NOT(CBT.Connected) THEN
                            ToError_Sts := TRUE;
                            ToErrorID:=16#10;
                            ToErrorDescrip:='Robot not connected';
                    ELSIF (CBT.WaitingReturn) THEN
                            ToError_Sts := TRUE;
                            ToErrorID:=16#11;
                            ToErrorDescrip:='Other FB is under execution';
                    END_IF;


    20: (*  Clear receive buffer            *)
                    TCP_Clear_Buffer_Exe:=TRUE;

                    IF TCP_Clear_Buffer.Done THEN
                            TCP_Clear_Buffer_Exe:=FALSE;
                            FB_step:=25;
                    END_IF;

                    IF TCP_Clear_Buffer.Error THEN
                            TCP_Clear_Buffer_Exe:=FALSE;
                            ToError_Sts := TRUE;
                            ToErrorID:=16#20;
                            ToErrorDescrip:='Clear buffer error';
                    END_IF;


    25: (*  Do not exceed the seetable range *)

                    IF iParameters.Speed > 4500 THEN
                            ToError_Sts := TRUE;
                            ToErrorID:=16#25;
                            ToErrorDescrip:='Speed range in Line: 0~4500 mm/s';
                    END_IF;

            IF iParameters.Speed > 100 THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#26;
                ToErrorDescrip:='Speed range in PTP: 0~100 %';
            END_IF;

            IF (iParameters.AccelTime < 150) OR (iParameters.AccelTime > 9999) THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#27;
                ToErrorDescrip:='Acceleration Time range is: 150~9999 ms';
            END_IF;

            IF (iParameters.CommandID < 2) OR (iParameters.CommandID > 9) THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#28;
                ToErrorDescrip:='Command ID range is: 2~9';
            END_IF;

            FB_step:=30;


    30: (*  iParameters conversion to string *)

            str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

            str_MovementCommand:='Circle';

            //iParameters.DataFormat
            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdCircle_DataFormat#CAP_Circle THEN
                str_DataFormat:='CAP';
            END_IF;

            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdCircle_DataFormat#CPP_Circle THEN
                str_DataFormat:='CPP';
            END_IF;

            str_AP_X := RealToFormatString(In:=iParameters.ArcPoint.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_AP_Y := RealToFormatString(In:=iParameters.ArcPoint.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_AP_Z := RealToFormatString(In:=iParameters.ArcPoint.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_AP_RX := RealToFormatString(In:=iParameters.ArcPoint.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_AP_RY := RealToFormatString(In:=iParameters.ArcPoint.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_AP_RZ := RealToFormatString(In:=iParameters.ArcPoint.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_AP1:=CONCAT(str_AP_X,',',str_AP_Y,',',str_AP_Z);
            str_AP2:=CONCAT(str_AP_RX,',',str_AP_RY,',',str_AP_RZ);
            str_ArcPoint:=CONCAT(str_AP1,',',str_AP2);

            str_EP_X := RealToFormatString(In:=iParameters.EndPoint.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_EP_Y := RealToFormatString(In:=iParameters.EndPoint.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_EP_Z := RealToFormatString(In:=iParameters.EndPoint.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_EP_RX := RealToFormatString(In:=iParameters.EndPoint.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_EP_RY := RealToFormatString(In:=iParameters.EndPoint.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_EP_RZ := RealToFormatString(In:=iParameters.EndPoint.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_EP1:=CONCAT(str_EP_X,',',str_EP_Y,',',str_EP_Z);
            str_EP2:=CONCAT(str_EP_RX,',',str_EP_RY,',',str_EP_RZ);
            str_EndPoint:=CONCAT(str_EP1,',',str_EP2);

```

```
                str_Speed:=UINT_TO_STRING(iParameters.Speed);
                str_AccelTime:=UINT_TO_STRING(iParameters.AccelTime);

                IF iParameters.BlendingEnable THEN
                    str_BlendingValue:=UINT_TO_STRING(iParameters.BlendingValue);
                ELSE
                    str_BlendingValue:='0';
                END_IF;

                str_AcrAngle:=UINT_TO_STRING(iParameters.ArcAngle);

                IF iParameters.PrecisePositioning THEN
                    str_PrecisePositioning:='true';
                ELSE
                    str_PrecisePositioning:='false';
                END_IF;

            FB_step:=40;

    40: (*  Frame buildingfor CheckSum calculation *)

                Header:='TMSCT';

                Script_Command:=CONCAT(    CONCAT(str_MovementCommand,'(',str_DataFormat,','),

                CONCAT(str_ArcPoint,','),

                CONCAT(str_EndPoint,',',str_Speed,','),

                CONCAT(str_AccelTime,',',str_BlendingValue,','),

                CONCAT(str_AcrAngle,',',str_PrecisePositioning,')') );


                //When no blending Motion ended acknowledgement with QueueTag()
                IF NOT(iParameters.BlendingEnable) THEN
                    Script_Command:=CONCAT(Script_Command,'$R$LQueueTag(',str_CommandID,',0)');
                END_IF;

                //Exits Listen Node in TMflow with ScriptExit()
                IF iParameters.ExitNode THEN
                    Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
                END_IF;

                // CheckSum calculation
                str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));
            //Length for DATA command
                str_Checksum_Calc :=
CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');                //Checksum
includes: Header, Length for DATA command, str_CommandID and Script_Command


    Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);

                IF Checksum_Length>0 THEN
                    FB_step:=50;
                ELSE
                    ToError_Sts := TRUE;
                    ToErrorID:=16#40;
                    ToErrorDescrip:='Checksum length not valid';
                END_IF;



    50: (*  Frame buildingto send command to TMflow *)

                FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
                    Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
                END_FOR;


            str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
            str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');

            Length:=LEN(str_SendFrame);
            Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);


            IF Long>0 THEN
                FB_step:=60;
            ELSE
                ToError_Sts := TRUE;
                ToErrorID:=16#50;
                ToErrorDescrip:='Final frame building is error end';
            END_IF;


    60: (*    Send command                *)

            TCP_Send_Size:=LEN(str_SendFrame);
            ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH, AryOut:=TCP_Send_Data[0]);

            TCP_Send_Exe      :=TRUE;

            IF TCP_Send.Done THEN
                CBT.WaitingReturn:=TRUE;            //Flag to set the robot in "busy" state to avoid other FB to be executed
                TCP_Send_Exe:=FALSE;
                FB_step:=70;
            END_IF;

            IF TCP_Send.Error THEN
                TCP_Send_Exe:=FALSE;
                ToError_Sts := TRUE;
                ToErrorID:=16#60;
                ToErrorDescrip:='TCP send error';
            END_IF;


    70: (*    Request receiving data                *)

            TCP_Rcv_TimeOut:=0;                            //0: No timeouts
            TCP_Rcv_Size:=256;                             //Set number of bytes to read from the receive buffer
            StringOfReceivedData:='';              //Clear the variable where Receive data array is compiled

            TCP_Rcv_Exe :=TRUE;

            IF TCP_Rcv.Done THEN
                StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);    //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
                TCP_Rcv_Exe:=FALSE;
                FB_step:=80;
            END_IF;

            IF TCP_Rcv.Error THEN
                TCP_Rcv_Exe:=FALSE;
                ToError_Sts := TRUE;
                ToErrorID:=16#70;
                ToErrorDescrip:='TCP receive error';
            END_IF;


    80: (*    Check acknowledgement Command accepted                *)

        IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN                //Message no vaild
            FB_step:=70;
        END_IF;

        IF FIND(StringOfReceivedData,'TMSCT') <> 0 THEN
            IF FIND(StringOfReceivedData,'OK') <> 0 THEN                    //Command accepted
                IF NOT(iParameters.BlendingEnable) THEN
                    FB_step:=90;                //"Done" signal waits until motion is finished (no blending)
                ELSE
                    CBT.WaitingReturn:=FALSE;
                    //Flag to set the robot in "released" state to avoid other FB to be executed
                    CmdID_Ack:=STRING_TO_UINT(str_CommandID);  //Output the Command ID when ack is done
                    FB_step:=200;               //"Done" signal does not wait until motion ends (allows blending)
                END_IF;
            ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
                CBT.WaitingReturn:=FALSE;
                    //Flag to set the robot in "released" state to avoid other FB to be executed
                ToError_Sts := TRUE;
                ToErrorID:=16#80;
                ToErrorDescrip:=' Command rejected';
            END_IF;
        END_IF;

    90: (*  Clear receive buffer        *)
            TCP_Clear_Buffer_Exe:=TRUE;

            IF TCP_Clear_Buffer.Done THEN
                TCP_Clear_Buffer_Exe:=FALSE;
                FB_step:=100;
            END_IF;

            IF TCP_Clear_Buffer.Error THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#90;
                ToErrorDescrip:='Clear buffer error';
            END_IF;


    100: (*  Request receiving data              *)

            CmdID_Ack:=STRING_TO_UINT(str_CommandID);                   //Output the Command ID when ack is done

            TCP_Rcv_TimeOut:=0;                         //0: No timeouts
            TCP_Rcv_Size:=256;                          //Set number of bytes to read from the receive buffer
            StringOfReceivedData:='';           //Clear the variable where Receive data array is compiled

            TCP_Rcv_Exe :=TRUE;

            IF TCP_Rcv.Done THEN
                StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
    //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
                TCP_Rcv_Exe:=FALSE;
                FB_step:=110;
            END_IF;

            IF TCP_Rcv.Error THEN
                TCP_Rcv_Exe:=FALSE;
                ToError_Sts := TRUE;
                ToErrorID:=16#100;
                ToErrorDescrip:='TCP receive error';
            END_IF;


    110: (*  Check acknowledgement Motion Completed        *)

            IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN
    //QueueTag acknowledgement
```

```
                IF FIND(StringOfReceivedData,str_CommandID) <> 0 THEN                   //QueueTag for the last motion command
                    IF FIND(StringOfReceivedData,'true') <> 0 THEN            //Motion finished
                        CBT.WaitingReturn:=FALSE;                           //Flag to set the robot in "released" state to avoid other FB to be executed
                        CmdID_Ack:=STRING_TO_UINT(str_CommandID);           //Output the Command ID when ack is done
                        FB_step:=200;
                    ELSIF FIND(StringOfReceivedData,'false') <> 0 THEN
                        CBT.WaitingReturn:=FALSE;                           //Flag to set the robot in "released" state to avoid other FB to be executed
                        ToError_Sts := TRUE;
                        ToErrorID:=16#110;
                        ToErrorDescrip:=' Motion failed';
                    END_IF;
                ELSE        //no str_CommandID
                    FB_step:=100;
                END_IF;
            ELSE                //no TMSTA
                FB_step:=100;
            END_IF;



    200: (*                    End Execution           *)
        ToDone_Sts:=TRUE;

    END_CASE ;

END_IF;




(* Function Bolcks                                              *)
(* -----------------------------------------------------------  *)


TCP_Clear_Buffer(
    Execute:=TCP_Clear_Buffer_Exe,
    Socket:=CBT.Socket
    //Done=>, Busy=>, Error=>, ErrorID=>
    );

TCP_Send(
    Execute:=TCP_Send_Exe,
    Socket:=CBT.Socket,
    SendDat:=TCP_Send_Data[0],
    Size:=TCP_Send_Size
    //Done=>, Busy=>, Error=>, ErrorID=>
    );

TCP_Rcv(
    Execute:=TCP_Rcv_Exe,
    Socket:=CBT.Socket,
    TimeOut:=TCP_Rcv_TimeOut,
    Size:=TCP_Rcv_Size,
    RcvDat:=TCP_Rcv_Data[0],
    //Done=>, Busy=>, Error=>, ErrorID=>,
    RcvSize=>TCP_Rcv_RcvSize);
```

A5. CBT_ChangeBase

```
(* ----------------------------------------------------------------------------------
FB name:                CBT_ChangeBase
(* ----------------------------------------------------------------------------------
FB version:             V01
Library:                CBT_Commands
SS version:             V1.44
Date:                   June 2021
Author:                 Ruben Sanchez Boli
Description:            Sets the User Coordinate System
                        - ChangeBase() -> syntax 3 in expression editor
(* ---------------------------------------------------------------------------------- *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
    IF FB_status <> 2 THEN                  //FBstatus_Step different than 2 - Busy (avoids re-execution)
              iParameters:=Parameters;      //Copy internal variable
              FB_status:=1;                 //initates secuences
              FB_step:=10;                  //Initates algorithm sequence
    END_IF;
END_IF;

//In case of EStop is triggered during FB execution
IF Execute AND NOT(CBT.Connected) THEN
    ToError_Sts := TRUE;
    ToErrorID:=16#99;
    ToErrorDescrip:='Robot not connected';
END_IF;

(* FB State Diagram control                                                       *)
(* ------------------------------------------------------------------------------ *)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive


//Sequence
CASE FB_status OF

0: (* Idle                          *)
              FB_status:=0;


1: (*Reset                *)
              ToError_Sts:=FALSE;
              ToDone_Sts:=FALSE;

              Done := FALSE;
              Busy := FALSE;
              Error := FALSE;
              ErrorID:=0;
              ErrorDescrip:=' ';
              FB_status:=2;


2: (*Busy state *)
              Done := FALSE;
              Busy := TRUE;
              Error := FALSE;
              ErrorID:=0;
              ErrorDescrip:=' ';
```

```

            IF ToError_Sts THEN
                    FB_status:=3;                   //Error flag
            END_IF;

            IF ToDone_Sts THEN
                    FB_status:=5;                   //Done flag
            END_IF;


3: (*Error state *)
            Done := FALSE;
            Busy := FALSE;
            Error := TRUE;
            ErrorID:=ToErrorID;
            ErrorDescrip:=ToErrorDescrip;

            IF NOT Execute THEN
                    FB_status:= 4;                                  //Returns to idle
            END_IF;

4: (*Error Deactive if not execute      *)
            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=ToErrorID;
            ErrorDescrip:=ToErrorDescrip;


5: (*Done state *)
            Done := TRUE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';

            IF NOT Execute THEN
                    FB_status:= 6;                                  //Returns to idle
            END_IF;

6: (*Done deactive if not Execute                   *)
            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';

END_CASE ;



(* FB Algorithm                                             *)
(* -------------------------------------------------------------------  *)

IF Busy THEN

   CASE FB_step OF
   0: (*    Idle                *)
                    FB_step:=0;

   10: (*   Initialization              *)
                    //InternalStep variables
                    TCP_Clear_Buffer_Exe        :=FALSE;
                    TCP_Send_Exe                                :=FALSE;
                    TCP_Rcv_Exe                                 :=FALSE;
                    ToErrorID                                           :=0;
                    ToErrorDescrip                              :=' ';

                    IF CBT.Connected and NOT(CBT.WaitingReturn) THEN
                            FB_step:=20;
                    ELSIF NOT(CBT.Connected) THEN
```

```
                                ToError_Sts := TRUE;
                                ToErrorID:=16#10;
                                ToErrorDescrip:='Robot not connected';
                        ELSIF (CBT.WaitingReturn) THEN
                                ToError_Sts := TRUE;
                                ToErrorID:=16#11;
                                ToErrorDescrip:='Other FB is under execution';
                        END_IF;

        20: (*      Clear receive buffer                *)
                        TCP_Clear_Buffer_Exe:=TRUE;

                        IF TCP_Clear_Buffer.Done THEN
                                TCP_Clear_Buffer_Exe:=FALSE;
                                FB_step:=30;
                        END_IF;

                        IF TCP_Clear_Buffer.Error THEN
                                TCP_Clear_Buffer_Exe:=FALSE;
                                ToError_Sts := TRUE;
                                ToErrorID:=16#20;
                                ToErrorDescrip:='Clear buffer error';
                        END_IF;



        30: (*  iParam conversion to string *)

                        str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

                        str_TP_X := RealToFormatString(In:=iParameters.Transformation.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
                        str_TP_Y := RealToFormatString(In:=iParameters.Transformation.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
                        str_TP_Z := RealToFormatString(In:=iParameters.Transformation.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
                        str_TP_RX := RealToFormatString(In:=iParameters.Transformation.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
                        str_TP_RY := RealToFormatString(In:=iParameters.Transformation.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
                        str_TP_RZ := RealToFormatString(In:=iParameters.Transformation.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
                        str_TP1:=CONCAT(str_TP_X,',',str_TP_Y,',',str_TP_Z);
                        str_TP2:=CONCAT(str_TP_RX,',',str_TP_RY,',',str_TP_RZ);
                        str_TargetPosition:=CONCAT(str_TP1,',',str_TP2);

                        FB_step:=40;

        40: (*  Frame buildingfor CheckSum calculation *)

                        Header:='TMSCT';
                        str_FunctionCommand:='ChangeBase';
                        Script_Command:=CONCAT(str_FunctionCommand,'(',str_TargetPosition,')');

                        //Exits Listen Node in TMflow with ScriptExit()
                        IF iParameters.ExitNode THEN
                                Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
                        END_IF;

                        // CheckSum calculation
                        str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));                    //Length for DATA command
                        str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');                    //Checksum includes: Header, Length for DATA command, str_CommandID and Script_Command


        Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);

```

```
                    IF Checksum_Length>0 THEN
                        FB_step:=50;
                    ELSE
                        ToError_Sts := TRUE;
                        ToErrorID:=16#40;
                        ToErrorDescrip:='Checksum length not valid';
                    END_IF;



        50: (*  Frame buildingto send command to TMflow *)

                    FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
                        Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
                    END_FOR;

                    str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
                    str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');

                    Length:=LEN(str_SendFrame);
                    Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);


                    IF Long>0 THEN
                        FB_step:=60;
                    ELSE
                        ToError_Sts := TRUE;
                        ToErrorID:=16#50;
                        ToErrorDescrip:='Final frame building is error end';
                    END_IF;


        60: (*      Send command                    *)

                    TCP_Send_Size:=LEN(str_SendFrame);
                    ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH, AryOut:=TCP_Send_Data[0]);

                    TCP_Send_Exe        :=TRUE;

                    IF TCP_Send.Done THEN
                        CBT.WaitingReturn:=TRUE;                //Flag to set the robot in "busy" state to avoid other FB to be executed
                        TCP_Send_Exe:=FALSE;
                        FB_step:=70;
                    END_IF;

                    IF TCP_Send.Error THEN
                        TCP_Send_Exe:=FALSE;
                        ToError_Sts := TRUE;
                        ToErrorID:=16#60;
                        ToErrorDescrip:='TCP send error';
                    END_IF;


        70: (*      Request receiving data                  *)

                    TCP_Rcv_TimeOut:=0;                                 //0: No timeouts
                    TCP_Rcv_Size:=256;                                  //Set number of bytes to read from the receive buffer
                    StringOfReceivedData:='';               //Clear the variable where Receive data array is compiled

                    TCP_Rcv_Exe :=TRUE;

                    IF TCP_Rcv.Done THEN
                        StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);        //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
                        TCP_Rcv_Exe:=FALSE;
                        FB_step:=80;
                    END_IF;
```

```

                IF TCP_Rcv.Error THEN
                    TCP_Rcv_Exe:=FALSE;
                    ToError_Sts := TRUE;
                    ToErrorID:=16#70;
                    ToErrorDescrip:='TCP receive error';
                END_IF;


        80: (*  Check acknowledgement Command accepted                    *)

            IF FIND(StringOfReceivedData,'OK') <> 0 THEN                    //Command accepted
                CBT.WaitingReturn:=FALSE;
                //Flag to set the robot in "released" state to avoid other FB to be executed
                FB_step:=90;
                                                    //Done FB when motion is finished (no blending)
            ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
                CBT.WaitingReturn:=FALSE;
                //Flag to set the robot in "released" state to avoid other FB to be executed
                ToError_Sts := TRUE;
                ToErrorID:=16#80;
                ToErrorDescrip:=' Command rejected';
            END_IF;


        90: (*          End Execution       *)
            ToDone_Sts:=TRUE;

        END_CASE ;

    END_IF;




    (* Function Blocks                                                  *)
    (* ---------------------------------------------------------------- *)


    TCP_Clear_Buffer(
        Execute:=TCP_Clear_Buffer_Exe,
        Socket:=CBT.Socket
        //Done=>, Busy=>, Error=>, ErrorID=>
        );

    TCP_Send(
        Execute:=TCP_Send_Exe,
        Socket:=CBT.Socket,
        SendDat:=TCP_Send_Data[0],
        Size:=TCP_Send_Size
        //Done=>, Busy=>, Error=>, ErrorID=>
        );

    TCP_Rcv(
        Execute:=TCP_Rcv_Exe,
        Socket:=CBT.Socket,
        TimeOut:=TCP_Rcv_TimeOut,
        Size:=TCP_Rcv_Size,
        RcvDat:=TCP_Rcv_Data[0],
        //Done=>, Busy=>, Error=>, ErrorID=>,
        RcvSize=>TCP_Rcv_RcvSize);
```

A6. CBT_ChangeTCP

```
(* -------------------------------------------------------------------------------
FB name:                    CBT_ChangeTCP
(* -------------------------------------------------------------------------------
FB version:                 V01
Library:                    CBT_Commands
SS version:                 V1.44
Date:                       June 2021
Author:                     Ruben Sanchez Boli
Description:                Sets the Tool Coordinate System
                            - ChangeTCP() -> Syntax 7 in expression editor
(* ------------------------------------------------------------------------------- *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
    IF FB_status <> 2 THEN                      //FBstatus_Step different than 2 - Busy (avoids re-execution)
            iParameters:=Parameters;            //Copy internal variable
            FB_status:=1;                       //initates secuences
            FB_step:=10;                        //Initates algorithm sequence
    END_IF;
END_IF;

//In case of EStop is triggered during FB execution
IF Execute AND NOT(CBT.Connected) THEN
    ToError_Sts := TRUE;
    ToErrorID:=16#99;
    ToErrorDescrip:='Robot not connected';
END_IF;

(* FB State Diagram control                                                        *)
(* ------------------------------------------------------------------------------- *)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive


//Sequence
CASE FB_status OF

0: (*       Idle              *)
            FB_status:=0;


1: (*       Rese              *)
            ToError_Sts:=FALSE;
            ToDone_Sts:=FALSE;

            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';
            FB_status:=2;


2: (*       Busy state  *)
            Done := FALSE;
            Busy := TRUE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';
```

```

           IF ToError_Sts THEN
                   FB_status:=3;           //Error flag
           END_IF;

           IF ToDone_Sts THEN
                   FB_status:=5;           //Done flag
           END_IF;


3: (*      Error state *)
           Done := FALSE;
           Busy := FALSE;
           Error := TRUE;
           ErrorID:=ToErrorID;
           ErrorDescrip:=ToErrorDescrip;

           IF NOT Execute THEN
                   FB_status:= 4;                          //Returns to idle
           END_IF;

4: (*      Error Deactive if not execute   *)
           Done := FALSE;
           Busy := FALSE;
           Error := FALSE;
           ErrorID:=ToErrorID;
           ErrorDescrip:=ToErrorDescrip;


5: (*      Done state *)
           Done := TRUE;
           Busy := FALSE;
           Error := FALSE;
           ErrorID:=0;
           ErrorDescrip:=' ';

           IF NOT Execute THEN
                   FB_status:= 6;                          //Returns to idle
           END_IF;

6: (*              Done deactive if not Execute    *)
           Done := FALSE;
           Busy := FALSE;
           Error := FALSE;
           ErrorID:=0;
           ErrorDescrip:=' ';

END_CASE ;



(* FB Algorithm                            *)
(* --------------------------------------------------------------------  *)

IF Busy THEN

   CASE FB_step OF
   0: (*   Idle            *)
                   FB_step:=0;

   10: (*  Initialization          *)
                   //InternalStep variables
                   TCP_Clear_Buffer_Exe    :=FALSE;
                   TCP_Send_Exe            :=FALSE;
                   TCP_Rcv_Exe             :=FALSE;
                   ToErrorID               :=0;
                   ToErrorDescrip          :=' ';

                   IF CBT.Connected and NOT(CBT.WaitingReturn) THEN
                           FB_step:=20;
                   ELSIF NOT(CBT.Connected) THEN
```

```
                ToError_Sts := TRUE;
                ToErrorID:=16#10;
                ToErrorDescrip:='Robot not connected';
            ELSIF (CBT.WaitingReturn) THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#11;
                ToErrorDescrip:='Other FB is under execution';
            END_IF;


    20: (*    Clear receive buffer            *)
            TCP_Clear_Buffer_Exe:=TRUE;

            IF TCP_Clear_Buffer.Done THEN
                TCP_Clear_Buffer_Exe:=FALSE;
                FB_step:=25;
            END_IF;

            IF TCP_Clear_Buffer.Error THEN
                TCP_Clear_Buffer_Exe:=FALSE;
                ToError_Sts := TRUE;
                ToErrorID:=16#20;
                ToErrorDescrip:='Clear buffer error';
            END_IF;

    25: (*    Do not exceed the seetable range *)

            IF iParameters.Weight > 14 THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#25;
                ToErrorDescrip:='Weight range: 0~14 kg';
            END_IF;

            FB_step:=30;

    30: (*  iBaseParam conversion to string *)

            str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

            //TCPOffset
            str_TO_X := RealToFormatString(In:=iParameters.TCPOffset.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_TO_Y := RealToFormatString(In:=iParameters.TCPOffset.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_TO_Z := RealToFormatString(In:=iParameters.TCPOffset.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_TO_RX := RealToFormatString(In:=iParameters.TCPOffset.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_TO_RY := RealToFormatString(In:=iParameters.TCPOffset.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_TO_RZ := RealToFormatString(In:=iParameters.TCPOffset.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_TO1:=CONCAT(str_TO_X,',',str_TO_Y,',',str_TO_Z);
            str_TO2:=CONCAT(str_TO_RX,',',str_TO_RY,',',str_TO_RZ);
            str_TCPOffset:=CONCAT(str_TO1,',',str_TO2);

            //MomentOfInertia
            str_Ixx := RealToFormatString(In:=iParameters.MomentOfInertia.Ixx, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_Iyy := RealToFormatString(In:=iParameters.MomentOfInertia.Iyy, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_Izz := RealToFormatString(In:=iParameters.MomentOfInertia.Izz, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_Inertia:=CONCAT(str_Ixx,',',str_Iyy,',',str_Izz);

            //Weight
            str_Weight := RealToFormatString(In:=iParameters.Weight, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);

            //MassCenter
            str_MC_X := RealToFormatString(In:=iParameters.MassCenter.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_MC_Y := RealToFormatString(In:=iParameters.MassCenter.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_MC_Z := RealToFormatString(In:=iParameters.MassCenter.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_MC_RX := RealToFormatString(In:=iParameters.MassCenter.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_MC_RY := RealToFormatString(In:=iParameters.MassCenter.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_MC_RZ := RealToFormatString(In:=iParameters.MassCenter.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
            str_MC1:=CONCAT(str_MC_X,',',str_MC_Y,',',str_MC_Z);
            str_MC2:=CONCAT(str_MC_RX,',',str_MC_RY,',',str_MC_RZ);
            str_MassCenter:=CONCAT(str_MC1,',',str_MC2);

            str_TCP1:=CONCAT(str_TCPOffset,',',str_Weight,',');
            str_TCP2:=CONCAT(str_Inertia,',',str_MassCenter);
            str_TCPCommand:=CONCAT(str_TCP1,str_TCP2);

            FB_step:=40;

    40: (*  Frame buildingfor CheckSum calculation *)

            Header:='TMSCT';
            str_FunctionCommand:='ChangeTCP';
            Script_Command:=CONCAT(str_FunctionCommand,'(',str_TCPCommand,')');

            //Exits Listen Node in TMflow with ScriptExit()
            IF iParameters.ExitNode THEN
                Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
            END_IF;

            // CheckSum calculation
            str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));        //Length for DATA command
            str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');        //Checksum includes: Header, Length for DATA command, str_CommandID and Script_Command


    Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);

            IF Checksum_Length>0 THEN
                FB_step:=50;
            ELSE
                ToError_Sts := TRUE;
                ToErrorID:=16#40;
                ToErrorDescrip:='Checksum length not valid';
            END_IF;



    50: (*  Frame buildingto send command to TMflow *)

            FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
                Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
            END_FOR;


            str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
            str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');

            Length:=LEN(str_SendFrame);
            Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);


            IF Long>0 THEN
                FB_step:=60;
                ELSE
                    ToError_Sts := TRUE;
                    ToErrorID:=16#50;
                    ToErrorDescrip:='Final frame building is error end';
                END_IF;


        60: (*  Send command            *)

                TCP_Send_Size:=LEN(str_SendFrame);
                ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH, AryOut:=TCP_Send_Data[0]);

                TCP_Send_Exe    :=TRUE;

                IF TCP_Send.Done THEN
                    CBT.WaitingReturn:=TRUE;        //Flag to set the robot in "busy" state to avoid other FB to be executed
                    TCP_Send_Exe:=FALSE;
                    FB_step:=70;
                END_IF;

                IF TCP_Send.Error THEN
                    TCP_Send_Exe:=FALSE;
                    ToError_Sts := TRUE;
                    ToErrorID:=16#60;
                    ToErrorDescrip:='TCP send error';
                END_IF;


        70: (*  Request receiving data          *)

                TCP_Rcv_TimeOut:=0;                         //0: No timeouts
                TCP_Rcv_Size:=256;                          //Set number of bytes to read from the receive buffer
                StringOfReceivedData:='';           //Clear the variable where Receive data array is compiled

                TCP_Rcv_Exe :=TRUE;

                IF TCP_Rcv.Done THEN
                    StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
        //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
                    TCP_Rcv_Exe:=FALSE;
                    FB_step:=80;
                END_IF;

                IF TCP_Rcv.Error THEN
                    TCP_Rcv_Exe:=FALSE;
                    ToError_Sts := TRUE;
                    ToErrorID:=16#70;
                    ToErrorDescrip:='TCP receive error';
                END_IF;


        80: (*  Check acknowledgement Command accepted          *)

            IF FIND(StringOfReceivedData,'OK') <> 0 THEN            //Command accepted
                CBT.WaitingReturn:=FALSE;
                //Flag to set the robot in "released" state to avoid other FB to be executed
                FB_step:=90;
                                        //Done FB when motion is finished (no blending)
            ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
                CBT.WaitingReturn:=FALSE;
                //Flag to set the robot in "released" state to avoid other FB to be executed
                ToError_Sts := TRUE;
                ToErrorID:=16#80;
                ToErrorDescrip:=' Command rejected';
            END_IF;


        90: (*          End Execution       *)
```

```
                    ToDone_Sts:=TRUE;

        END_CASE ;

END_IF;




(* Function Blocks                                                          *)
(* -----------------------------------------------------------------------  *)


TCP_Clear_Buffer(
        Execute:=TCP_Clear_Buffer_Exe,
        Socket:=CBT.Socket
        //Done=>, Busy=>, Error=>, ErrorID=>
        );

TCP_Send(
        Execute:=TCP_Send_Exe,
        Socket:=CBT.Socket,
        SendDat:=TCP_Send_Data[0],
        Size:=TCP_Send_Size
        //Done=>, Busy=>, Error=>, ErrorID=>
        );

TCP_Rcv(
        Execute:=TCP_Rcv_Exe,
        Socket:=CBT.Socket,
        TimeOut:=TCP_Rcv_TimeOut,
        Size:=TCP_Rcv_Size,
        RcvDat:=TCP_Rcv_Data[0],
        //Done=>, Busy=>, Error=>, ErrorID=>,
        RcvSize=>TCP_Rcv_RcvSize);
```

A7. CBT_ProgramControl

```
(* ----------------------------------------------------------------------------------
FB name:                    CBT_ProgramControl
(* ----------------------------------------------------------------------------------
FB version:                 V01
Library:                    CBT_Commands
SS version:                 V1.44
Date:                       June 2021
Author:                     Ruben Sanchez Boli
Description:                Sets the User Coordinate System
(* ---------------------------------------------------------------------------------- *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
        IF FB_status <> 2 THEN                          //FBstatus_Step different than 2 - Busy (avoids re-execution)
                    iParameters:=Parameters;            //Copy internal variable
                    FB_status:=1;                       //initates secuences
                    FB_step:=10;
        END_IF;
END_IF;

//In case of EStop is triggered during FB execution
IF Execute AND NOT(CBT.Connected) THEN
        ToError_Sts := TRUE;
```

```
    ToErrorID:=16#99;
    ToErrorDescrip:='Robot not connected';
END_IF;

(* FB State Diagram control                                                    *)
(* -------------------------------------------------------------------------------- *)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive


//Sequence
CASE FB_status OF

0: (*                    Idle                           *)
              FB_status:=0;


1: (*                    Reset                *)
              ToError_Sts:=FALSE;
              ToDone_Sts:=FALSE;

              Done := FALSE;
              Busy := FALSE;
              Error := FALSE;
              ErrorID:=0;
              ErrorDescrip:=' ';
              FB_status:=2;


2: (*          Busy state  *)
              Done := FALSE;
              Busy := TRUE;
              Error := FALSE;
              ErrorID:=0;
              ErrorDescrip:=' ';

              IF ToError_Sts THEN
                        FB_status:=3;                //Error flag
              END_IF;

              IF ToDone_Sts THEN
                        FB_status:=5;                //Done flag
              END_IF;


3: (*          Error state *)
              Done := FALSE;
              Busy := FALSE;
              Error := TRUE;
              ErrorID:=ToErrorID;
              ErrorDescrip:=ToErrorDescrip;

              IF NOT Execute THEN
                        FB_status:= 4;                                  //Returns to idle
              END_IF;

4: (*          Error Deactive if not execute      *)
              Done := FALSE;
              Busy := FALSE;
              Error := FALSE;
              ErrorID:=ToErrorID;
              ErrorDescrip:=ToErrorDescrip;


5: (*          Done state *)
              Done := TRUE;
```

```
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';

            IF NOT Execute THEN
                    FB_status:= 6;                                  //Returns to idle
            END_IF;

6: (*                   Done deactive if not Execute          *)
            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';

END_CASE ;



(* FB Algorithm                                               *)
(* ---------------------------------------------------------  *)

IF Busy THEN

    CASE FB_step OF
    0: (*   Idle                *)
                    FB_step:=0;

    10: (*  Initialization              *)
                    //InternalStep variables
                    TCP_Clear_Buffer_Exe        :=FALSE;
                    TCP_Send_Exe                        :=FALSE;
                    TCP_Rcv_Exe                         :=FALSE;
                    ToErrorID                           :=0;
                    ToErrorDescrip                      :=' ';

                    IF CBT.Connected THEN
                            FB_step:=20;
                    ELSE
                            ToError_Sts := TRUE;
                            ToErrorID:=16#10;
                            ToErrorDescrip:='Robot not connected';
                    END_IF;


    20: (*  Clear receive buffer        *)
                    TCP_Clear_Buffer_Exe:=TRUE;

                    IF TCP_Clear_Buffer.Done THEN
                            TCP_Clear_Buffer_Exe:=FALSE;
                            FB_step:=30;
                    END_IF;

                    IF TCP_Clear_Buffer.Error THEN
                            TCP_Clear_Buffer_Exe:=FALSE;
                            ToError_Sts := TRUE;
                            ToErrorID:=16#20;
                            ToErrorDescrip:='Clear buffer error';
                    END_IF;




    30: (*  iParam conversion to string *)

                    str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

                    // iParameters.MovementCommand
                    IF iParameters.Command=\\CBT_Commands_Lib\eCBT_PrgControlCmd#Pause THEN
                            str_FunctionCommand:='Pause()';
```

```
                        END_IF;

                        IF iParameters.Command=\\CBT_Commands_Lib\eCBT_PrgControlCmd#Resume THEN
                                str_FunctionCommand:='Resume()';
                        END_IF;

                        FB_step:=40;

        40: (*  Frame buildingfor CheckSum calculation *)

                        Header:='TMSCT';

                        Script_Command:=str_FunctionCommand;

                        // CheckSum calculation
                        str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));            //Length for DATA command
                        str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');            //Checksum includes: Header, Length for DATA command, str_CommandID and Script_Command


        Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);

                        IF Checksum_Length>0 THEN
                                FB_step:=50;
                        ELSE
                                ToError_Sts := TRUE;
                                ToErrorID:=16#40;
                                ToErrorDescrip:='Checksum length not valid';
                        END_IF;



        50: (*  Frame buildingto send command to TMflow *)

                        FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
                                Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
                        END_FOR;

                        str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
                        str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');

                        Length:=LEN(str_SendFrame);
                        Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);


                        IF Long>0 THEN
                                FB_step:=60;
                        ELSE
                                ToError_Sts := TRUE;
                                ToErrorID:=16#50;
                                ToErrorDescrip:='Final frame building is error end';
                        END_IF;


        60: (*        Send command                    *)

                        TCP_Send_Size:=LEN(str_SendFrame);
                        ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH, AryOut:=TCP_Send_Data[0]);

                        TCP_Send_Exe          :=TRUE;

                        IF TCP_Send.Done THEN
                                TCP_Send_Exe:=FALSE;
                                //FB_step:=70;
                                FB_step:=90;                    //This FB does not checks the return confirmation data from robot controller.
```

```
                END_IF;

                IF TCP_Send.Error THEN
                    TCP_Send_Exe:=FALSE;
                    ToError_Sts := TRUE;
                    ToErrorID:=16#60;
                    ToErrorDescrip:='TCP send error';
                END_IF;


        70: (*      Request receiving data                          *)

                TCP_Rcv_TimeOut:=0;                                     //0: No timeouts
                TCP_Rcv_Size:=256;                                      //Set number of bytes to read from the receive buffer
                StringOfReceivedData:='';                       //Clear the variable where Receive data array is compiled

                TCP_Rcv_Exe :=TRUE;

                IF TCP_Rcv.Done THEN
                    StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
        //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
                    TCP_Rcv_Exe:=FALSE;
                    FB_step:=80;
                END_IF;

                IF TCP_Rcv.Error THEN
                    TCP_Rcv_Exe:=FALSE;
                    ToError_Sts := TRUE;
                    ToErrorID:=16#70;
                    ToErrorDescrip:='TCP receive error';
                END_IF;


        80: (*      Check acknowledgement Command accepted                          *)

            IF FIND(StringOfReceivedData,'OK') <> 0 THEN                        //Command accepted
                FB_step:=90;
                                                        //Done FB when motion is finished (no blending)
            ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
                ToError_Sts := TRUE;
                ToErrorID:=16#80;
                ToErrorDescrip:=' Command rejected';
            END_IF;


        90: (*              End Execution           *)
            ToDone_Sts:=TRUE;

        END_CASE ;

END_IF;




(* Function Blocks                                              *)
(* ---------------------------------------------------------------- *)


TCP_Clear_Buffer(
        Execute:=TCP_Clear_Buffer_Exe,
        Socket:=CBT.Socket
        //Done=>, Busy=>, Error=>, ErrorID=>
        );

TCP_Send(
        Execute:=TCP_Send_Exe,
        Socket:=CBT.Socket,
        SendDat:=TCP_Send_Data[0],
        Size:=TCP_Send_Size
```

```
    //Done=>, Busy=>, Error=>, ErrorID=>
    );

TCP_Rcv(
    Execute:=TCP_Rcv_Exe,
    Socket:=CBT.Socket,
    TimeOut:=TCP_Rcv_TimeOut,
    Size:=TCP_Rcv_Size,
    RcvDat:=TCP_Rcv_Data[0],
    //Done=>, Busy=>, Error=>, ErrorID=>,
    RcvSize=>TCP_Rcv_RcvSize);
```

Annex B: Program example

B1. Pick and Place sequence

```
(* ------------------------------------------------------------------------------
Prg Name:               PnP_ExampleV1.44
Date:                   June 2021
Author:                 Ruben Sanchez Boli
Description:            Pick And Place cycle
(* ------------------------------------------------------------------------------ *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=PnP_Example_Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
    Prg_Step:=1;
END_IF;

IF NOT(PnP_Example_Execute) THEN
    MoveExecute:=FALSE;
    Prg_Step:=0;
END_IF;

IF PnP_Example_Execute THEN
    CASE Prg_Step OF

            0:
                    Prg_Step:=0;
                    MoveExecute:=FALSE;

            1:
                    MoveExecute:=FALSE;
                    Timer_Enable:=FALSE;
                    Prg_Step:=10;

            10:
                    MoveParameters:=PnP_MoveParameters[0];
                    Prg_Step:=11;

            11:
                    MoveExecute:=TRUE;
                    IF MoveDone THEN
                            MoveExecute:=FALSE;
                            Prg_Step:=12;
                    END_IF;


            12:
                    MoveParameters:=PnP_MoveParameters[1];
                    Prg_Step:=13;

            13:
                    MoveExecute:=TRUE;
                    IF MoveDone THEN
                            MoveExecute:=FALSE;
                            Prg_Step:=14;
                    END_IF;


            14:
                    MoveParameters:=PnP_MoveParameters[2];
                    Prg_Step:=15;

            15:
                    MoveExecute:=TRUE;
                    IF MoveDone THEN
                            MoveExecute:=FALSE;
                            Prg_Step:=16;
                    END_IF;

            16:
                    Timer_Enable:=TRUE;
                    IF Timer_1.Q THEN
                            Timer_Enable:=FALSE;
                            Prg_Step:=20;
                    END_IF;


(*-------------------PLACE---------------*)

            20:
                    MoveParameters:=PnP_MoveParameters[3];
                    Prg_Step:=21;

            21:
                    MoveExecute:=TRUE;
                    IF MoveDone THEN
                            MoveExecute:=FALSE;
                            Prg_Step:=22;
                    END_IF;


            22:
                    MoveParameters:=PnP_MoveParameters[4];
                    Prg_Step:=23;

            23:
                    MoveExecute:=TRUE;
                    IF MoveDone THEN
                            MoveExecute:=FALSE;
                            Prg_Step:=24;
                    END_IF;


            24:
                    MoveParameters:=PnP_MoveParameters[5];
                    Prg_Step:=25;

            25:
                    MoveExecute:=TRUE;
                    IF MoveDone THEN
                            MoveExecute:=FALSE;
                            Prg_Step:=26;
                    END_IF;

            26:
                    Timer_Enable:=TRUE;
                    IF Timer_1.Q THEN
                            Timer_Enable:=FALSE;
                            Prg_Step:=10;
                    END_IF;

    ELSE
                            Prg_Step:=0;

    END_CASE;
END_IF;


(* Function Bolcks                                              *)
(* ------------------------------------------------------------  *)

IF PnP_Example_Execute or PnP_MoveBusy or PnP_MoveDone THEN

    PnP_MoveDone := MoveDone;
    PnP_Move_CmdID_Ack := Move_CmdID_Ack;
    PnP_MoveBusy := MoveBusy;
    PnP_MoveError := MoveError;
    PnP_MoveErrorDescrip := MoveErrorDescrip;

    CBT_MoveAbsolute_Instance(
            CBT         :=Cobot,
            Execute     :=MoveExecute,
            Parameters:=MoveParameters,
            Done        =>MoveDone,
            CmdID_Ack=>Move_CmdID_Ack,
            Busy        =>MoveBusy,
            Error       =>MoveError,
            ErrorID     =>MoveErrorID,
            ErrorDescrip=>MoveErrorDescrip);


    Timer_1(In:=Timer_Enable, PT:=T#2000ms, Q=>Timer_Done);

END_IF;
```