Master Program_5

Author: HC Finnson, HC Ringstad, Fredrik Taklo
Created: 20/02/2020 15:08:02
Last Modified: 10/05/2020 15:06:30
Comment:

Sysmac Studio Module Version: 1.29.1.0

Table of Contents

1.IPC

1-3.Controller Setup

1-3-2.Built-in EtherNet/IP Port Settings

Built-in EtherNet/IP Port Settings

TCP/IP Settings	
EtherNet/IP Port - IP Address Settings	Fixed setting
EtherNet/IP Port - IP address	192.168.250.1
EtherNet/IP Port - Subnet mask	255.255.255.0
Internal Port1 - IP address	192.168.254.1
Internal Port1 - Subnet mask	255.255.255.0
Default gateway	192.168.250.240
DNS	Do not use
Priority DNS server	
Secondary DNS server	
Domain name	
Host Name - IP Address	No settings
Keep Alive	Use
Keep Alive monitoring time	300 sec
Linger option	Do not specify
IP Router Table	No settings
IP Forward	Use
NAT	Use
Packet filter	Use
Pass Frame	
No.1 Specification Method	Interface network
No.1 IP Address	
No.1 Mask	
No.1 Protocol	any
No.1 Range specification	False
No.1 Port A	
No.1 Port B	
LINK Settings	
EtherNet/IP Port - LINK settings	Auto
FTP Settings	
FTP server	Do not use
Port No.	21
Login name	
Password	
SNMP Settings	
SNMP service	Do not use
Port No.	161
Address	
Location	
Send a recognition trap.	FALSE
Recognition method	IP address
IP address	0.0.0.0
Host name	
Community name	public
Recognition 2	Do not use
Recognition method	IP address

IP address	0.0.0.0
Host name	
Community name	public
SNMP Trap Settings	
SNMP trap	Do not use
Port No.	162
Specifying method	IP address
IP address	
Host name	
Community name	public
Version	SNMPv2C
Trap 2	Do not use
Specifying method	IP address
IP address	
Host name	
Community name	public
Version	SNMPv2C
CIP Settings	
CIP routing	Use

1-5.Task Settings

Task Name		Period/ Execution Conditions	Detailed Execution Conditions	Task Period Exceeded Detection	Task Timeout Detection Time	Execution Priority	Variable Access Time [%]
Primary periodic task							
	PrimaryTask	2ms		Detect	10ms	4	3
Periodic task							
	Periodic	100ms		Detect	500ms	16	3

Unit	Task Name
EtherCAT Network Configuration	

Task Name		Assigned programs	Initial status
PrimaryTask			
	1	Master	Run
	2	commsCSV	Run
	3	commsLD	Run
	4	commsTM	Run
Periodic			
	1	Background	Run

Variable to be refreshed	Accessing Task
PrimaryTask	
Periodic	

1-6.POUs

1-6-1.Programs

1-6-1-1.Master

1-6-1-1-1.Variables

Name	Data Type	Initial Value	AT	Retain	Constant	Comment
VAR						
orderChosenPrinterNr	INT			False	False	PrinterNr of the first available printer that was found
orderCurrent	ST_OrderDetails			False	False	Struct containing information on the order that is currently being processed
customerQueue	ARRAY[1..100] OF ST_OrderDetails			False	False	Array of orders that have been received while another order was being processed
customerQueueLength	INT			False	False	Length of the customer queue, due to Sysmac being unable to have flexible length arrays.
startedFromQueue	BOOL			False	False	Boolean used to check if the currently processing order was started from the queue or not, to determine if queue updating is necessary
index	INT			False	False	Basic index used for various iteration tasks
connectedPrinters	INT			False	False	Number of connected printers
printerPartsInStore	BOOL			False	False	If the part needed for the order is in stock, then this is TRUE
orderCutterDone	BOOL			False	False	Bool used for storing that the printed part has been collected
orderPrinterDone	BOOL			False	False	Bool used for storing that the lasercut part has been collected
refreshTimer	TON			False	False	General purpose timer, used for waiting before checking connection status rather than spam continuously
VAR_EXTERNAL						
masterState	INT				False	
customerOrder	ST_OrderDetails				False	
csvBools	ST_CsvTools				False	
cutterDxfInfo	ST_CutterDxfExport				False	
printerStatus	ARRAY[0..19] OF ST_3DPrinterStatus				False	
printerCommand	ST_3DPrinterCommand				False	
moviconSendStatus	ST_MoviconStatus				False	
LDCommands	ST_LDTools				False	
moviconSendCommand	ST_MoviconCommands				False	
TMCommands	ST_TMTools				False	
printerActivePrints	ARRAY[0..19] OF ST_3DPrinterCurrentPrintInfo				False	
cutterStatus	ST_CutterStatus				False	
firstBackgroundRunDone	BOOL				False	
cutterProductInfo	ARRAY[1..2] OF ST_CutterDxfExport				True	
printerProductInfo	ARRAY[1..2] OF STRING[256]				True	

	backgroundWritingCommand	BOOL				False	

1-6-1-1-2.Program Body

```
1   //0000------------------------------------------------------
2   // Startup functions
3   IF masterState = 0 THEN
4       // Run startup code. Set initial value for variables and initialize function blocks
5       customerQueueLength := 0;
6       connectedPrinters := 0;
7       refreshTimer.In := FALSE;
8
9       IF firstBackgroundRunDone THEN
10                  masterState := 100;
11      END_IF;
12
13      //masterState := 10000;
14  END_IF;
15  IF masterState = 100 THEN
16      // Check cutter for connection/errors. If no errors, proceed.
17      IF cutterStatus.done='True' THEN
18                  masterState := 200;
19      ELSIF cutterStatus.error='True' THEN
20                  masterState := 9010;
21      END_IF;
22  END_IF;
23  IF masterState = 200 THEN
24      // Check how many printers are connected. If at least 1, proceed.
25      FOR index := 0 TO SizeOfAry(printerStatus)-1 DO
26                  IF printerStatus[index].connected='True' THEN
27                          connectedPrinters := connectedPrinters + 1;
28                  END_IF;
29      END_FOR;
30      IF connectedPrinters > 0 THEN
31                  //1 or more printers available, proceed.
32                  masterState := 300;
33      ELSIF connectedPrinters = 0 THEN
34                  //No printer available, error
35                  masterState := 9020;
36      END_IF;
37  END_IF;
38  IF masterState = 300 THEN
39      //Connect to LD robots. If successful, proceed, if not, throw error.
40      LDCommands.connect := TRUE;
41      IF LDCommands.connectSuccessful THEN
42                  masterState := 400;
43      ELSIF LDCommands.connectError THEN
44                  masterState := 9030;
45      END_IF;
46  END_IF;
47  IF masterState = 400 THEN
48      //Connect to TM robots. If successful, proceed, if not, throw error.
49      TMCommands.connect := TRUE;
50      IF TMCommands.connectSuccessful THEN
51                  masterState := 500;
52      ELSIF TMCommands.connectError THEN
53                  masterState := 9040;
54      END_IF;
55  END_IF;
56
```

```
57      IF masterState = 500 THEN
58          // Check viper, quattro, etc, then go to idle
59
60          masterState := 1000;
61      END_IF;
62
63      //1000----------------------------------------------------------------------------------------------------------
------------------------
64      // Idle, awaiting order
65      IF masterState = 1000 THEN
66
67          IF customerOrder.orderReceived THEN
68                      //Order is received from Movicon, copy information to local variable for processing and reset
customerOrder to prevent a loop
69                      orderCurrent := customerOrder;
70                      customerOrder.orderReceived := FALSE;
71                      customerOrder.productID := 0;
72                      customerOrder.name := '';
73                      masterState := 2000;
74          ELSIF NOT(customerOrder.orderReceived) AND customerQueue[1].orderReceived THEN
75                      // There is no new order from Movicon, but the first position in the queue is filled with an order.
76                      // Copy the order from the first spot in the queue into the local variable for processing. Signify
that this was an order started from the queue.
77                      orderCurrent := customerQueue[1];
78                      startedFromQueue := TRUE;
79                      masterState := 2000;
80          END_IF;
81
82      END_IF;
83
84      // Add order to the back of the queue when a new order is received while the program is busy processing
another.
85      // Reset customerOrder after completing to open up for a new order if one comes
86      // Note, this always runs in the background.
87      IF NOT(masterState = 1000) AND customerOrder.orderReceived THEN
88          customerQueueLength := customerQueueLength + 1;
89          customerQueue[customerQueueLength] := customerOrder;
90          customerOrder.orderReceived := FALSE;
91          customerOrder.productID := 0;
92          customerOrder.name := '';
93      END_IF;
94
95
96      //2000----------------------------------------------------------
97      //Process customer order
98      // 2000 = setup
99      // 2100 = process dxf for cutter
100     // 2200 = check kanban storage for printer
101
102     IF masterState = 2000 THEN
103         // Setup
104         masterState := 2100;
105     END_IF;
106     IF masterState = 2100 THEN
107         // Cutter
108         // Depending on the product ID, either apply the preset for the nameplate or the keychain for export to
Dxf, and add the customer's name to it.
109         IF orderCurrent.productID > 0 AND orderCurrent.productID <= SizeOfAry(cutterProductInfo) THEN
110                     cutterDxfInfo := cutterProductInfo[orderCurrent.productID];
```

```
111                 cutterDxfInfo.textToInsert := orderCurrent.name;
112                 masterState := 2200;
113
114         ELSE
115                 masterState := 9210;
116         END_IF;
117     END_IF;
118     IF masterState = 2200 THEN
119         //Printer
120         // Check the kanban storage to see if the required part is in storage.
121         CASE orderCurrent.productID OF
122         1: //Bracket pairs
123                 IF moviconSendStatus.ID1BracketPairNumber > 0 THEN
124                         printerPartsInStore := TRUE;
125                         masterState := 3000;
126                 ELSIF moviconSendStatus.ID1BracketPairNumber = 0 THEN
127                         printerPartsInStore := FALSE;
128                         masterState := 3000;
129                 ELSE
130                         masterState := 9221;
131                 END_IF;
132         2: //Keychain frame
133                 IF moviconSendStatus.ID2KeychainNumber > 0 THEN
134                         printerPartsInStore := TRUE;
135                         masterState := 3000;
136                 ELSIF moviconSendStatus.ID2KeychainNumber = 0 THEN
137                         printerPartsInStore := FALSE;
138                         masterState := 3000;
139                 ELSE
140                         masterState := 9222;
141                 END_IF;
142         //Fill in new products here
143         ELSE
144                 //Non-existent product ID
145                 masterState := 9220;
146         END_CASE;
147     END_IF;
148
149
150     //3000------------------------------------------------------
151     // Assign work to cutter, printer, robots, etc
152     // 3000 = setup
153     // 3100 = assign cutter task
154     // 3200 = assign printer task
155     // 3300 = assign LD robots
156     // 3400 = assign TM robots
157     IF masterState = 3000 THEN
158         //Setup
159         orderChosenPrinterNr := -1;
160         masterState := 3100;
161     END_IF;
162     IF masterState = 3100 THEN
163         //Cutter
164         //With the information added to the dxfInfo variable and ready for export, start the csv conversion
program
165         //After that is done, tell Movicon to start the Python script
166         IF NOT(csvBools.writeCutterDxfStart) AND NOT(csvBools.writeCutterDxfDone) THEN
167                 csvBools.writeCutterDxfStart := TRUE;
168         END_IF;
```

```
169         IF csvBools.writeCutterDxfStart AND csvBools.writeCutterDxfDone THEN
170                 csvBools.writeCutterDxfStart := FALSE;
171                 moviconSendCommand.startDxfScript := TRUE; //Note, needs to be turned off at some point.
Could do in cleanup
172                 masterState := 3200;
173         END_IF;
174     END_IF;
175     IF masterState = 3200 THEN
176         //Printer
177         //Regardless of whether needed part is or is not in store, print a new one.
178         //Iterate through the array of printer statuses to find the first printer that is ready and available to use.
179         orderChosenPrinterNr := FindAvailablePrinter(statusArray:=printerStatus);
180         IF orderChosenPrinterNr = -1 THEN
181                 //If no available printer was found, go to error.
182                 masterState := 9320;
183         ELSE
184                 //Otherwise, continue as planned
185                 masterState := 3210;
186         END_IF;
187     END_IF;
188     IF masterState = 3210 AND NOT(backgroundWritingCommand) THEN
189         //Now that an available printer has been found, create a command to send to the printer.
190         printerCommand.separator := 'sep='; //Ensures that the file can be opened in excel if required
191         printerCommand.IP := CONCAT(In1:='$R$L', In2:=printerStatus[orderChosenPrinterNr].IP); //$R$L is
newline
192         printerCommand.command := 'print';
193         printerCommand.number := '1';
194         printerCommand.argument := printerProductInfo[orderCurrent.productID];
195         printerActivePrints[orderChosenPrinterNr].productID := orderCurrent.productID;
196         printerActivePrints[orderChosenPrinterNr].number := STRING_TO_UINT(printerCommand.number);
197         CASE orderCurrent.productID OF
198                 1:
199                         moviconSendStatus.ID1BracketPairPrinting :=
moviconSendStatus.ID1BracketPairPrinting + 1;
200                 2:
201                         moviconSendStatus.ID2KeychainPrinting := moviconSendStatus.ID2KeychainPrinting +
1;
202         END_CASE;
203         masterState := 3220;
204     END_IF;
205     IF masterState = 3220 AND NOT(backgroundWritingCommand) THEN
206         //Run the csv conversion program to put the previously created printer command into a csv, then tell
Movicon to run the python script
207         IF NOT(csvBools.writePrinterCommandStart) AND NOT(csvBools.writePrinterCommandDone) THEN
208                 csvBools.writePrinterCommandStart := TRUE;
209         END_IF;
210         IF csvBools.writePrinterCommandStart AND csvBools.writePrinterCommandDone THEN
211                 csvBools.writePrinterCommandStart := FALSE;
212                 moviconSendCommand.startPrinterScript := TRUE; //Note, needs to be turned off at some point,
could do in cleanup
213                 masterState := 3300;
214         END_IF;
215     END_IF;
216     IF masterState = 3300 THEN
217         // LD robots
218         IF printerPartsInStore THEN
219                 //Command LD robot to pick up parts
220                 //May use CASE if rack is too large to only pick from one spot
```

```
221                 //Could push to 4300?
222                 LDCommands.productID := orderCurrent.productID;
223                 LDCommands.getFromKanbanStart := TRUE;
224                 masterState := 4000;
225         ELSIF NOT(printerPartsInStore) THEN
226                 //Part needed is not in store
227                 masterState := 9330; //Temporary error state, wait until printing is done and kanban has the
given part.
228         END_IF;
229     END_IF;
230
231     //4000------------------------------------------------------------
232     // Wait for processes to complete, then continue production
233     // Pick up the different parts and move to assembly table
234     // 4000 initial setup
235     // 4100 wait for cutter to be done, then flag it as done
236     // 4200
237     // 4300 LD tasks. wait for LD robot to arrive, then order TM robot to do things
238     // 4400 send parts to assembly? may be a part of 4300
239     IF masterState = 4000 THEN
240         masterState := 4100;
241     END_IF;
242     IF masterState = 4100 THEN
243         //Cutter
244         //Cutter status csv is being continuously read/refreshed in Background. Wait until it reads that the cutting
is done
245         IF cutterStatus.done='True' THEN
246                 //Cutter is done. Move to next task
247                 orderCutterDone := TRUE;
248                 masterState := 4200;
249         ELSIF cutterStatus.error='True' THEN
250                 //Error happened, go to appropriate state.
251                 masterState := 9410;
252         END_IF;
253     END_IF;
254     IF masterState = 4200 THEN
255         //3D Printer, what to do depends on parts being in stock or not
256         IF LDCommands.getFromKanbanDone THEN
257                 orderPrinterDone := TRUE;
258                 LDCommands.getFromKanbanStart := FALSE;
259                 IF orderCurrent.productID=1 THEN
260                         moviconSendStatus.ID1BracketPairNumber :=
moviconSendStatus.ID1BracketPairNumber - 1;
261                 ELSIF orderCurrent.productID=2 THEN
262                         moviconSendStatus.ID2KeychainNumber := moviconSendStatus.ID2KeychainNumber -
1;
263                 END_IF;
264
265                 masterState := 4300;
266         END_IF;
267     END_IF;
268     IF masterState = 4300 THEN
269         //LD robots
270         masterState := 4310;
271     END_IF;
272     IF masterState = 4310 THEN
273         //Retreive item from cutter
274         IF orderCutterDone AND NOT(LDCommands.getFromKanbanStart) AND NOT
(LDCommands.getFromCutterDone) THEN
```

```
275                 LDCommands.productID := orderCurrent.productID;
276                 LDCommands.getFromCutterStart := TRUE;
277         END_IF;
278         IF orderCutterDone AND LDCommands.getFromCutterStart AND LDCommands.getFromCutterDone
THEN
279                 LDCommands.getFromCutterStart := FALSE;
280                 masterState := 5000;
281         END_IF;
282     END_IF;
283
284     //5000----------------------------------------------------------
285     // Order assembly and delivery
286     IF masterState = 5000 THEN
287         TMCommands.productID := orderCurrent.productID;
288
289         TMCommands.arm1_jobNr := orderCurrent.productID;
290         TMCommands.arm1_jobStart := TRUE;
291
292         TMCommands.arm2_jobNr := orderCurrent.productID;
293         TMCommands.arm2_jobStart := TRUE;
294
295         masterState := 5100;
296     END_IF;
297     IF masterState = 5100 THEN
298         IF TMCommands.arm1_jobDone AND TMCommands.arm2_jobDone THEN
299                 TMCommands.arm1_jobStart := FALSE;
300                 TMCommands.arm2_jobStart := FALSE;
301                 TMCommands.productID := 0;
302                 LDCommands.productID := orderCurrent.productID;
303                 LDCommands.deliverProductStart := TRUE;
304                 masterState := 5200;
305         END_IF;
306     END_IF;
307
308     IF masterState = 5200 THEN
309         IF LDCommands.deliverProductDone THEN
310                 LDCommands.deliverProductStart := FALSE;
311                 masterState := 6000;
312         END_IF;
313     END_IF;
314
315
316     //6000----------------------------------------------------------
317     // Update status and clean up
318
319     IF masterState = 6000 THEN
320         //Reset local variables
321         printerPartsInStore := FALSE;
322         orderCutterDone := FALSE;
323         orderPrinterDone := FALSE;
324         moviconSendCommand.startDxfScript := FALSE;
325         moviconSendCommand.startPrinterScript := FALSE;
326         orderCurrent.orderReceived := FALSE;
327         orderCurrent.productID := 0;
328         orderCurrent.name := '';
329
330         IF startedFromQueue THEN
331                 masterState := 6050;
332         ELSIF NOT(startedFromQueue) THEN
```

```
333                 masterState := 6100;
334         END_IF;
335     END_IF;
336     IF masterState = 6050 THEN
337         //If the current process was started from a queue, then move all spots in the queue down one level and
reset the last.
338         FOR index := 1 TO customerQueueLength-1 DO
339                 customerQueue[index] := customerQueue[index+1];
340         END_FOR;
341         customerQueue[customerQueueLength].orderReceived := FALSE;
342         customerQueue[customerQueueLength].productID := 0;
343         customerQueue[customerQueueLength].name := '';
344         customerQueueLength := customerQueueLength - 1;
345         startedFromQueue := FALSE;
346 
347         //Queue process complete, continue with order cleanup
348         masterState := 6100;
349     END_IF;
350     IF masterState = 6100 THEN
351         //Cleanup complete, return to idle
352         masterState := 1000;
353     END_IF;
354 
355 
356     //9000-------------------------------------------------------
357     // Error handling
358     // Actual errors, handling "no printers/cutters available"/"can't complete task", etc
359     // 9X00, replace X with whatever X000 master state the program was in when error occured.
360     // 90Y0, replace Y with whatever 0Y00 master state the program was in when error occured.
361     // 900Z, replace Z with whatever 00Z0 master state the program was in when error occured.
362 
363     IF masterState = 9010 THEN
364         //Cutter is unable to connect
365         //Send to Movicon?
366         //Wait a while, then set master state to 100 to try again.
367         //Maybe send "Unrecoverable error" status to Movicon if this happens too many times
368         IF NOT(refreshTimer.In) THEN
369                 refreshTimer.In := TRUE;
370                 refreshTimer.PT := T#5s;
371         END_IF;
372         IF refreshTimer.In AND refreshTimer.Q THEN
373                 refreshTimer.In := FALSE;
374                 masterState := 100;
375         END_IF;
376     END_IF;
377     IF masterState = 9020 THEN
378         //No printers available in rack
379         //Send to movicon?
380         //Wait a while, then set master state to 200 to try again
381         //Maybe send "Unrecoverable error" status to Movicon if this happens too many times
382         IF NOT(refreshTimer.In) THEN
383                 refreshTimer.In := TRUE;
384                 refreshTimer.PT := T#5s;
385         END_IF;
386         IF refreshTimer.In AND refreshTimer.Q THEN
387                 refreshTimer.In := FALSE;
388                 masterState := 200;
389         END_IF;
390     END_IF;
```

```
391  IF masterState = 9030 THEN
392      //Unable to connect to LDs
393      //Send to movicon?
394      //Wait a while, then set master state to 300.
395      //Maybe send "Unrecoverable error" status to Movicon if this happens too many times
396      IF NOT(refreshTimer.In) THEN
397              refreshTimer.In := TRUE;
398              refreshTimer.PT := T#5s;
399              LDCommands.connect := FALSE;
400      END_IF;
401      IF refreshTimer.In AND refreshTimer.Q THEN
402              refreshTimer.In := FALSE;
403              masterState := 300;
404      END_IF;
405  END_IF;
406  IF masterState = 9040 THEN
407      //Unable to connect to TMs
408      //Send to movicon?
409      //Wait a while, then set master state to 300.
410      //Maybe send "Unrecoverable error" status to Movicon if this happens too many times
411      IF NOT(refreshTimer.In) THEN
412              refreshTimer.In := TRUE;
413              refreshTimer.PT := T#5s;
414              TMCommands.connect := FALSE;
415      END_IF;
416      IF refreshTimer.In AND refreshTimer.Q THEN
417              refreshTimer.In := FALSE;
418              masterState := 400;
419      END_IF;
420  END_IF;
421
422  IF masterState = 9210 THEN
423      //Invalid product ID from cutter, return to idle.
424      masterState := 1000;
425  END_IF;
426  IF masterState = 9220 THEN
427      //Invalid product ID from printer, return to idle.
428      masterState := 1000;
429  END_IF;
430  IF masterState = 9221 THEN
431      //Bracket-pair-in-storage number is negative.
432      //Setting to 0 should solve it and force a new print.
433      moviconSendStatus.ID1BracketPairNumber := 0;
434      masterState := 2200;
435  END_IF;
436  IF masterState = 9222 THEN
437      //Keychain frame in storage number is negative.
438      //Setting to 0 should solve it and force a new print.
439      moviconSendStatus.ID2KeychainNumber := 0;
440      masterState := 2200;
441  END_IF;
442
443  IF masterState = 9320 THEN
444      //No available printer is found, do something about this.
445      //Set printer text to "no available printer"? Prompt asking whether to cancel or wait?
446      //For now, loop printer search until a printer becomes available.
447      IF NOT(refreshTimer.In) THEN
448              refreshTimer.In := TRUE;
449              refreshTimer.PT := T#1s;
```

```
450     END_IF;
451     IF refreshTimer.In AND refreshTimer.Q THEN
452             refreshTimer.In := FALSE;
453             orderChosenPrinterNr := FindAvailablePrinter(statusArray:=printerStatus);
454             IF NOT(orderChosenPrinterNr = -1) THEN
455                     masterState := 3210;
456             END_IF;
457     END_IF;
458 END_IF;
459 IF masterState = 9330 THEN
460     //Part required to pick up is not in stock. Wait until part has finished printing and added to storage
461     //Ask in movicon whether the customer wants to wait or cancel in the meantime?
462     IF orderCurrent.productID = 1 AND moviconSendStatus.ID1BracketPairNumber > 0 THEN
463             LDCommands.getFromKanbanStart := TRUE;
464             LDCommands.productID := orderCurrent.productID;
465             printerPartsInStore := TRUE;
466             masterState := 4000;
467     ELSIF orderCurrent.productID = 2 AND moviconSendStatus.ID2KeychainNumber > 0 THEN
468             LDCommands.getFromKanbanStart := TRUE;
469             LDCommands.productID := orderCurrent.productID;
470             printerPartsInStore := TRUE;
471             masterState := 4000;
472     END_IF;
473 END_IF;
474
475 IF masterState = 9410 THEN
476     //Cutter has given off an error before completing.
477     //Cannot be fixed without access to physical hardware to know what I can and cannot do about it    .
478     masterState := 9410;
479 END_IF;
```

1-6-1-2.Background

1-6-1-2-1.Variables

Name	Data Type	Initial Value	AT	Retain	Constant	Comment
VAR						
index	INT			False	False	Basic index used for various iteration tasks
printerTempArray	ARRAY[0..19] OF INT			False	False	Temporary array to store information while it's converted to a format that Movicon is capable of reading (can't do arrays or structures inside structures)
printerNr	INT			False	False	Integer containing the number of the currently selected, available printer
partsToPrint	INT			False	False	Number of parts that needs to be printed, ranging from 1-5
firstCutterReadDone	BOOL			False	False	Cutter status has been successfully read for the first time.
firstPrinterReadDone	BOOL			False	False	Printer status has been successfully read for the first time.
VAR_EXTERNAL						
printerStatus	ARRAY[0..19] OF ST_3DPrinterStatus				False	
moviconSendStatus	ST_MoviconStatus				False	
printerActivePrints	ARRAY[0..19] OF ST_3DPrinterCurrentPrintInfo				False	
printerCommand	ST_3DPrinterCommand				False	
moviconSendCommand	ST_MoviconCommands				False	
csvBools	ST_CsvTools				False	
firstBackgroundRunDone	BOOL				False	
printerProductInfo	ARRAY[1..2] OF STRING[256]				True	
masterState	INT				False	
LDCommands	ST_LDTools				False	
LDTakloStatus	ST_LDStatus				False	
LDFinnsonStatus	ST_LDStatus				False	
LDRingstadStatus	ST_LDStatus				False	
backgroundWritingCommand	BOOL				False	
TMCommands	ST_TMTools				False	
cutterStatus	ST_CutterStatus				False	

1-6-1-2-2.Program Body

```
1     //Reset a few write commands once they are completed.
2     IF csvBools.writeCutterDxfStart AND csvBools.writeCutterDxfDone THEN
3         csvBools.writeCutterDxfStart := FALSE;
4     END_IF;
5     IF csvBools.writePrinterCommandStart AND csvBools.writePrinterCommandDone THEN
6         csvBools.writePrinterCommandStart := FALSE;
7     END_IF;
8
9     //Refresh cutter status
10    IF NOT(csvBools.readCutterStatusStart) AND NOT(csvBools.readCutterStatusDone) THEN
11        csvBools.readCutterStatusStart := TRUE;
12    END_IF;
13    IF csvBools.readCutterStatusStart AND csvBools.readCutterStatusDone THEN
14        csvBools.readCutterStatusStart := FALSE;
15        firstCutterReadDone := TRUE;
16    END_IF;
17
18    //Refresh printer status
19    IF NOT(csvBools.readPrinterStatusStart) AND NOT(csvBools.readPrinterStatusDone) THEN
20        csvBools.readPrinterStatusStart := TRUE;
21    END_IF;
22    IF csvBools.readPrinterStatusStart AND csvBools.readPrinterStatusDone THEN
23        csvBools.readPrinterStatusStart := FALSE;
24        firstPrinterReadDone := TRUE;
25    END_IF;
26
27    //Initial run of background program done
28    IF firstCutterReadDone AND firstPrinterReadDone THEN
29        firstBackgroundRunDone := TRUE;
30    END_IF;
31
32    IF firstBackgroundRunDone THEN
33        //Update list of current prints
34        FOR index := 0 TO SizeOfAry(printerStatus)-1 DO
35                IF printerStatus[index].printing='True' OR printerStatus[index].finished='True' THEN
36                        //Something is being printed in this printer, update status array on the given index.
37                        SubDelimiter(In:=ProcessGcodeName(gcodeName:=printerStatus[index].printJob),
OutStruct:=printerActivePrints[index], Delimiter:=_eDELIMITER#_SEMICOLON);
38                ELSIF printerStatus[index].printing='False' AND printerStatus[index].finished='False' THEN
39                        //Nothing is being printed here, update status.
40                        printerActivePrints[index].productID := 0;
41                        printerActivePrints[index].number := 0;
42                END_IF;
43        END_FOR;
44
45
46        //Confirm that buffer doesn't use unrealistic values, correct if it does
47        IF moviconSendStatus.ID1BracketPairBuffer > 20 THEN
48                moviconSendStatus.ID1BracketPairBuffer := 20;
49        ELSIF moviconSendStatus.ID1BracketPairBuffer < 0 THEN
50                moviconSendStatus.ID1BracketPairBuffer := 0;
51        END_IF;
52        IF moviconSendStatus.ID2KeychainBuffer > 20 THEN
53                moviconSendStatus.ID2KeychainBuffer := 20;
54        ELSIF moviconSendStatus.ID2KeychainBuffer < 0 THEN
55                moviconSendStatus.ID2KeychainBuffer := 0;
56        END_IF;
57
```

```
58          // Check if printer's bed and nozzle temperatures aren't too high. If yes, tell printer to do an emergency
shutdown.
59          // Protect against malicious or stupid intentions causing high enough temperatures to damage or
destroy equipment
60          FOR printerNr := 0 TO SizeOfAry(printerStatus)-1 DO
61                  IF STRING_TO_INT(printerStatus[printerNr].bedTemp) > 100 OR STRING_TO_INT(printerStatus
[printerNr].nozzleTemp) > 400 AND NOT(masterState = 3210) AND NOT(masterState = 3220) THEN
62                          //Printer with too large temperature detected
63                          printerCommand.separator := 'sep=';
64                          printerCommand.IP := CONCAT(In1:='$R$L', In2:=printerStatus[printerNr].IP);
65                          printerCommand.command := 'shutdown';
66                          //printerCommand.argument := 'emergency';
67                          csvBools.writePrinterCommandStart := TRUE;
68                  END_IF;
69          END_FOR;
70
71          // Check if a printer has finished printing. If yes, bring the part to kanban storage
72          FOR printerNr := SizeOfAry(printerStatus)-1 TO 0 BY -1 DO
73                  IF (printerStatus[printerNr].finished = 'True') AND NOT(LDCommands.getFromPrinterStart) AND
NOT(LDCommands.getFromPrinterDone) THEN
74                          // This printer has finished a print and hasn't been retrieved.
75                          LDCommands.getFromPrinterStart := TRUE;
76                          LDCommands.printerNr := index;
77                  END_IF;
78                  IF (printerStatus[printerNr].finished = 'True') AND LDCommands.getFromPrinterStart AND
LDCommands.getFromPrinterDone THEN
79                          //The print has been picked up
80                          CASE printerActivePrints[printerNr].productID OF
81                                  1:
82                                          //Bracket pair was picked up
83                                          moviconSendStatus.ID1BracketPairNumber :=
moviconSendStatus.ID1BracketPairNumber + printerActivePrints[printerNr].number;
84                                          moviconSendStatus.ID1BracketPairPrinting :=
moviconSendStatus.ID1BracketPairPrinting - printerActivePrints[printerNr].number;
85                                  2:
86                                          //Keychain frame was picked up
87                                          moviconSendStatus.ID2KeychainNumber :=
moviconSendStatus.ID2KeychainNumber + printerActivePrints[printerNr].number;
88                                          moviconSendStatus.ID2KeychainPrinting :=
moviconSendStatus.ID2KeychainPrinting - printerActivePrints[printerNr].number;
89                                  ELSE
90                                          //Error. Placeholder code to stop compile error.
91                                          moviconSendStatus.ID1BracketPairNumber := 999;
92                          END_CASE;
93
94                          //Tell printer that the part is picked up and that the printer is ready for use again.
Doing this will set finished to False.
95                          printerCommand.separator := 'sep=';
96                          printerCommand.IP := CONCAT(In1:='$R$L', In2:=printerStatus[printerNr].IP);
97                          printerCommand.command := 'retrievedPrint';
98                          csvBools.writePrinterCommandStart := TRUE;
99                          LDCommands.getFromPrinterStart := FALSE;
100
101                         //To prevent issues for another finished print, exit for-loop to prevent further finished
prints from being found.
102                         //This way, it's guaranteed to only work on one print at a time.
103                         EXIT;
104
105                 END_IF;
```

```
106         END_FOR;
107
108
109         //Check bracket buffer, print up to 5 new bracket pairs (per printer) if current stock is smaller than buffer
110         IF (moviconSendStatus.ID1BracketPairNumber+moviconSendStatus.ID1BracketPairPrinting) <
moviconSendStatus.ID1BracketPairBuffer AND NOT(masterState = 3210) AND NOT(masterState = 3220) THEN
111                 IF NOT(csvBools.writePrinterCommandStart) AND NOT(csvBools.writePrinterCommandDone)
THEN
112                         //IF statements ensure that no other part of the program is currently trying to write to
the same file
113                         printerNr := -1; //Set unreachable default value to check for no available printer
114                         printerNr := FindAvailablePrinter(statusArray:=printerStatus);
115                         IF NOT(printerNr = -1) THEN
116                                 //If there is an available printer, use start sending write commands to it
117                                 printerCommand.separator := 'sep=';
118                                 printerCommand.IP := CONCAT(In1:='$R$L', In2:=printerStatus[printerNr].IP);
119                                 printerCommand.command := 'print';
120                                 printerCommand.argument := printerProductInfo[1];
121                                 //Find number of parts required left to print, counting current in storage and
currently printing
122                                 partsToPrint := moviconSendStatus.ID1BracketPairBuffer -
moviconSendStatus.ID1BracketPairNumber - moviconSendStatus.ID1BracketPairPrinting;
123                                 //Ensure that no more than 5 are printed. If more than 5 parts are needed,
then 5 are subtracted and the next 1-5 will be printed next time Background loops.
124                                 printerCommand.number := INT_TO_STRING(LIMIT(MN:=INT#1,
In:=partsToPrint, MX:=INT#5));
125                                 csvBools.writePrinterCommandStart := TRUE;
126                         END_IF;
127                 END_IF;
128                 IF csvBools.writePrinterCommandStart AND csvBools.writePrinterCommandDone THEN
129                         //Note, method for adding to local database of prints removed because it now updates
in the top.
130                         moviconSendStatus.ID1BracketPairPrinting :=
moviconSendStatus.ID1BracketPairPrinting + STRING_TO_INT(printerCommand.number);
131                         moviconSendCommand.startPrinterScript := TRUE; //Note, must be turned off
eventually
132                         csvBools.writePrinterCommandStart := FALSE;
133                 END_IF;
134         //Check keychain buffer, print up to 5 new keychain frames (per printer) if current stock is smaller than
buffer. ELSIF to stop errors with the IF statement above
135         ELSIF (moviconSendStatus.ID2KeychainNumber+moviconSendStatus.ID2KeychainPrinting) <
moviconSendStatus.ID2KeychainBuffer AND NOT(masterState = 3210) AND NOT(masterState = 3220) THEN
136                 IF NOT(csvBools.writePrinterCommandStart) AND NOT(csvBools.writePrinterCommandDone)
THEN
137                         printerNr := FindAvailablePrinter(statusArray:=printerStatus);
138                         IF NOT(printerNr = -1) THEN
139                                 //If there is an available printer, start sending write commands to it
140                                 printerCommand.separator := 'sep=';
141                                 printerCommand.IP := CONCAT(In1:='$R$L', In2:=printerStatus[printerNr].IP);
142                                 printerCommand.command := 'print';
143                                 printerCommand.argument := printerProductInfo[2];
144                                 //Find number of parts required left to print, counting current in storage and
currently printing
145                                 partsToPrint := moviconSendStatus.ID2KeychainBuffer -
moviconSendStatus.ID2KeychainNumber - moviconSendStatus.ID2KeychainPrinting;
146                                 //Ensure that no more than 5 are printed. If more than 5 parts are needed,
then 5 are subtracted and the next 1-5 will be printed next time Background loops.
147                                 printerCommand.number := INT_TO_STRING(LIMIT(MN:=INT#1,
In:=partsToPrint, MX:=INT#5));
```

```
148                                 csvBools.writePrinterCommandStart := TRUE;
149                                 backgroundWritingCommand := TRUE;
150                         END_IF;
151                 END_IF;
152                 IF csvBools.writePrinterCommandStart AND csvBools.writePrinterCommandDone THEN
153                         moviconSendStatus.ID2KeychainPrinting := moviconSendStatus.ID2KeychainPrinting +
STRING_TO_INT(printerCommand.number);
154                         moviconSendCommand.startPrinterScript := TRUE; //Note, must be turned off
eventually
155                         csvBools.writePrinterCommandStart := FALSE;
156                         backgroundWritingCommand := FALSE;
157                 END_IF;
158         END_IF;
159
160
161         //-------------------------------------------------------------------------------
162
163         //Send updated statuses to Movicon
164         //Master state
165         moviconSendStatus.masterState := masterState;
166
167         //LD robots
168         IF LDFinnsonStatus.connected AND NOT(LDFinnsonStatus.busy) THEN
169                 moviconSendStatus.mobileRobot01 := 1;
170         END_IF;
171         IF LDFinnsonStatus.busy THEN
172                 moviconSendStatus.mobileRobot01 := 2;
173         END_IF;
174         IF LDFinnsonStatus.error OR NOT(LDFinnsonStatus.connected) THEN
175                 moviconSendStatus.mobileRobot01 := 0;
176         END_IF;
177
178         IF LDTakloStatus.connected AND NOT(LDTakloStatus.busy) THEN
179                 moviconSendStatus.mobileRobot00 := 1;
180         END_IF;
181         IF LDTakloStatus.busy THEN
182                 moviconSendStatus.mobileRobot00 := 2;
183         END_IF;
184         IF LDTakloStatus.error OR NOT(LDTakloStatus.connected) THEN
185                 moviconSendStatus.mobileRobot00 := 0;
186         END_IF;
187
188         IF LDRingstadStatus.connected AND NOT(LDRingstadStatus.busy) THEN
189                 moviconSendStatus.mobileRobot02 := 1;
190         END_IF;
191         IF LDRingstadStatus.busy THEN
192                 moviconSendStatus.mobileRobot02 := 2;
193         END_IF;
194         IF LDRingstadStatus.error OR NOT(LDRingstadStatus.connected) THEN
195                 moviconSendStatus.mobileRobot02 := 0;
196         END_IF;
197
198         //TM Arms
199         IF TMCommands.connectSuccessful AND NOT(TMCommands.armLD_jobStart) THEN
200                 moviconSendStatus.cobot00 := 1;
201         END_IF;
202         IF TMCommands.connectSuccessful AND NOT(TMCommands.arm1_jobStart) THEN
203                 moviconSendStatus.cobot01 := 1;
204         END_IF;
```

```
205     IF TMCommands.connectSuccessful AND NOT(TMCommands.arm2_jobStart) THEN
206             moviconSendStatus.cobot02 := 1;
207     END_IF;
208     IF TMCommands.armLD_jobStart THEN
209             moviconSendStatus.cobot00 := 2;
210     END_IF;
211     IF TMCommands.arm1_jobStart THEN
212             moviconSendStatus.cobot01 := 2;
213     END_IF;
214     IF TMCommands.arm2_jobStart THEN
215             moviconSendStatus.cobot02 := 2;
216     END_IF;
217     IF TMCommands.connectError OR NOT(TMCommands.connectSuccessful) THEN
218             moviconSendStatus.cobot00 := 0;
219             moviconSendStatus.cobot01 := 0;
220             moviconSendStatus.cobot02 := 0;
221     END_IF;
222
223     //Cutter
224     IF cutterStatus.done='True' AND cutterStatus.working='False' THEN
225             moviconSendStatus.laserCutter00 := 1;
226     END_IF;
227     IF cutterStatus.working='True' THEN
228             moviconSendStatus.laserCutter00 := 2;
229     END_IF;
230     IF cutterStatus.error='True' THEN
231             moviconSendStatus.laserCutter00 := 0;
232     END_IF;
233
234     //Printers
235     FOR index :=0 TO SizeOfAry(printerStatus)-1 DO
236             //Update printer status
237             // 0 = error
238             // 1 = idle
239             // 2 = printing
240             IF printerStatus[index].ready='True' THEN
241                     printerTempArray[index] := 1;
242             END_IF;
243             IF printerStatus[index].printing='True' OR printerStatus[index].finished='True' THEN
244                     printerTempArray[index] := 2;
245             END_IF;
246             IF printerStatus[index].connected='False' OR printerStatus[index].operational='False' OR
printerStatus[index].pausing='True' OR printerStatus[index].paused='True' THEN
247                     printerTempArray[index] := 0;
248             END_IF;
249     END_FOR;
250     moviconSendStatus.printer00 := printerTempArray[0]; moviconSendStatus.printer10 := printerTempArray
[10];
251     moviconSendStatus.printer01 := printerTempArray[1]; moviconSendStatus.printer11 := printerTempArray
[11];
252     moviconSendStatus.printer02 := printerTempArray[2]; moviconSendStatus.printer12 := printerTempArray
[12];
253     moviconSendStatus.printer03 := printerTempArray[3]; moviconSendStatus.printer13 := printerTempArray
[13];
254     moviconSendStatus.printer04 := printerTempArray[4]; moviconSendStatus.printer14 := printerTempArray
[14];
255     moviconSendStatus.printer05 := printerTempArray[5]; moviconSendStatus.printer15 := printerTempArray
[15];
```

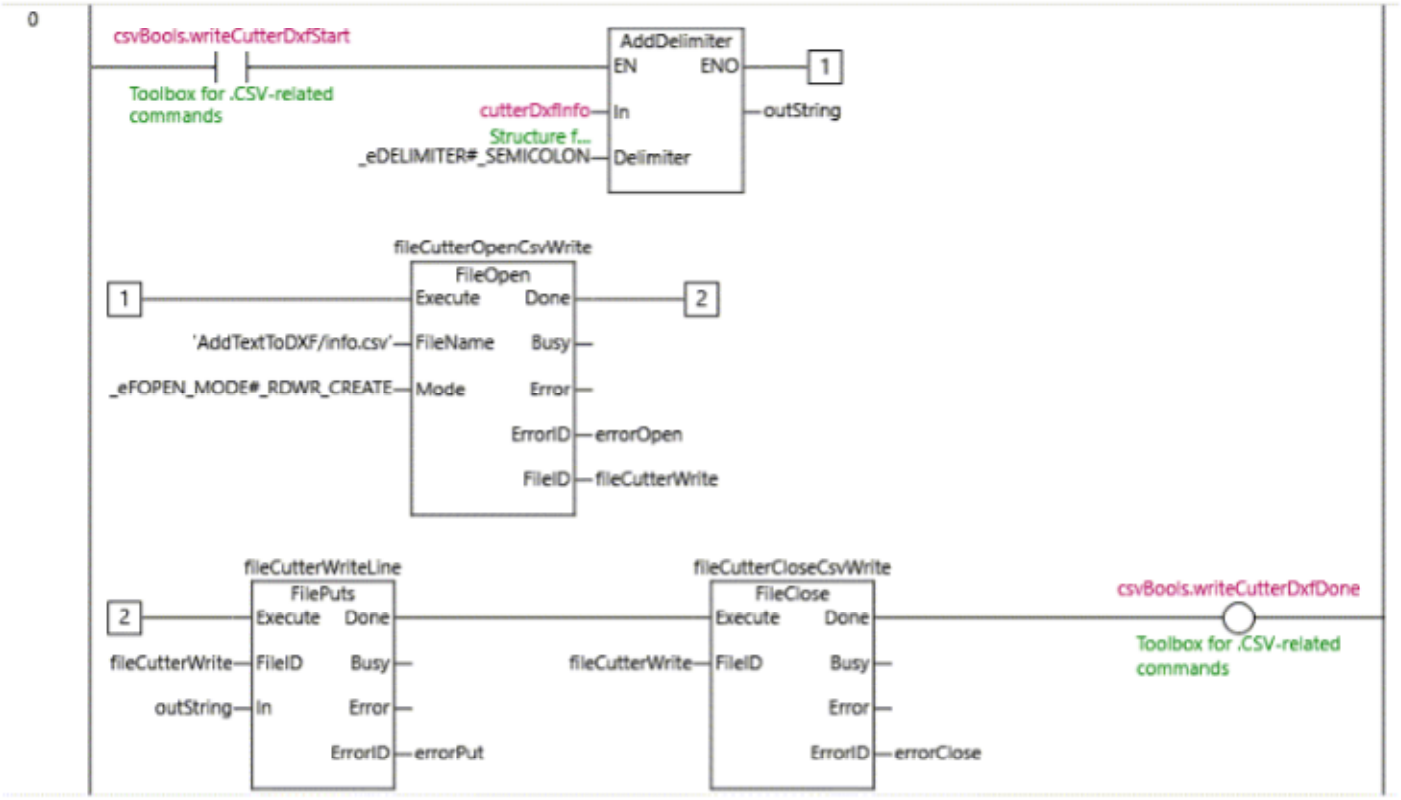
```
256         moviconSendStatus.printer06 := printerTempArray[6]; moviconSendStatus.printer16 := printerTempArray
[16];
257         moviconSendStatus.printer07 := printerTempArray[7]; moviconSendStatus.printer17 := printerTempArray
[17];
258         moviconSendStatus.printer08 := printerTempArray[8]; moviconSendStatus.printer18 := printerTempArray
[18];
259         moviconSendStatus.printer09 := printerTempArray[9]; moviconSendStatus.printer19 := printerTempArray
[19];
260         // ^Thanks Movicon for neither accepting array inputs nor structs inside other structs, would use the
earlier for loop for assigning if it did.
261     END_IF;
```

1-6-1-3.commsCSV

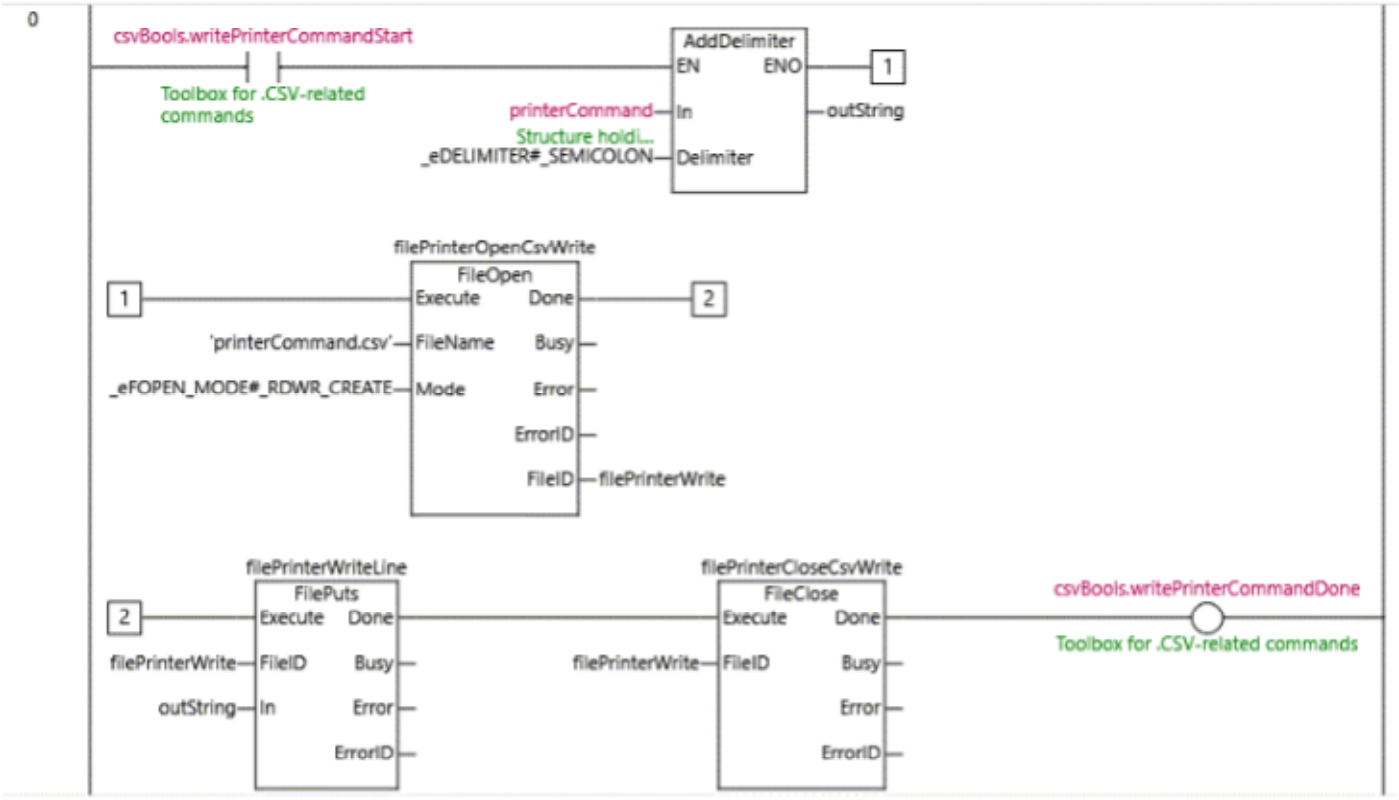
1-6-1-3-1.Variables

Name	Data Type	Initial Value	AT	Retain	Constant	Comment
VAR						
outString	STRING[256]			False	False	
fileCutterWrite	DWORD			False	False	
fileCutterRead	DWORD			False	False	
filePrinterWrite	DWORD			False	False	
filePrinterRead	DWORD			False	False	
offset	UINT			False	False	
state	UINT			False	False	
index	UINT			False	False	
errorOpen	WORD			False	False	
errorSeek	WORD			False	False	
errorPut	WORD			False	False	
errorRead	WORD			False	False	
errorClose	WORD			False	False	
fileCutterOpenCsvWrite	FileOpen			False	False	
fileCutterWriteLine	FilePuts			False	False	
fileCutterCloseCsvWrite	FileClose			False	False	
fileCutterOpenCsvRead	FileOpen			False	False	
fileCutterSetStartPoint	FileSeek			False	False	
fileCutterReadLine	FileGets			False	False	
fileCutterCloseCsvRead	FileClose			False	False	
filePrinterOpenCsvRead	FileOpen			False	False	
filePrinterSeek	FileSeek			False	False	
filePrinterReadLine	FileGets			False	False	
filePrinterCloseCsvRead	FileClose			False	False	
filePrinterOpenCsvWrite	FileOpen			False	False	
filePrinterWriteLine	FilePuts			False	False	
filePrinterCloseCsvWrite	FileClose			False	False	
VAR_EXTERNAL						
cutterDxfInfo	ST_CutterDxfExport				False	
cutterStatus	ST_CutterStatus				False	
csvBools	ST_CsvTools				False	
printerStatus	ARRAY[0..19] OF ST_3DPrinterStatus				False	
_Card1Ready	BOOL				True	
printerCommand	ST_3DPrinterCommand				False	

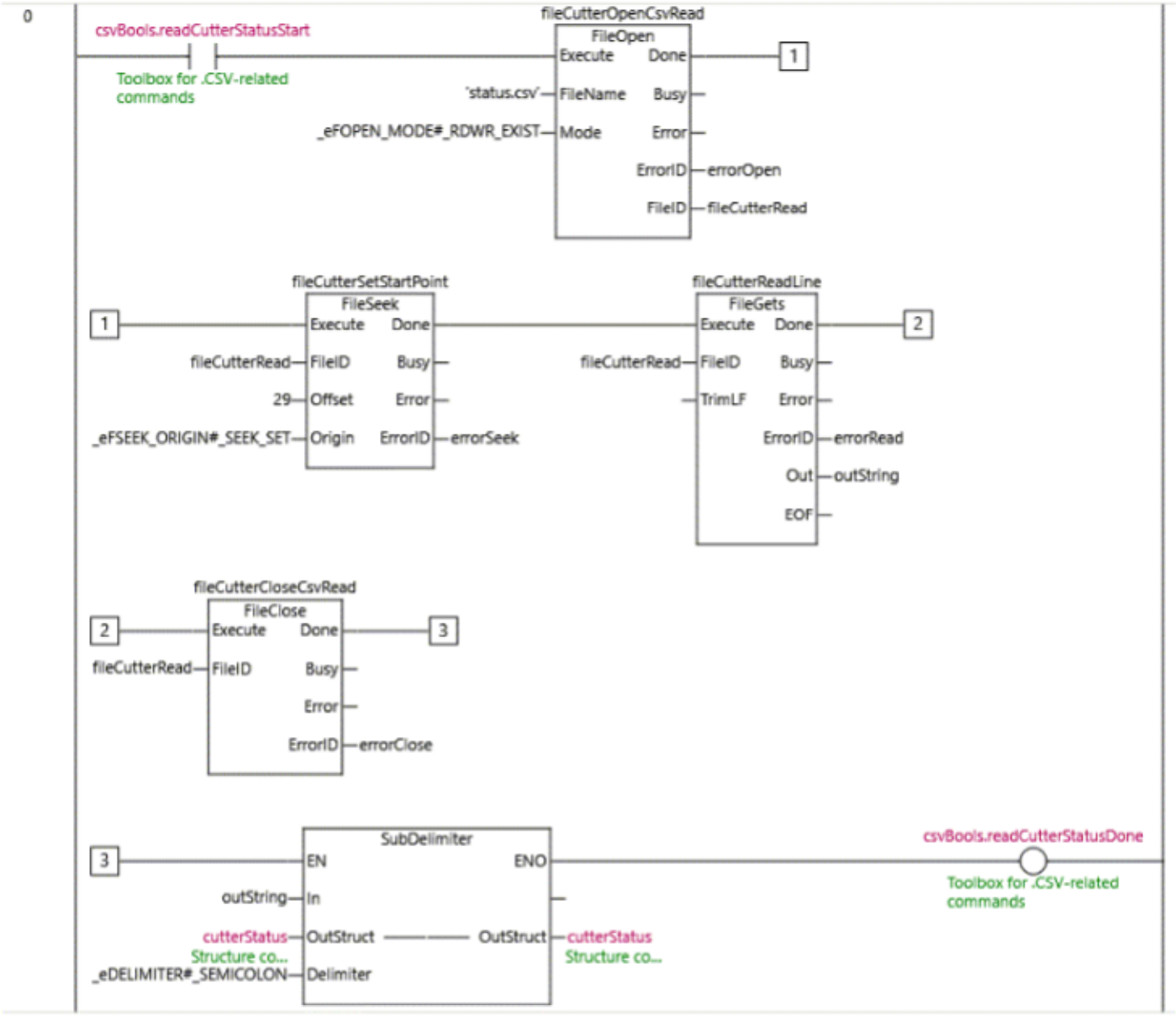
1-6-1-3-2.writeCutterDxf



1-6-1-3-3.writePrinterCommand



1-6-1-3-4.readCutterStatus



1-6-1-3-5.readPrinterStatus

0

csvBools.readPrinterStatusStart

1

Toolbox for .CSV-related commands

1

```
1  //Initialize variables
2  IF state = 0 THEN
3      offset := 116;
4      outString:='';
5      index := 0;
6
7      state := 11;
8  END_IF;
9
10 // Open csv file from SD card using the FileOpen function block
11 // State 10 = Category encompassing all FileOpen functions
12 // State 11 = Initiate function block
13 // State 12 = Execute function block
14 IF state = 11 THEN
15     IF _Card1Ready THEN
16         filePrinterOpenCsvRead(Execute:=FALSE);
17         state := 12;
18     END_IF;
19 END_IF;
20 IF state = 12 THEN
21     filePrinterOpenCsvRead(Execute:=TRUE,
   FileName:='printerStatus.csv',Mode:=_RDWR_EXIST, ErrorID=>errorOpen,
   FileID=>filePrinterRead);
22     IF NOT(filePrinterOpenCsvRead.Busy) THEN
23         IF filePrinterOpenCsvRead.Done THEN
24             // Open success
25
26             state := 21;
27         END_IF;
28         IF filePrinterOpenCsvRead.Error THEN
29             // Open failed
30             state := 110;
31         END_IF;
32     END_IF;
33 END_IF;
34
35 // Set the start point where reading will happen using the FileSeek function
   block
36 // State 20 = Category encompassing all FileSeek functions
37 // State 21 = Initiate function block
38 // State 22 = Execute function block
39 IF state = 21 THEN
40     filePrinterSeek(Execute:=FALSE);
41     state := 22;
42 END_IF;
43 IF state = 22 THEN
44     filePrinterSeek(Execute:=TRUE, FileID:=filePrinterOpenCsvRead.FileID,
   Offset:=offset, Origin:=_SEEK_SET);
45     IF NOT(filePrinterSeek.Busy) THEN
46         IF filePrinterSeek.Done THEN
47             // Seek success
48             state := 31;
49         END_IF;
50         IF filePrinterSeek.Error THEN
51             // Seek failed
52             state := 120;
```

```
53          END_IF;
54      END_IF;
55  END_IF;
56
57  // Read the rest of the line from the start point set in state 20
58  // State 30 = Category encompassing all FileGets functions
59  // State 31 = Initiate function block
60  // State 32 = Execute function block
61  IF state = 31 THEN
62      filePrinterReadLine(Execute:=FALSE);
63      state := 32;
64  END_IF;
65  IF state = 32 THEN
66      filePrinterReadLine(Execute:=TRUE, FileID:=filePrinterOpenCsvRead.FileID);//,
    Out=>outString);
67      IF NOT(filePrinterReadLine.Busy) THEN
68          IF filePrinterReadLine.Done THEN
69              // Read success
70              state := 40;
71          END_IF;
72          IF filePrinterReadLine.Error THEN
73              // Read failed
74              state := 130;
75          END_IF;
76      END_IF;
77  END_IF;
78
79  // 40 = Process data.
80  // Increase the offset to go to the end of the line, so that the next FileSeek will
    start from the next line.
81  // Separate the string at each ' ; ' and put each string segment into the
    printerStatus struct.
82  // Remove the $R$L linebreak from the end of the last string segment
83  // Increase index by 1.
84  // - If index is still within range of the printerStatus array, go back to state 20 and
    repeat until this is no longer the case.
85  // - If index becomes larger than total amount of elements in the printerStatus
    array, continue with the program.
86  IF state = 40 THEN
87      offset := offset + GetByteLen(In:=filePrinterReadLine.Out);
88      SubDelimiter(In:=filePrinterReadLine.Out, OutStruct:=printerStatus[index],
    Delimiter:=_eDELIMITER#_SEMICOLON);
89      index := index + 1;
90      IF index > (SizeOfAry(printerStatus)-1) THEN
91          state := 51;
92      ELSIF index <= (SizeOfAry(printerStatus)-1) THEN
93          printerStatus[index].posY := LEFT(In:=printerStatus[index].posY, L:=LEN
    (printerStatus[index].posY)-2);
94          state := 21;
95      END_IF;
96  END_IF;
97
98
99  // Now that reading is complete, close the file with the FileClose function block
100 // State 50 = Category encompassing all FileClose functions
101 // State 51 = Initiate function block
102 // State 52 = Execute function block
103 IF state = 51 THEN
104     filePrinterCloseCsvRead(Execute:=FALSE);
105     state := 52;
106 END_IF;
107 IF state = 52 THEN
108     filePrinterCloseCsvRead(Execute:=TRUE, FileID:=filePrinterOpenCsvRead.FileID,
```

```
    Done=>csvBools.readPrinterStatusDone);
109   IF NOT(filePrinterCloseCsvRead.Busy) THEN
110     IF filePrinterCloseCsvRead.Done THEN
111       // Closed successfully
112       state := 60;
113     END_IF;
114     IF filePrinterCloseCsvRead.Error THEN
115       // Close failed
116       state := 150;
117     END_IF;
118   END_IF;
119 END_IF;
```

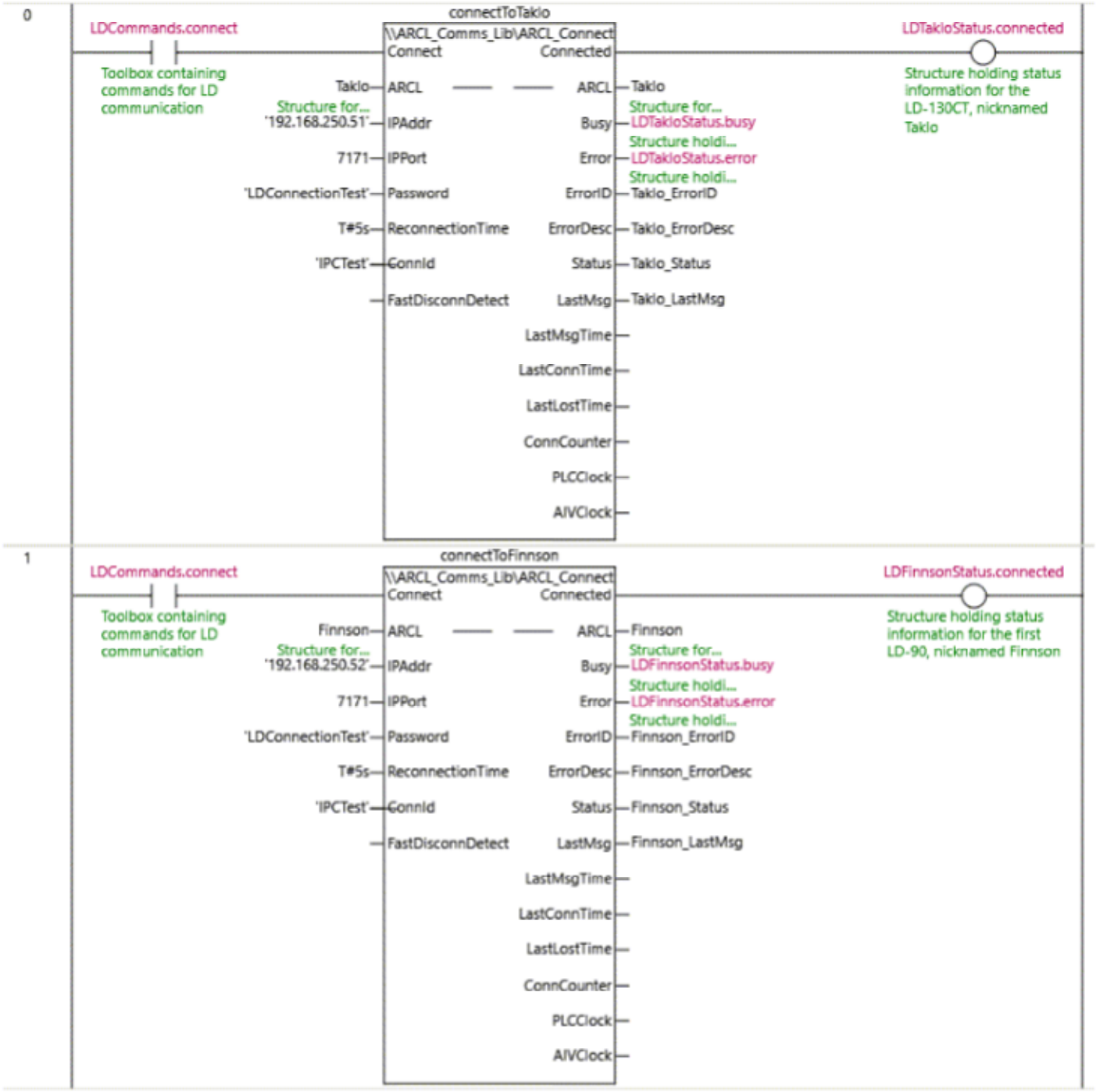
1-6-1-4.commsLD

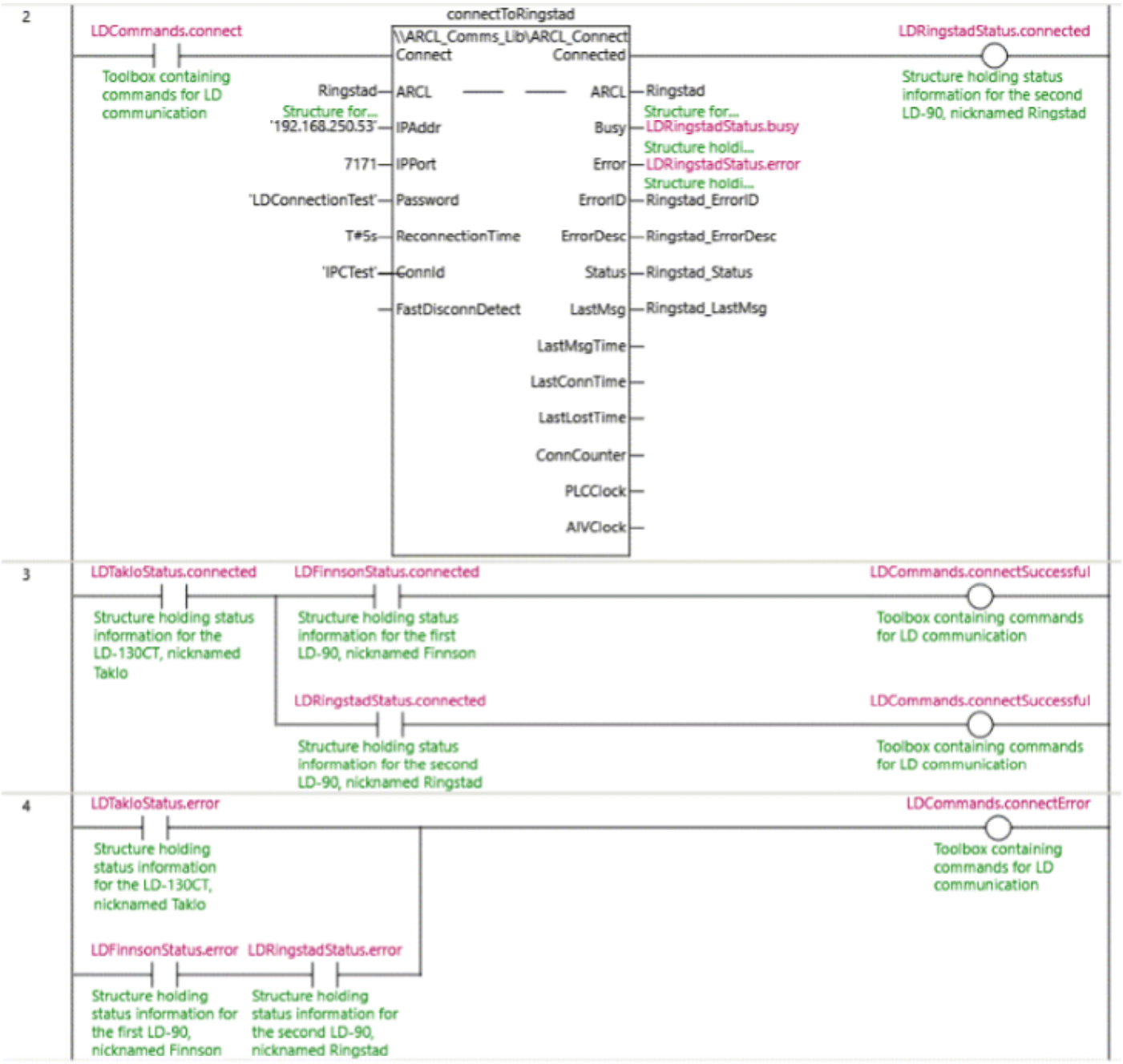
1-6-1-4-1.Variables

Name	Data Type	Initial Value	AT	Retain	Constant	Comment
VAR						
Taklo	ARCL_Comms_Lib\ARCL_ST_Connection			False	False	
Taklo_Status	STRING[30]			False	False	
Taklo_ErrorID	WORD			False	False	
Taklo_ErrorDesc	STRING[30]			False	False	
Taklo_LastMsg	STRING[200]			False	False	
connectToTaklo	ARCL_Comms_Lib\ARCL_Connect			False	False	
connectToFinnson	ARCL_Comms_Lib\ARCL_Connect			False	False	
Finnson	ARCL_Comms_Lib\ARCL_ST_Connection			False	False	
Finnson_ErrorID	WORD			False	False	
Finnson_ErrorDesc	STRING[30]			False	False	
Finnson_Status	STRING[30]			False	False	
Finnson_LastMsg	STRING[200]			False	False	
connectToRingstad	ARCL_Comms_Lib\ARCL_Connect			False	False	
Ringstad	ARCL_Comms_Lib\ARCL_ST_Connection			False	False	
Ringstad_ErrorID	WORD			False	False	
Ringstad_ErrorDesc	STRING[30]			False	False	
Ringstad_Status	STRING[30]			False	False	
jobName	STRING[25]			False	False	
Ringstad_LastMsg	STRING[200]			False	False	
printerPickup	ARCL_Comms_Lib\ARCL_QPickup			False	False	
printerConveyorArrived	BOOL			False	False	
printerDropoff	ARCL_Comms_Lib\ARCL_QDropoff			False	False	
kanbanPickup	ARCL_Comms_Lib\ARCL_QPickup			False	False	
kanbanDropoff	ARCL_Comms_Lib\ARCL_QDropoff			False	False	
cutterPickup	ARCL_Comms_Lib\ARCL_QPickup			False	False	
cutterDropoff	ARCL_Comms_Lib\ARCL_QDropoff			False	False	
deliveryPickup	ARCL_Comms_Lib\ARCL_QPickup			False	False	
deliveryDropoff	ARCL_Comms_Lib\ARCL_QDropoff			False	False	
printerPickupConv	ARCL_Comms_Lib\ARCL_QPickup			False	False	
printerDropoffConv	ARCL_Comms_Lib\ARCL_QDropoff			False	False	
VAR_EXTERNAL						
LDCommands	ST_LDTools				False	
TMCommands	ST_TMTools				False	

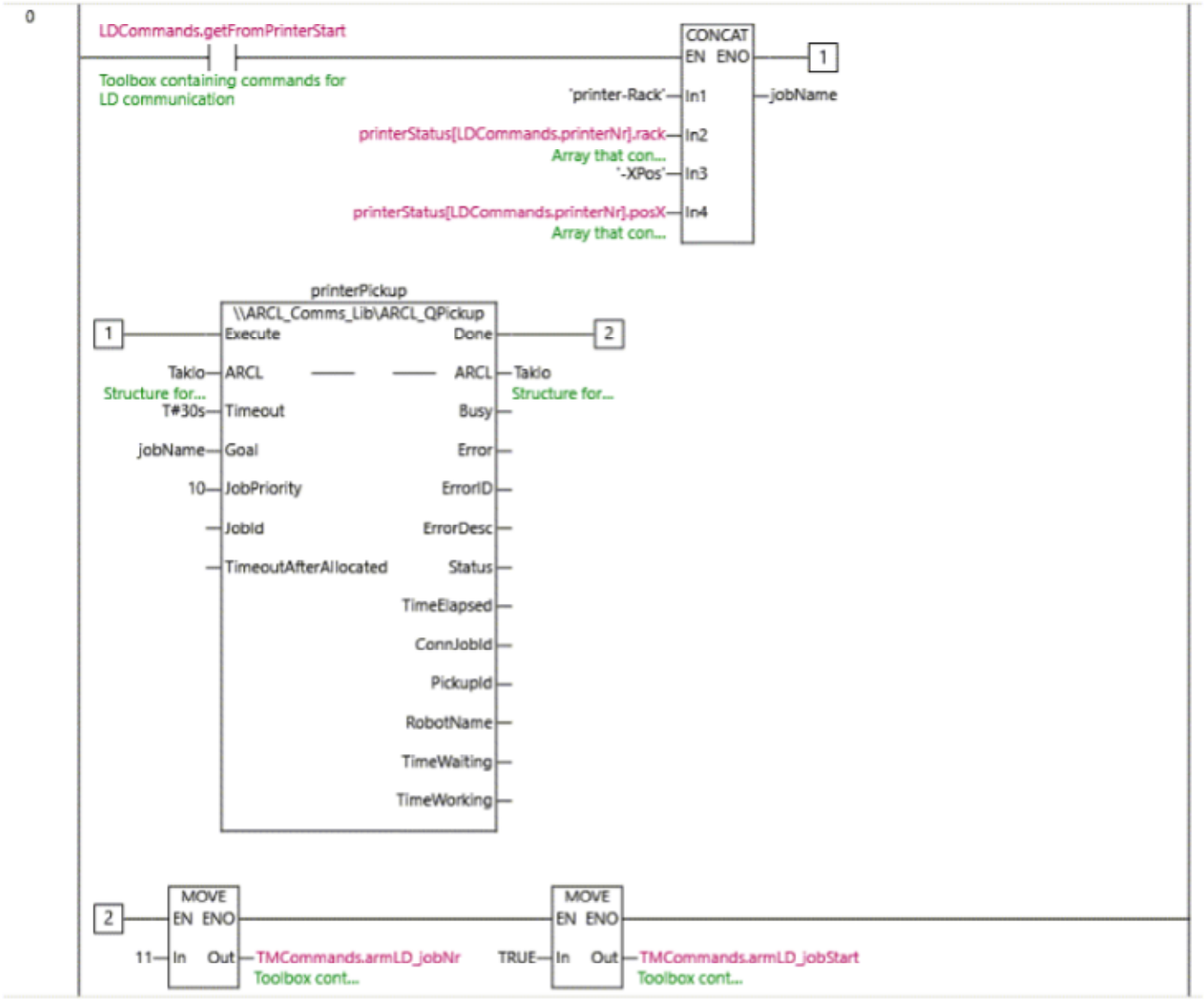
	printerStatus	ARRAY[0..19] OF ST_3DPrinterStatus				False	
	LDTakloStatus	ST_LDStatus				False	
	LDFinnsonStatus	ST_LDStatus				False	
	LDRingstadStatus	ST_LDStatus				False	

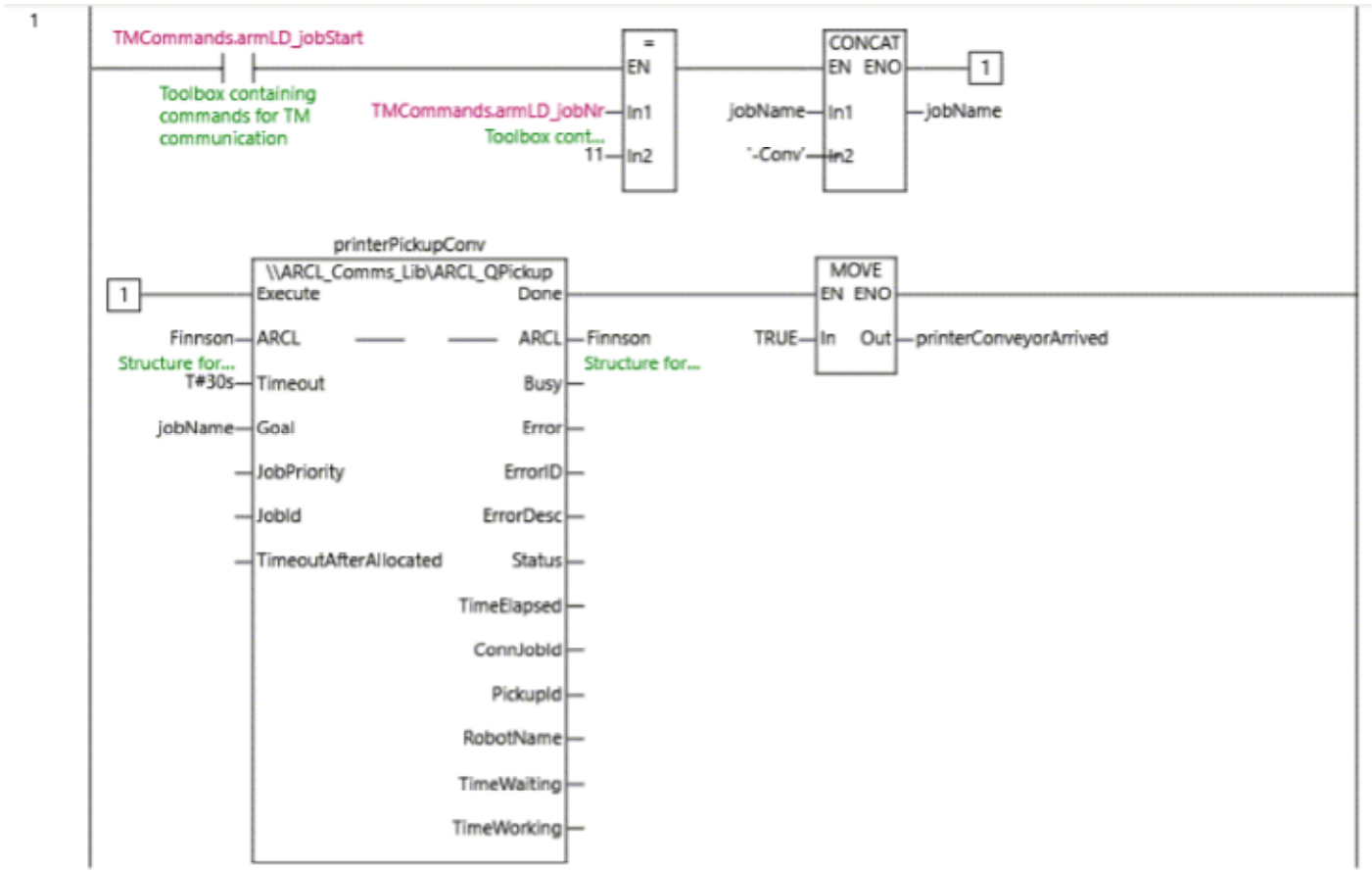
1-6-1-4-2.ConnectToLDs

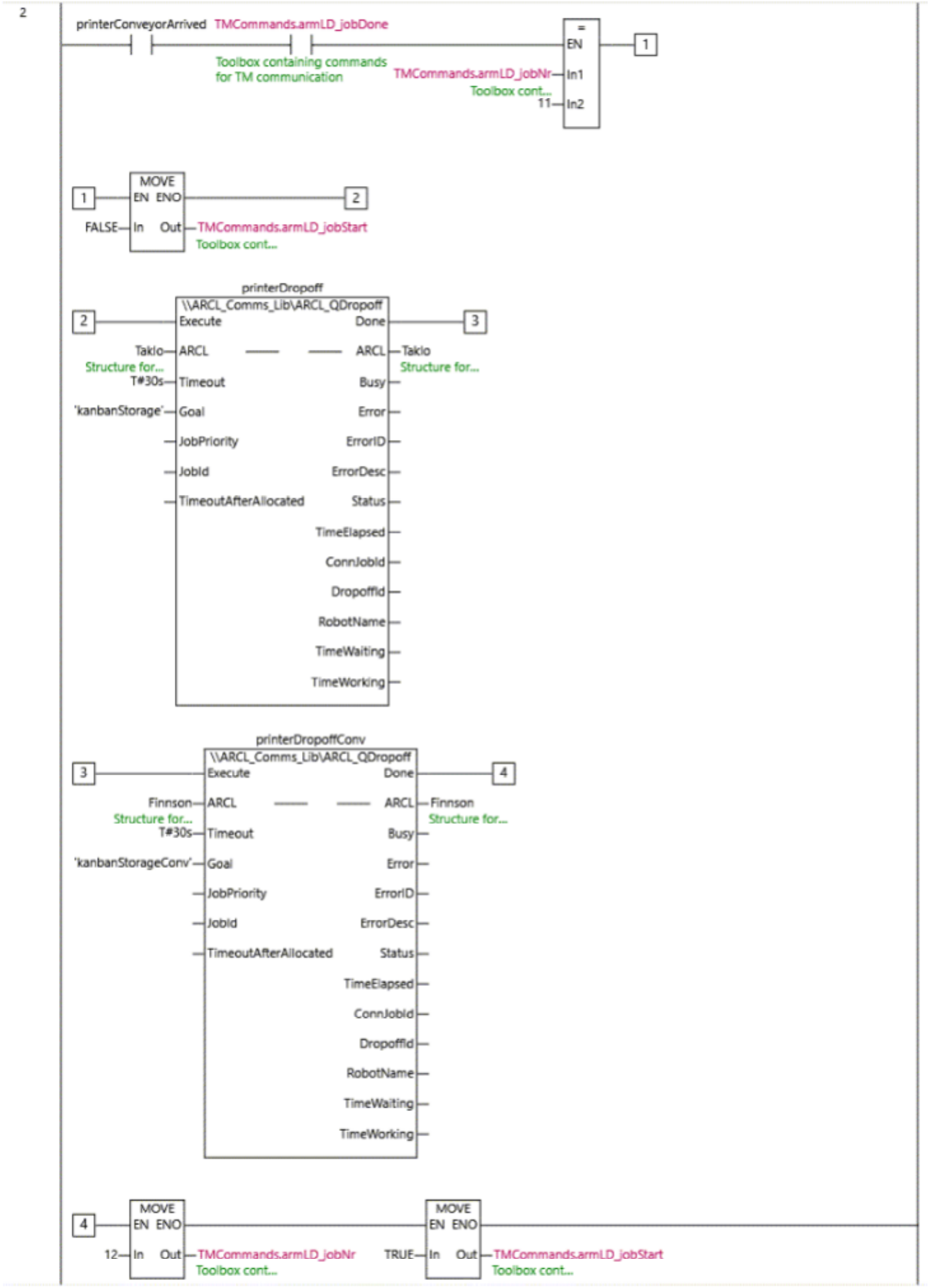


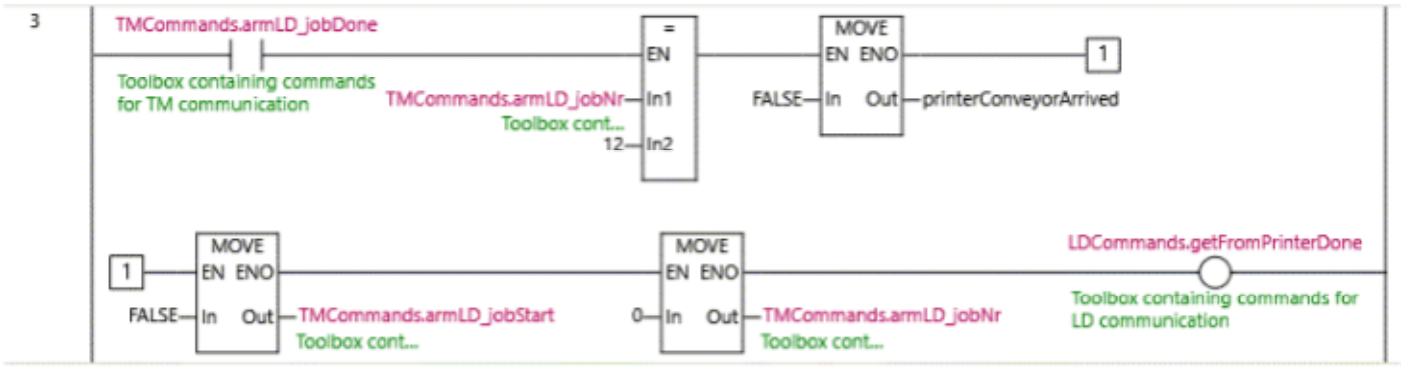

2
LDCommands.connect
Toolbox containing commands for LD communication
connectToRingstad
\\ARCL_Comms_Lib\ARCL_Connect
Connect
Connected
Ringstad
ARCL
ARCL
Ringstad
Structure for... '192.168.250.53'
IPAddr
Busy
Structure for... LDRingstadStatus.busy
7171
IPPort
Error
Structure holdi... LDRingstadStatus.error
'LDConnectionTest'
Password
ErrorID
Structure holdi... Ringstad_ErrorID
T#5s
ReconnectionTime
ErrorDesc
Ringstad_ErrorDesc
'IPCTest'
ConnId
Status
Ringstad_Status
FastDisconnDetect
LastMsg
Ringstad_LastMsg
LastMsgTime
LastConnTime
LastLostTime
ConnCounter
PLCClock
AIVClock
LDRingstadStatus.connected
Structure holding status information for the second LD-90, nicknamed Ringstad
3
LDTakloStatus.connected
Structure holding status information for the LD-130CT, nicknamed Taklo
LDFinnsonStatus.connected
Structure holding status information for the first LD-90, nicknamed Finnson
LDCommands.connectSuccessful
Toolbox containing commands for LD communication
LDRingstadStatus.connected
Structure holding status information for the second LD-90, nicknamed Ringstad
LDCommands.connectSuccessful
Toolbox containing commands for LD communication
4
LDTakloStatus.error
Structure holding status information for the LD-130CT, nicknamed Taklo
LDCommands.connectError
Toolbox containing commands for LD communication
LDFinnsonStatus.error
Structure holding status information for the first LD-90, nicknamed Finnson
LDRingstadStatus.error
Structure holding status information for the second LD-90, nicknamed Ringstad

1-6-1-4-3.getFromPrinter

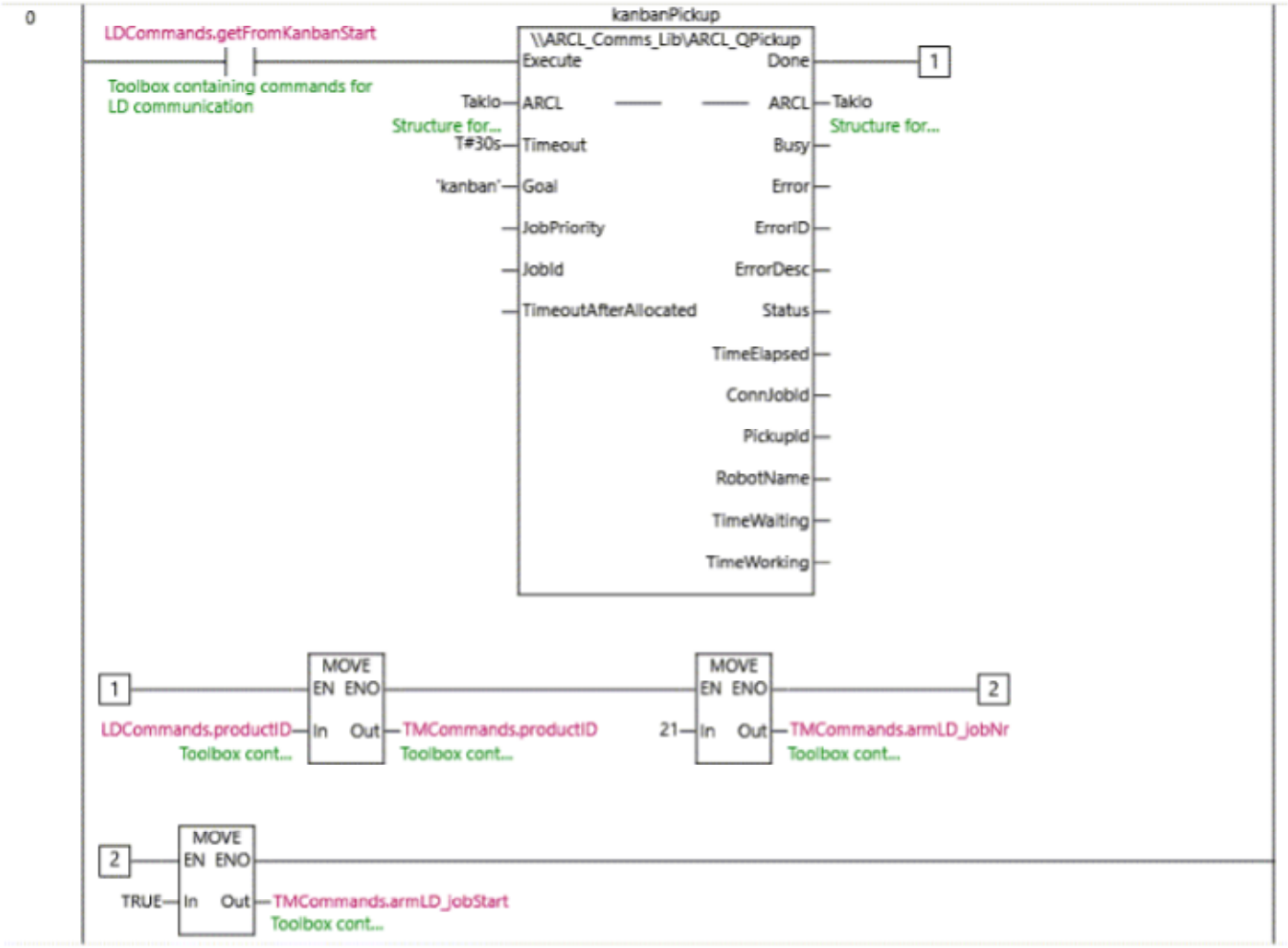


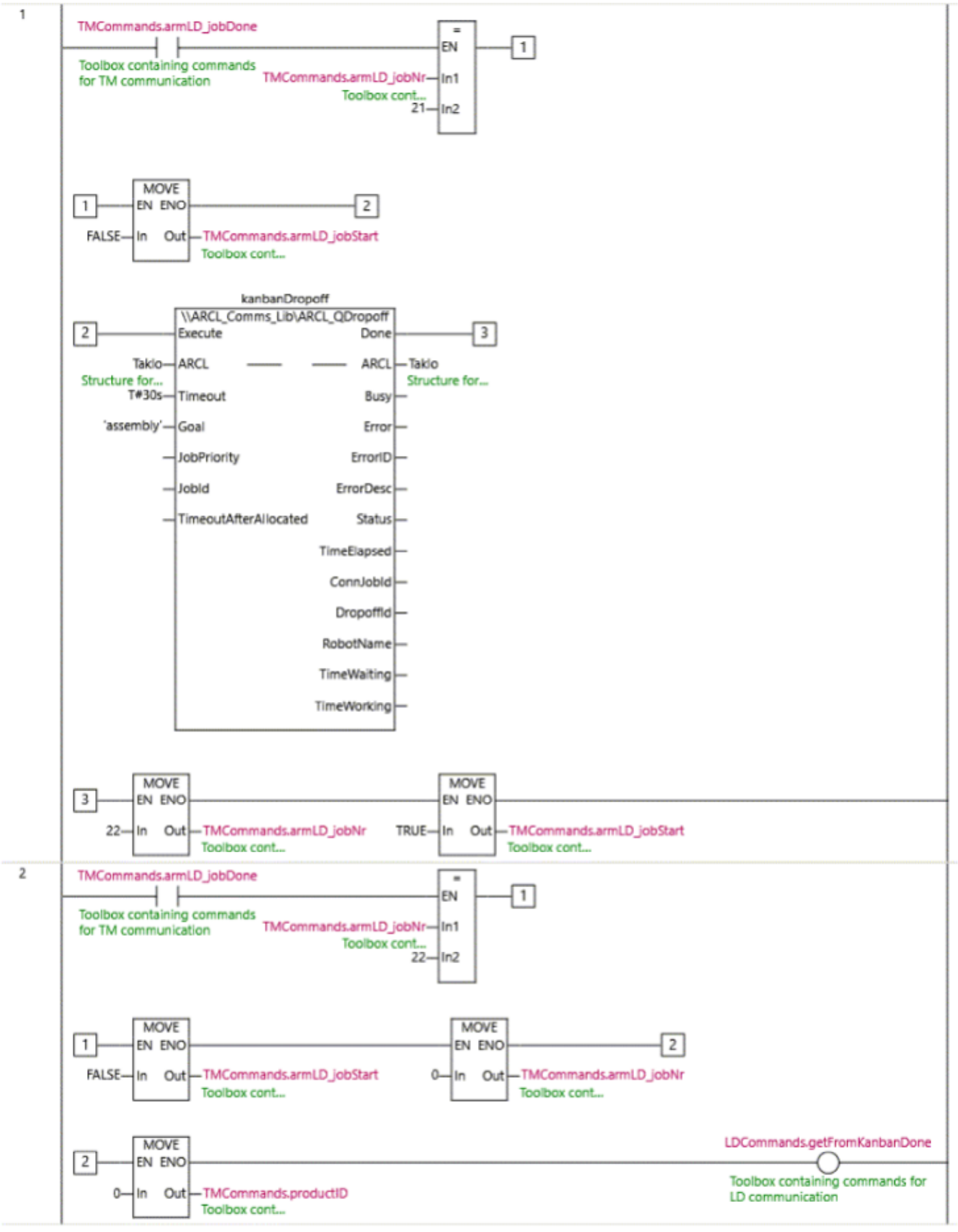

1
TMCommands.armLD_jobStart
Toolbox containing commands for TM communication
=
EN
TMCommands.armLD_jobNr
Toolbox cont...
11
In1
In2
CONCAT
EN ENO
jobName
In1
'-Conv'
In2
1
jobName
printerPickupConv
\\ARCL_Comms_Lib\ARCL_QPickup
1
Execute
Done
Finnson
Structure for...
ARCL
ARCL
Finnson
Structure for...
T#30s
Timeout
Busy
jobName
Goal
Error
JobPriority
ErrorID
JobId
ErrorDesc
TimeoutAfterAllocated
Status
TimeElapsed
ConnJobId
PickupId
RobotName
TimeWaiting
TimeWorking
MOVE
EN ENO
TRUE
In
Out
printerConveyorArrived


2
printerConveyorArrived
TMCommands.armLD_jobDone
Toolbox containing commands for TM communication
=
EN
1
TMCommands.armLD_jobNr
In1
Toolbox cont...
11
In2
MOVE
EN ENO
1
2
FALSE
In
Out
TMCommands.armLD_jobStart
Toolbox cont...
printerDropoff
\\ARCL_Comms_Lib\ARCL_QDropoff
Execute
Done
3
TakIo
ARCL
ARCL
TakIo
Structure for...
Structure for...
T#30s
Timeout
Busy
'kanbanStorage'
Goal
Error
JobPriority
ErrorID
JobId
ErrorDesc
TimeoutAfterAllocated
Status
TimeElapsed
ConnJobId
DropoffId
RobotName
TimeWaiting
TimeWorking
printerDropoffConv
\\ARCL_Comms_Lib\ARCL_QDropoff
Execute
Done
3
4
Finnson
ARCL
ARCL
Finnson
Structure for...
Structure for...
T#30s
Timeout
Busy
'kanbanStorageConv'
Goal
Error
JobPriority
ErrorID
JobId
ErrorDesc
TimeoutAfterAllocated
Status
TimeElapsed
ConnJobId
DropoffId
RobotName
TimeWaiting
TimeWorking
MOVE
EN ENO
4
12
In
Out
TMCommands.armLD_jobNr
Toolbox cont...
MOVE
EN ENO
TRUE
In
Out
TMCommands.armLD_jobStart
Toolbox cont...

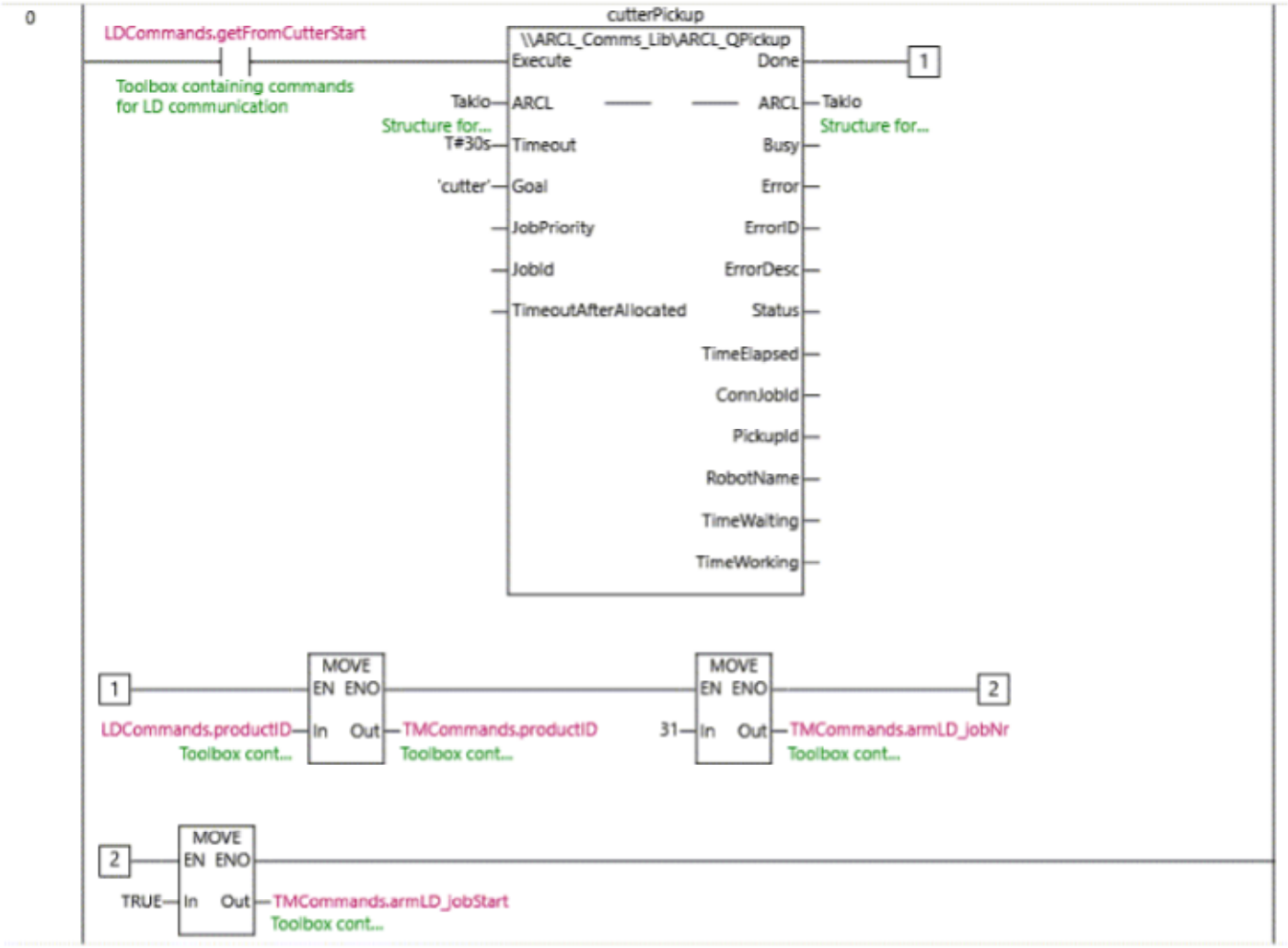

3
TMCommands.armLD_jobDone
Toolbox containing commands for TM communication
=
EN
TMCommands.armLD_jobNr
In1
Toolbox cont...
12
In2
MOVE
EN ENO
FALSE
In Out
printerConveyorArrived
1
1
MOVE
EN ENO
FALSE
In Out
TMCommands.armLD_jobStart
Toolbox cont...
MOVE
EN ENO
0
In Out
TMCommands.armLD_jobNr
Toolbox cont...
LDCommands.getFromPrinterDone
Toolbox containing commands for LD communication

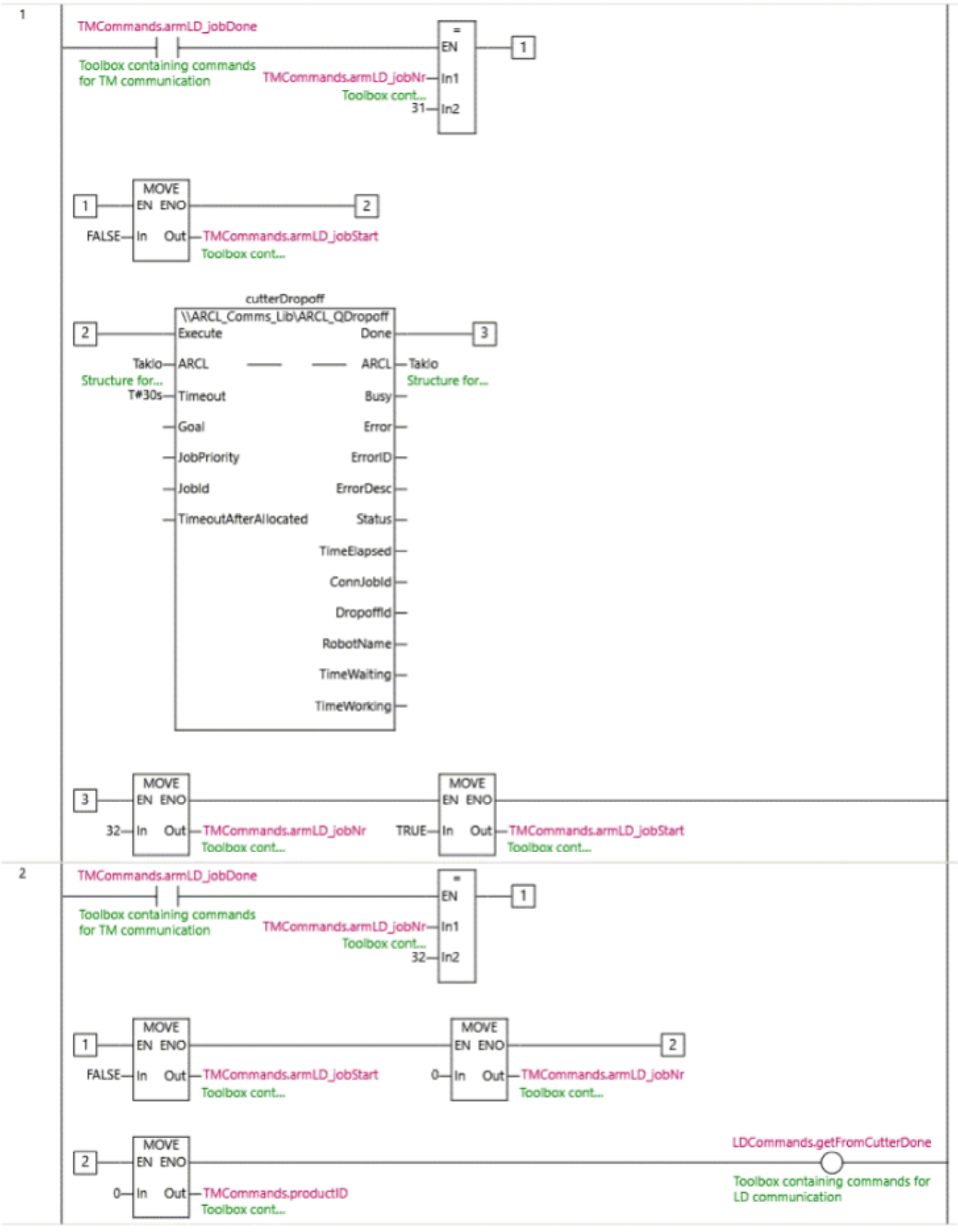
1-6-1-4-4.getFromKanban



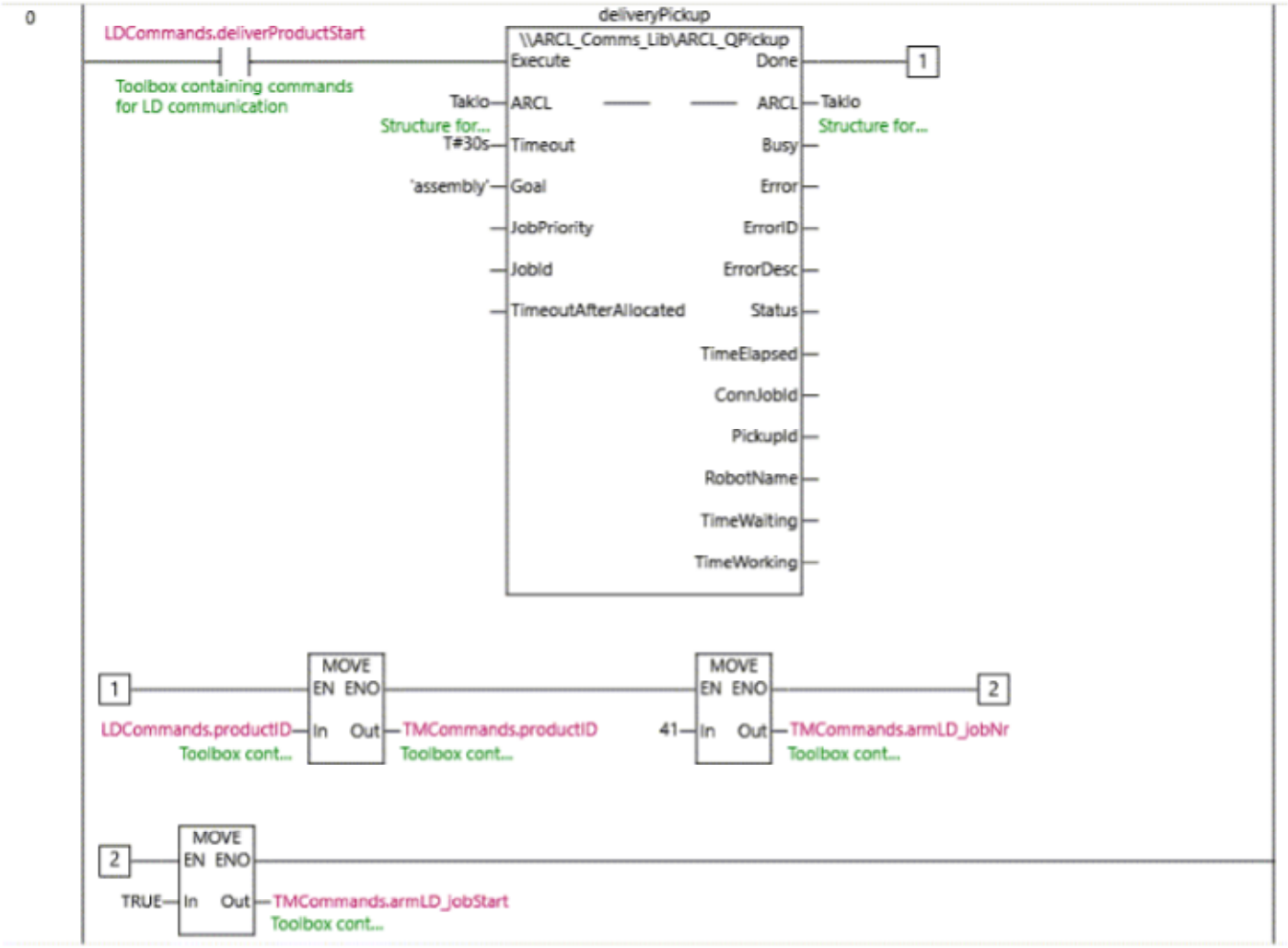

1
TMCommands.armLD_jobDone
Toolbox containing commands for TM communication
=
EN
1
TMCommands.armLD_jobNr
Toolbox cont...
In1
21
In2
MOVE
EN ENO
1
2
FALSE
In Out
TMCommands.armLD_jobStart
Toolbox cont...
kanbanDropoff
\\ARCL_Comms_Lib\ARCL_QDropoff
2
Execute
Done
3
TakIo
ARCL
ARCL
TakIo
Structure for...
Structure for...
T#30s
Timeout
Busy
'assembly'
Goal
Error
JobPriority
ErrorID
JobId
ErrorDesc
TimeoutAfterAllocated
Status
TimeElapsed
ConnJobId
DropoffId
RobotName
TimeWaiting
TimeWorking
MOVE
EN ENO
3
22
In Out
TMCommands.armLD_jobNr
Toolbox cont...
MOVE
EN ENO
TRUE
In Out
TMCommands.armLD_jobStart
Toolbox cont...
2
TMCommands.armLD_jobDone
Toolbox containing commands for TM communication
=
EN
1
TMCommands.armLD_jobNr
Toolbox cont...
In1
22
In2
MOVE
EN ENO
1
FALSE
In Out
TMCommands.armLD_jobStart
Toolbox cont...
MOVE
EN ENO
0
In Out
TMCommands.armLD_jobNr
Toolbox cont...
2
MOVE
EN ENO
2
0
In Out
TMCommands.productID
Toolbox cont...
LDCommands.getFromKanbanDone
Toolbox containing commands for LD communication

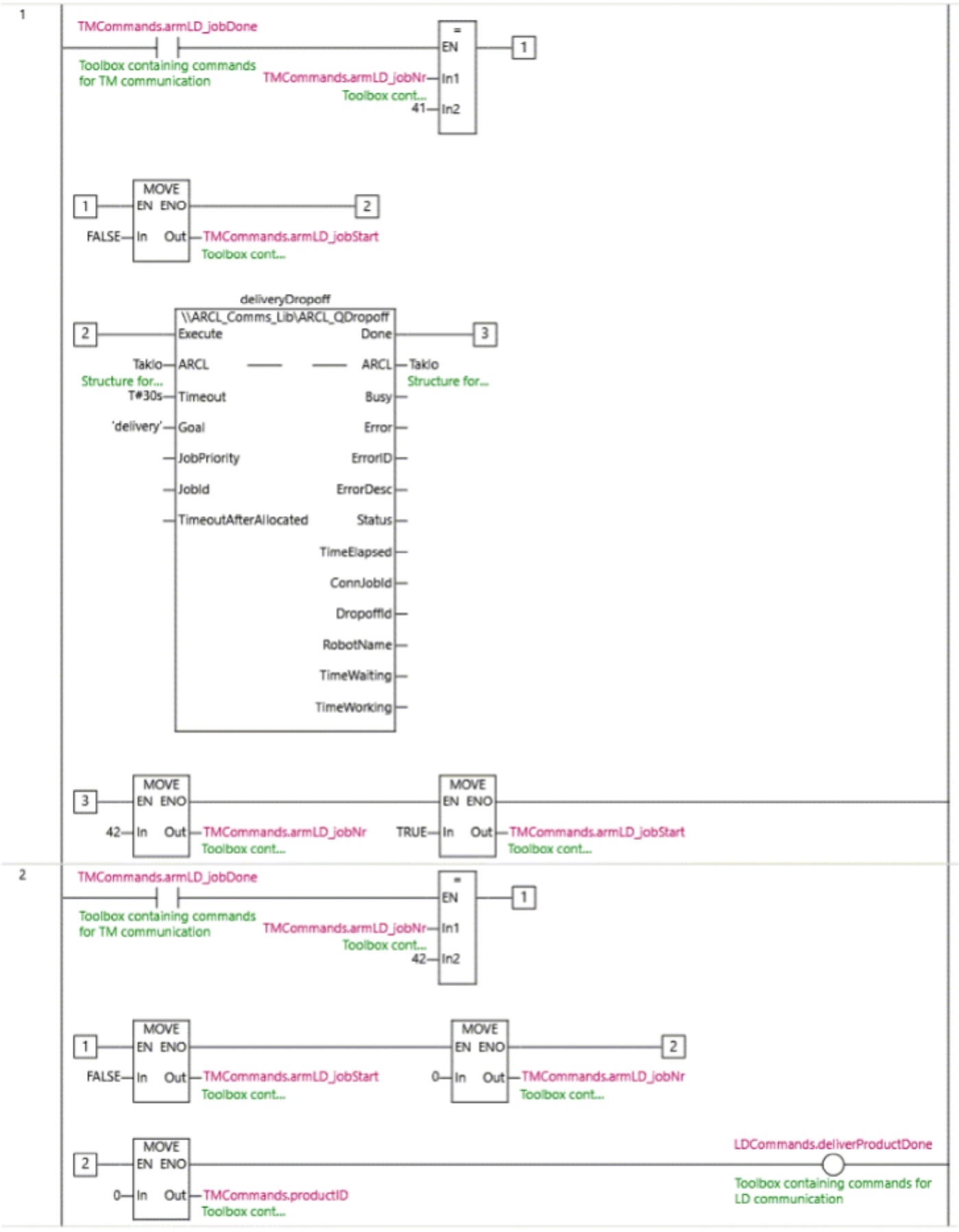
1-6-1-4-5.getFromCutter




1
TMCommands.armLD_jobDone
Toolbox containing commands for TM communication
=
EN
1
TMCommands.armLD_jobNr
Toolbox cont...
In1
31
In2
MOVE
EN ENO
1
2
FALSE
In Out
TMCommands.armLD_jobStart
Toolbox cont...
cutterDropoff
\\ARCL_Comms_Lib\ARCL_QDropoff
Execute
Done
3
2
TakIo
ARCL
ARCL
TakIo
Structure for...
Structure for...
T#30s
Timeout
Busy
Goal
Error
JobPriority
ErrorID
JobId
ErrorDesc
TimeoutAfterAllocated
Status
TimeElapsed
ConnJobId
DropoffId
RobotName
TimeWaiting
TimeWorking
MOVE
EN ENO
3
32
In Out
TMCommands.armLD_jobNr
Toolbox cont...
MOVE
EN ENO
TRUE
In Out
TMCommands.armLD_jobStart
Toolbox cont...
2
TMCommands.armLD_jobDone
Toolbox containing commands for TM communication
=
EN
1
TMCommands.armLD_jobNr
Toolbox cont...
In1
32
In2
MOVE
EN ENO
1
FALSE
In Out
TMCommands.armLD_jobStart
Toolbox cont...
MOVE
EN ENO
2
0
In Out
TMCommands.armLD_jobNr
Toolbox cont...
MOVE
EN ENO
2
0
In Out
TMCommands.productID
Toolbox cont...
LDCommands.getFromCutterDone
Toolbox containing commands for LD communication

1-6-1-4-6.deliverProduct




1
TMCommands.armLD_jobDone
Toolbox containing commands for TM communication
=
EN
1
TMCommands.armLD_jobNr
Toolbox cont...
In1
41
In2
MOVE
EN ENO
1
2
FALSE
In Out
TMCommands.armLD_jobStart
Toolbox cont...
deliveryDropoff
\\ARCL_Comms_Lib\ARCL_QDropoff
2
Execute
Done
3
Takio
ARCL
ARCL
Takio
Structure for...
Structure for...
T#30s
Timeout
Busy
'delivery'
Goal
Error
JobPriority
ErrorID
JobId
ErrorDesc
TimeoutAfterAllocated
Status
TimeElapsed
ConnJobId
DropoffId
RobotName
TimeWaiting
TimeWorking
3
42
TMCommands.armLD_jobNr
Toolbox cont...
TRUE
TMCommands.armLD_jobStart
Toolbox cont...
2
TMCommands.armLD_jobDone
Toolbox containing commands for TM communication
1
TMCommands.armLD_jobNr
Toolbox cont...
42
FALSE
TMCommands.armLD_jobStart
Toolbox cont...
0
TMCommands.armLD_jobNr
Toolbox cont...
2
0
TMCommands.productID
Toolbox cont...
LDCommands.deliverProductDone
Toolbox containing commands for LD communication

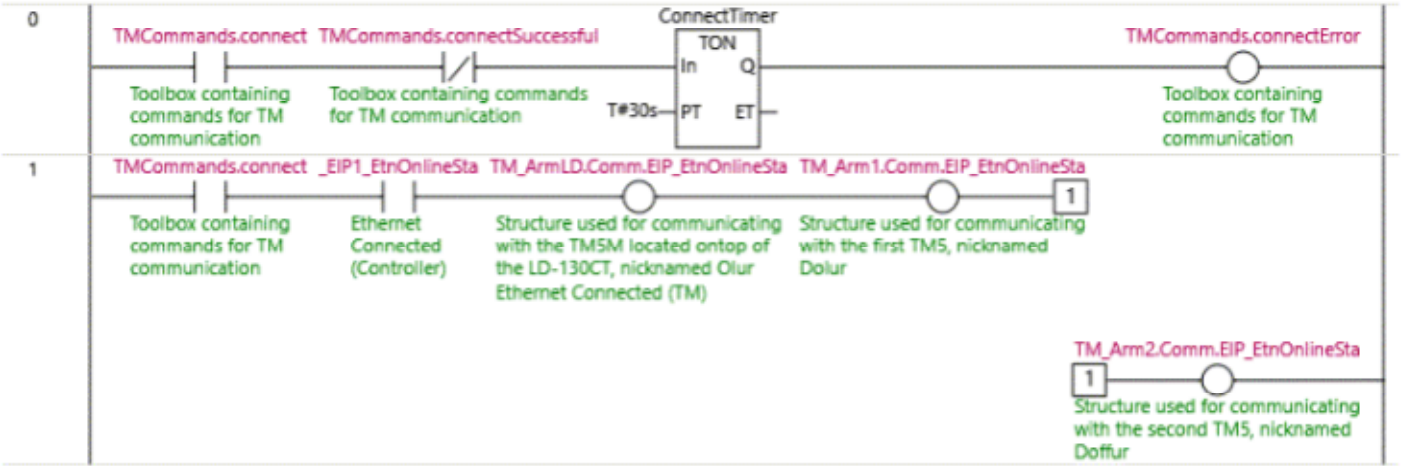
1-6-1-5.commsTM

1-6-1-5-1.Variables

Name	Data Type	Initial Value	AT	Retain	Constant	Comment
VAR						
FBModbus_ArmLD	OEN\Eth\TCP_Connect			False	False	
FBModbus_ArmLD_State	_eCONNECTION_STATE			False	False	
FBModbus_ArmLD_SocketError	BOOL			False	False	
FBModbus_ArmLD_ErrorID	WORD			False	False	
FBModbus_ArmLD_ErrorIDtxt	STRING[100]			False	False	
FB_TM_ArmLD	OEN\Robot\TM\TM_ModbusTCP			False	False	
TM_testconnect	BOOL			False	False	
FB_TMControl	OEN\Robot\TM\TM_Control			False	False	
ArmLD_PlayPausc	BOOL			False	False	
ArmLD_Stop	BOOL			False	False	
ArmLD_ToggleAuto	BOOL			False	False	
ArmLD_Mode	OEN\nRobot\nTM\eMode			False	False	
ArmLD_Connected	BOOL			False	False	
ArmLD_AttemptConnect	BOOL			False	False	
jobStarted	BOOL			False	False	
jobNr	INT			False	False	
jobStart	BOOL			False	False	
jobDone	BOOL			False	False	
ConnectTimer	TON			False	False	
Arm1_AttemptConnect	BOOL			False	False	
FBModbus_Arm1	OEN\Eth\TCP_Connect			False	False	
FB_TM_Arm1	OEN\Robot\TM\TM_ModbusTCP			False	False	
FB_TMControl_ArmLD	OEN\Robot\TM\TM_Control			False	False	
FB_TMControl_Arm1	OEN\Robot\TM\TM_Control			False	False	
Arm1_PlayPausc	BOOL			False	False	
Arm1_Stop	BOOL			False	False	
Arm1_ToggleAuto	BOOL			False	False	
Arm1_Mode	OEN\nRobot\nTM\eMode			False	False	
Arm1_Connected	BOOL			False	False	
Arm2_AttemptConnect	BOOL			False	False	
FBModbus_Arm2	OEN\Eth\TCP_Connect			False	False	
FBModbus_Arm2_State	_eCONNECTION_STATE			False	False	
FBModbus_Arm2_SocketError	BOOL			False	False	

	FBModbus_Arm2_ErrorID	WORD			False	False	
	FBModbus_Arm2_ErrorIDtxt	STRING[100]			False	False	
	FB_TM_Arm2	OEN\Robot\TM\TM_ModbusTCP			False	False	
	Arm2_PlayPause	BOOL			False	False	
	Arm2_Stop	BOOL			False	False	
	Arm2_ToggleAuto	BOOL			False	False	
	Arm2_Mode	OEN\nRobot\nTM\eMode			False	False	
	Arm2_Connected	BOOL			False	False	
	armLD_jobStarted	BOOL			False	False	
	arm1_jobStarted	BOOL			False	False	
	arm2_jobStarted	BOOL			False	False	
VAR_EXTERNAL							
	_EIP1_EtnOnlineSta	BOOL				True	Ethernet Connected (Controller)
	TM_ArmLD	OEN\nRobot\nTM\sRobot				False	
	TMCommands	ST_TMTools				False	
	TM_Arm1	OEN\nRobot\nTM\sRobot				False	
	TM_Arm2	OEN\nRobot\nTM\sRobot				False	

1-6-1-5-2.ModbusTCP_Setup



2

TM_ArmLD.Comm.EIP_EtnOnlineSta

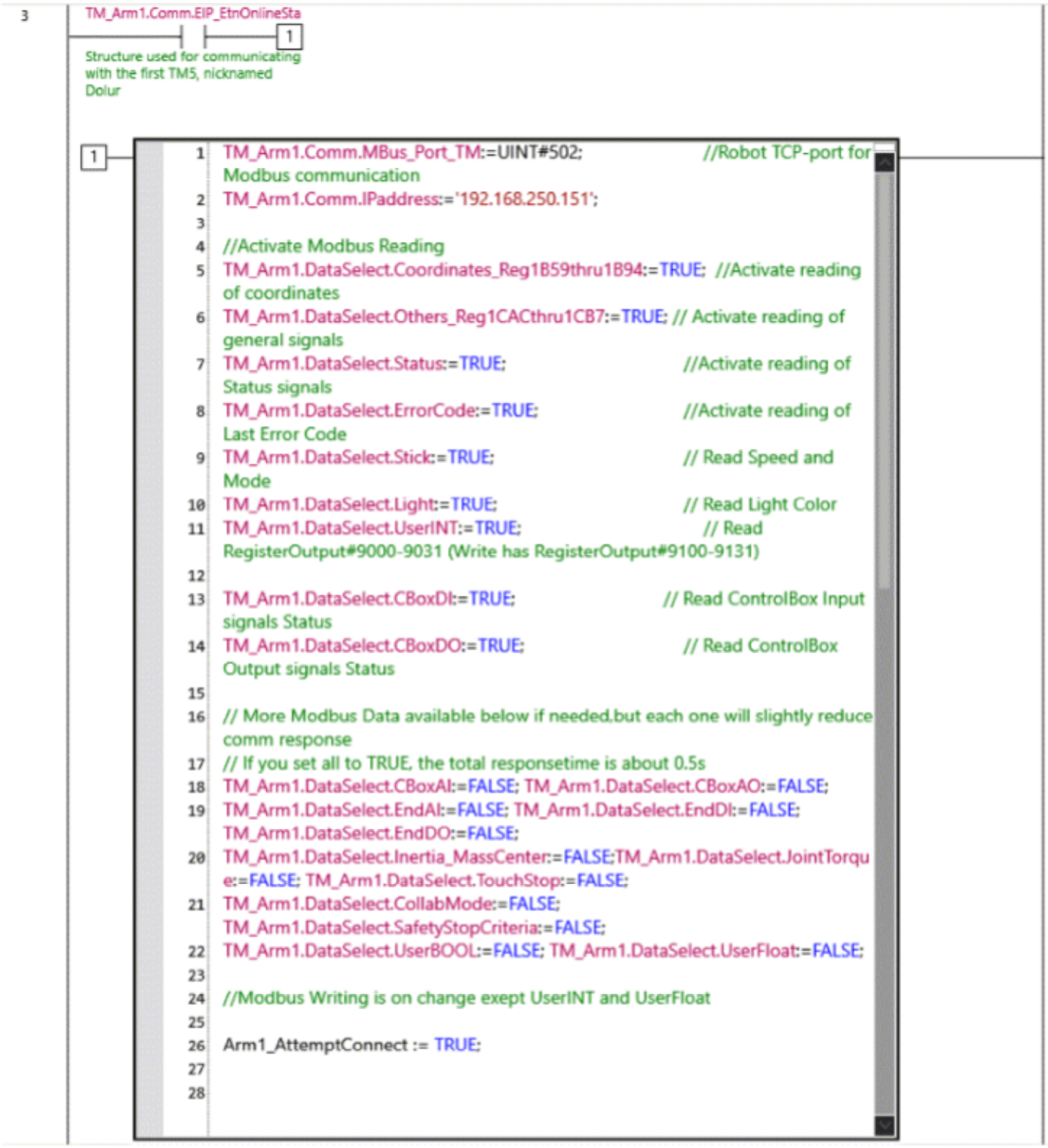
1

Structure used for communicating with the TM5M located ontop of the LD-130CT, nicknamed Olur Ethernet Connected (TM)

1

```
1  TM_ArmLD.Comm.MBus_Port_TM:=UINT#502;                    //Robot TCP-port for
   Modbus communication
2  TM_ArmLD.Comm.IPaddress:='192.168.250.150';
3
4  //Activate Modbus Reading
5  TM_ArmLD.DataSelect.Coordinates_Reg1B59thru1B94:=TRUE;  //Activate reading
   of coordinates
6  TM_ArmLD.DataSelect.Others_Reg1CACthru1CB7:=TRUE; // Activate reading of
   general signals
7  TM_ArmLD.DataSelect.Status:=TRUE;                        //Activate reading of
   Status signals
8  TM_ArmLD.DataSelect.ErrorCode:=TRUE;                     //Activate reading of
   Last Error Code
9  TM_ArmLD.DataSelect.Stick:=TRUE;                         // Read Speed and
   Mode
10 TM_ArmLD.DataSelect.Light:=TRUE;                         // Read Light Color
11 TM_ArmLD.DataSelect.UserINT:=TRUE;                          // Read
   RegisterOutput#9000-9031 (Write has RegisterOutput#9100-9131)
12
13 TM_ArmLD.DataSelect.CBoxDI:=TRUE;                        // Read ControlBox
   Input signals Status
14 TM_ArmLD.DataSelect.CBoxDO:=TRUE;                        // Read ControlBox
   Output signals Status
15
16 // More Modbus Data available below if needed,but each one will slightly reduce
   comm response
17 // If you set all to TRUE, the total responsetime is about 0.5s
18 TM_ArmLD.DataSelect.CBoxAI:=FALSE; TM_ArmLD.DataSelect.CBoxAO:=FALSE;
19 TM_ArmLD.DataSelect.EndAI:=FALSE; TM_ArmLD.DataSelect.EndDI:=FALSE;
   TM_ArmLD.DataSelect.EndDO:=FALSE;
20 TM_ArmLD.DataSelect.Inertia_MassCenter:=FALSE;TM_ArmLD.DataSelect.JointTor
   que:=FALSE; TM_ArmLD.DataSelect.TouchStop:=FALSE;
21 TM_ArmLD.DataSelect.CollabMode:=FALSE;
   TM_ArmLD.DataSelect.SafetyStopCriteria:=FALSE;
22 TM_ArmLD.DataSelect.UserBOOL:=FALSE;
   TM_ArmLD.DataSelect.UserFloat:=FALSE;
23
24 //Modbus Writing is on change exept UserINT and UserFloat
25
26 ArmLD_AttemptConnect := TRUE;
27
28
```

3


TM_Arm1.Comm.EIP_EtnOnlineSta
1
Structure used for communicating with the first TM5, nicknamed Dolur
1
1 TM_Arm1.Comm.MBus_Port_TM:=UINT#502; //Robot TCP-port for Modbus communication
2 TM_Arm1.Comm.IPaddress:='192.168.250.151';
3
4 //Activate Modbus Reading
5 TM_Arm1.DataSelect.Coordinates_Reg1B59thru1B94:=TRUE; //Activate reading of coordinates
6 TM_Arm1.DataSelect.Others_Reg1CACthru1CB7:=TRUE; // Activate reading of general signals
7 TM_Arm1.DataSelect.Status:=TRUE; //Activate reading of Status signals
8 TM_Arm1.DataSelect.ErrorCode:=TRUE; //Activate reading of Last Error Code
9 TM_Arm1.DataSelect.Stick:=TRUE; // Read Speed and Mode
10 TM_Arm1.DataSelect.Light:=TRUE; // Read Light Color
11 TM_Arm1.DataSelect.UserINT:=TRUE; // Read RegisterOutput#9000-9031 (Write has RegisterOutput#9100-9131)
12
13 TM_Arm1.DataSelect.CBoxDI:=TRUE; // Read ControlBox Input signals Status
14 TM_Arm1.DataSelect.CBoxDO:=TRUE; // Read ControlBox Output signals Status
15
16 // More Modbus Data available below if needed,but each one will slightly reduce comm response
17 // If you set all to TRUE, the total responsetime is about 0.5s
18 TM_Arm1.DataSelect.CBoxAI:=FALSE; TM_Arm1.DataSelect.CBoxAO:=FALSE;
19 TM_Arm1.DataSelect.EndAI:=FALSE; TM_Arm1.DataSelect.EndDI:=FALSE; TM_Arm1.DataSelect.EndDO:=FALSE;
20 TM_Arm1.DataSelect.Inertia_MassCenter:=FALSE;TM_Arm1.DataSelect.JointTorque:=FALSE; TM_Arm1.DataSelect.TouchStop:=FALSE;
21 TM_Arm1.DataSelect.CollabMode:=FALSE; TM_Arm1.DataSelect.SafetyStopCriteria:=FALSE;
22 TM_Arm1.DataSelect.UserBOOL:=FALSE; TM_Arm1.DataSelect.UserFloat:=FALSE;
23
24 //Modbus Writing is on change exept UserINT and UserFloat
25
26 Arm1_AttemptConnect := TRUE;
27
28

4

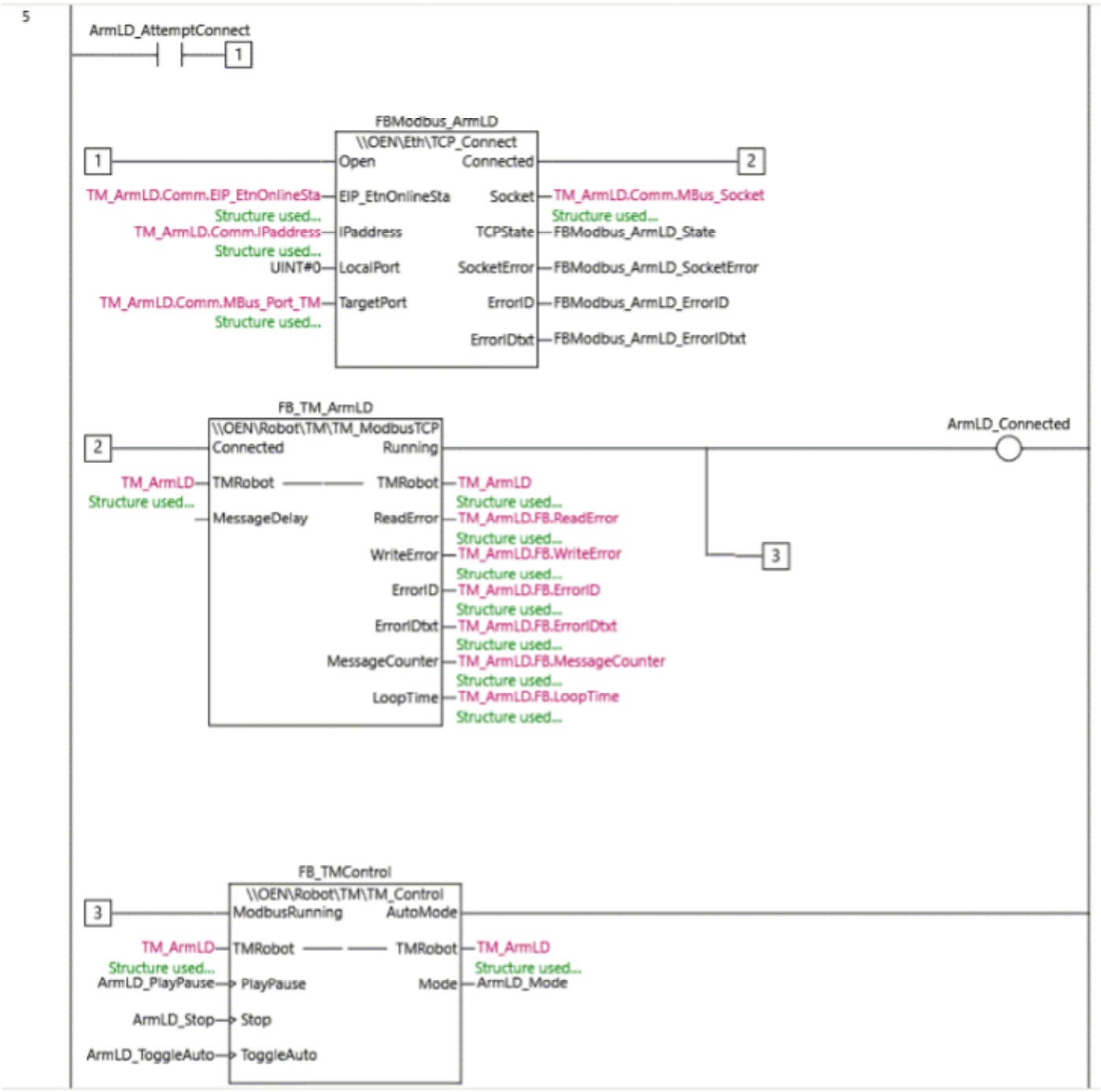
TM_Arm2.Comm.EIP_EtnOnlineSta

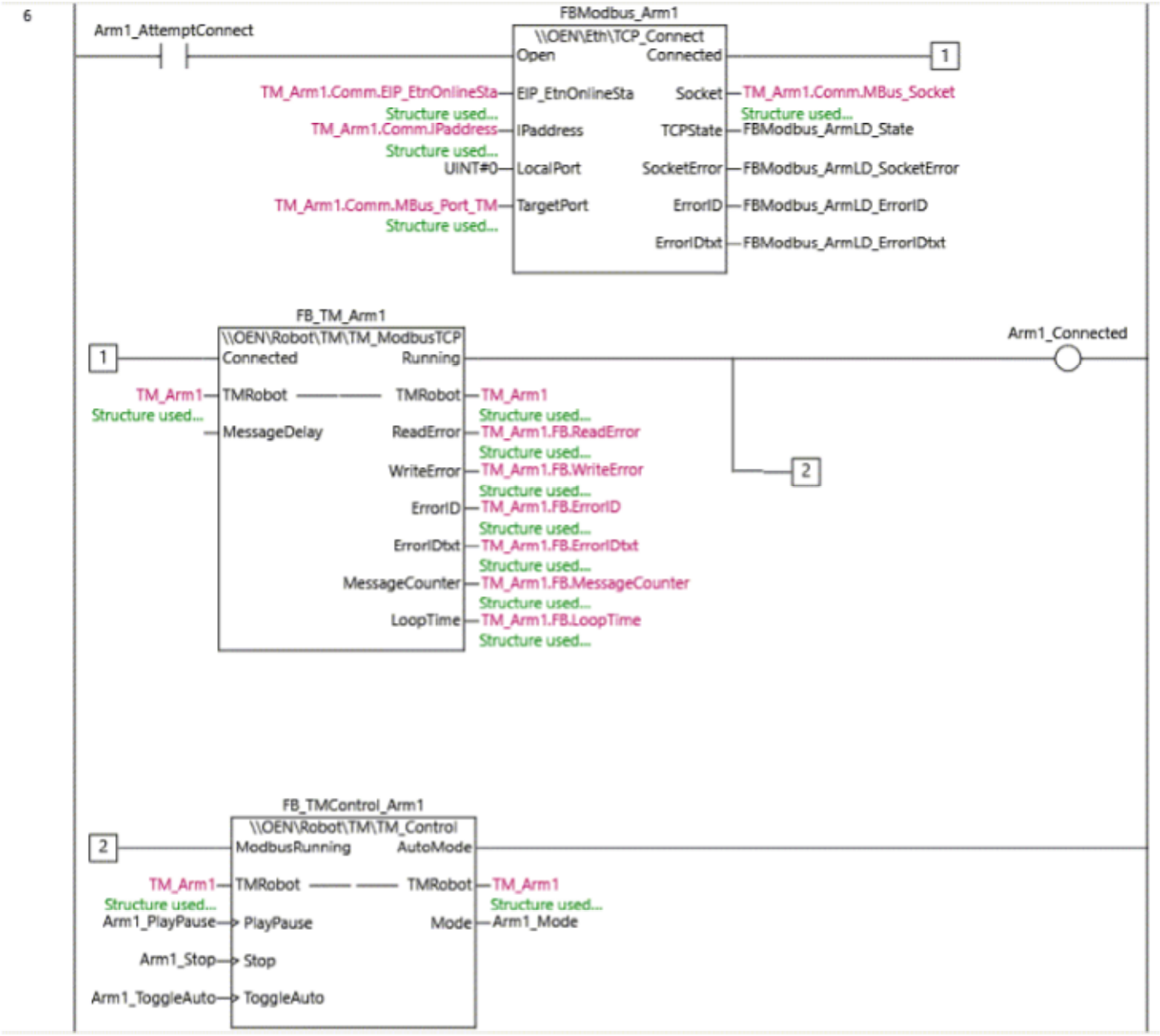
1

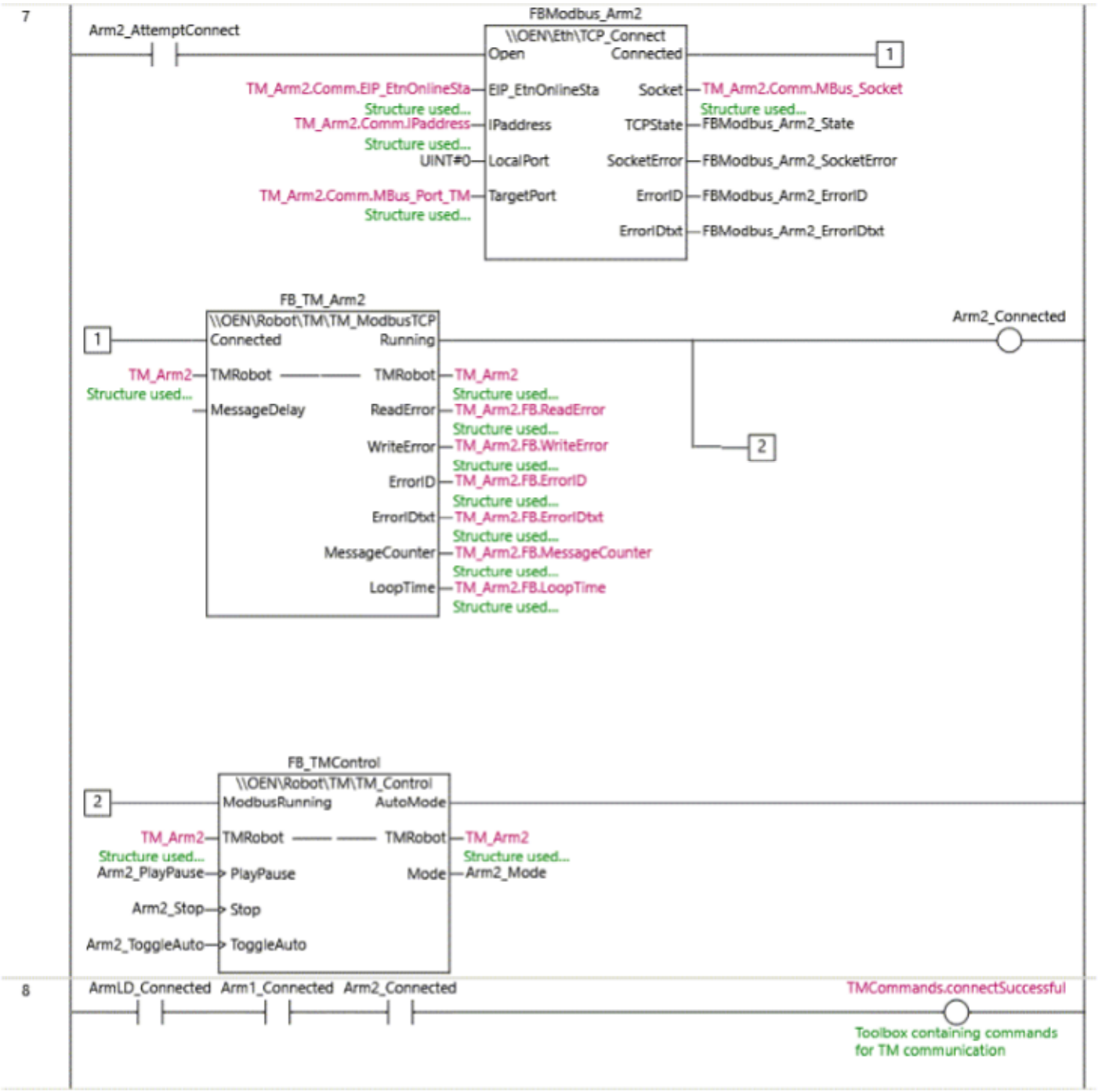
Structure used for communicating with the second TM5, nicknamed Doffur

1

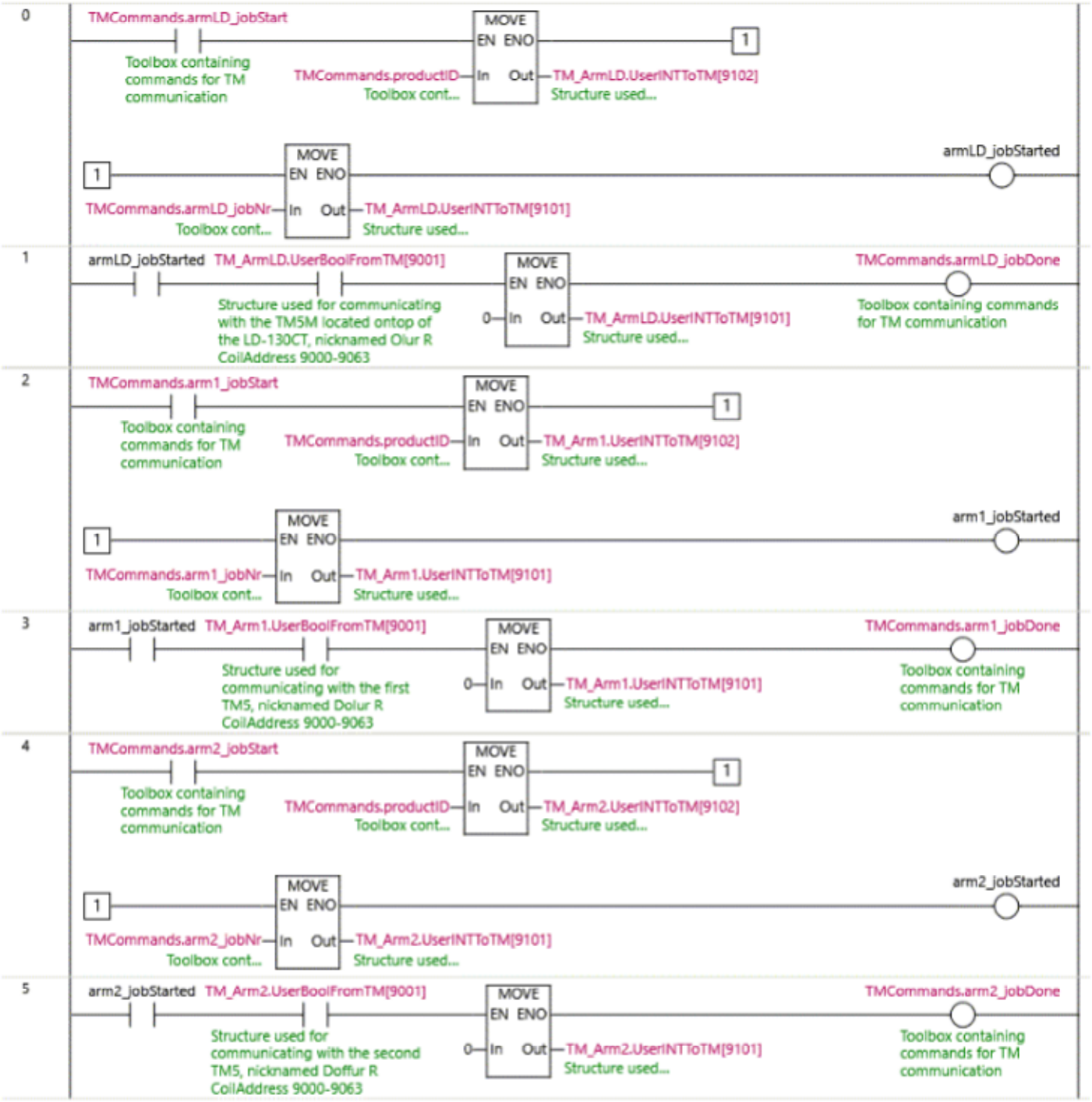
```
1  TM_Arm2.Comm.MBus_Port_TM:=UINT#502;                    //Robot TCP-port for
   Modbus communication
2  TM_Arm2.Comm.IPaddress:='192.168.250.152';
3
4  //Activate Modbus Reading
5  TM_Arm2.DataSelect.Coordinates_Reg1B59thru1B94:=TRUE;  //Activate reading
   of coordinates
6  TM_Arm2.DataSelect.Others_Reg1CACthru1CB7:=TRUE; // Activate reading of
   general signals
7  TM_Arm2.DataSelect.Status:=TRUE;                        //Activate reading of
   Status signals
8  TM_Arm2.DataSelect.ErrorCode:=TRUE;                     //Activate reading of
   Last Error Code
9  TM_Arm2.DataSelect.Stick:=TRUE;                         // Read Speed and
   Mode
10 TM_Arm2.DataSelect.Light:=TRUE;                         // Read Light Color
11 TM_Arm2.DataSelect.UserINT:=TRUE;                        // Read
   RegisterOutput#9000-9031 (Write has RegisterOutput#9100-9131)
12
13 TM_Arm2.DataSelect.CBoxDI:=TRUE;                       // Read ControlBox Input
   signals Status
14 TM_Arm2.DataSelect.CBoxDO:=TRUE;                        // Read ControlBox
   Output signals Status
15
16 // More Modbus Data available below if needed,but each one will slightly reduce
   comm response
17 // If you set all to TRUE, the total responsetime is about 0.5s
18 TM_Arm2.DataSelect.CBoxAI:=FALSE; TM_Arm2.DataSelect.CBoxAO:=FALSE;
19 TM_Arm2.DataSelect.EndAI:=FALSE; TM_Arm2.DataSelect.EndDI:=FALSE;
   TM_Arm2.DataSelect.EndDO:=FALSE;
20 TM_Arm2.DataSelect.Inertia_MassCenter:=FALSE;TM_Arm2.DataSelect.JointTorqu
   e:=FALSE; TM_Arm2.DataSelect.TouchStop:=FALSE;
21 TM_Arm2.DataSelect.CollabMode:=FALSE;
   TM_Arm2.DataSelect.SafetyStopCriteria:=FALSE;
22 TM_Arm2.DataSelect.UserBOOL:=FALSE; TM_Arm2.DataSelect.UserFloat:=FALSE;
23
24 //Modbus Writing is on change exept UserINT and UserFloat
25
26 Arm2_AttemptConnect := TRUE;
27
28
```


5
ArmLD_AttemptConnect
1
FBModbus_ArmLD
\\OEN\Eth\TCP_Connect
Open
Connected
2
TM_ArmLD.Comm.EIP_EtnOnlineSta
Structure used...
EIP_EtnOnlineSta
Socket
TM_ArmLD.Comm.MBus_Socket
Structure used...
TM_ArmLD.Comm.IPaddress
Structure used...
IPaddress
TCPState
FBModbus_ArmLD_State
UINT#0
LocalPort
SocketError
FBModbus_ArmLD_SocketError
TM_ArmLD.Comm.MBus_Port_TM
Structure used...
TargetPort
ErrorID
FBModbus_ArmLD_ErrorID
ErrorIDtxt
FBModbus_ArmLD_ErrorIDtxt
FB_TM_ArmLD
\\OEN\Robot\TM\TM_ModbusTCP
2
Connected
Running
ArmLD_Connected
TM_ArmLD
Structure used...
TMRobot
TMRobot
TM_ArmLD
Structure used...
MessageDelay
ReadError
TM_ArmLD.FB.ReadError
Structure used...
WriteError
TM_ArmLD.FB.WriteError
Structure used...
ErrorID
TM_ArmLD.FB.ErrorID
Structure used...
ErrorIDtxt
TM_ArmLD.FB.ErrorIDtxt
Structure used...
MessageCounter
TM_ArmLD.FB.MessageCounter
Structure used...
LoopTime
TM_ArmLD.FB.LoopTime
Structure used...
3
FB_TMControl
\\OEN\Robot\TM\TM_Control
3
ModbusRunning
AutoMode
TM_ArmLD
Structure used...
TMRobot
TMRobot
TM_ArmLD
Structure used...
ArmLD_PlayPause
PlayPause
Mode
ArmLD_Mode
ArmLD_Stop
Stop
ArmLD_ToggleAuto
ToggleAuto


6
Arm1_AttemptConnect
FBModbus_Arm1
\\OEN\Eth\TCP_Connect
Open
Connected
1
TM_Arm1.Comm.EIP_EtnOnlineSta
Structure used...
EIP_EtnOnlineSta
Socket
TM_Arm1.Comm.MBus_Socket
Structure used...
TM_Arm1.Comm.IPaddress
Structure used...
IPaddress
TCPState
FBModbus_ArmLD_State
UINT#0
LocalPort
SocketError
FBModbus_ArmLD_SocketError
TM_Arm1.Comm.MBus_Port_TM
Structure used...
TargetPort
ErrorID
FBModbus_ArmLD_ErrorID
ErrorIDtxt
FBModbus_ArmLD_ErrorIDtxt
FB_TM_Arm1
\\OEN\Robot\TM\TM_ModbusTCP
1
Connected
Running
Arm1_Connected
TM_Arm1
Structure used...
TMRobot
TMRobot
TM_Arm1
Structure used...
MessageDelay
ReadError
TM_Arm1.FB.ReadError
Structure used...
WriteError
TM_Arm1.FB.WriteError
Structure used...
ErrorID
TM_Arm1.FB.ErrorID
Structure used...
ErrorIDtxt
TM_Arm1.FB.ErrorIDtxt
Structure used...
MessageCounter
TM_Arm1.FB.MessageCounter
Structure used...
LoopTime
TM_Arm1.FB.LoopTime
Structure used...
2
FB_TMControl_Arm1
\\OEN\Robot\TM\TM_Control
2
ModbusRunning
AutoMode
TM_Arm1
Structure used...
TMRobot
TMRobot
TM_Arm1
Structure used...
Arm1_PlayPause
PlayPause
Mode
Arm1_Mode
Arm1_Stop
Stop
Arm1_ToggleAuto
ToggleAuto


7
Arm2_AttemptConnect
FBModbus_Arm2
\\OEN\Eth\TCP_Connect
Open
Connected
1
TM_Arm2.Comm.EIP_EtnOnlineSta
Structure used...
EIP_EtnOnlineSta
Socket
TM_Arm2.Comm.MBus_Socket
Structure used...
TM_Arm2.Comm.IPaddress
Structure used...
IPaddress
TCPState
FBModbus_Arm2_State
UINT#0
LocalPort
SocketError
FBModbus_Arm2_SocketError
TM_Arm2.Comm.MBus_Port_TM
Structure used...
TargetPort
ErrorID
FBModbus_Arm2_ErrorID
ErrorIDtxt
FBModbus_Arm2_ErrorIDtxt
FB_TM_Arm2
\\OEN\Robot\TM\TM_ModbusTCP
1
Connected
Running
Arm2_Connected
TM_Arm2
Structure used...
TMRobot
TMRobot
TM_Arm2
Structure used...
MessageDelay
ReadError
TM_Arm2.FB.ReadError
Structure used...
WriteError
TM_Arm2.FB.WriteError
Structure used...
ErrorID
TM_Arm2.FB.ErrorID
Structure used...
ErrorIDtxt
TM_Arm2.FB.ErrorIDtxt
Structure used...
MessageCounter
TM_Arm2.FB.MessageCounter
Structure used...
LoopTime
TM_Arm2.FB.LoopTime
Structure used...
2
FB_TMControl
\\OEN\Robot\TM\TM_Control
2
ModbusRunning
AutoMode
TM_Arm2
Structure used...
TMRobot
TMRobot
TM_Arm2
Structure used...
Arm2_PlayPause
PlayPause
Mode
Arm2_Mode
Arm2_Stop
Stop
Arm2_ToggleAuto
ToggleAuto
8
ArmLD_Connected
Arm1_Connected
Arm2_Connected
TMCommands.connectSuccessful
Toolbox containing commands
for TM communication

1-6-1-5-3.doTask



1-6-1-6.UDP_Socket

1-6-1-6-1.Variables

Name	Data Type	Initial Value	AT	Retain	Constant	Comment
VAR						
start	BOOL			False	False	
iSktCreate	SktUDPCreate			False	False	
state	DINT			False	False	
iSktSend	SktUDPSend			False	False	
iSktReceive	SktUDPRcv			False	False	
iSktClose	SktClose			False	False	
iSktClearBuf	SktClearBuf			False	False	
Socket	_sSOCKET			False	False	
SendStr	STRING[256]			False	False	
Packet	ARRAY[0..1999] OF byte			False	False	
SendSize	UINT			False	False	
RcvPacket	ARRAY[0..1999] OF BYTE			False	False	
RcvStr	string[256]			False	False	
Success	BOOL			False	False	
VAR_EXTERNAL						
_EIP_EtnOnline Sta	BOOL				True	

1-6-1-6-2.Program Body

```
IF state = 0 THEN
    IF _EIP_EtnOnlineSta THEN
            iSktCreate(Execute := FALSE, SrcUdpPort := 6000, Socket => Socket);
            state := 10;
    END_IF;
END_IF;

IF state = 10 THEN
    iSktCreate(Execute := TRUE, SrcUdpPort := 6000, Socket => Socket);
    IF NOT iSktCreate.Busy THEN
            IF iSktCreate.Done THEN
                    Socket.DstAdr.IpAdr :='192.168.250.200';
                    Socket.DstAdr.PortNo := 6000;
                    state := 20;
            END_IF;

            IF iSktCreate.Error THEN
                    state := 9000;
            END_IF;
    END_IF;
END_IF;

IF state = 20 THEN
    iSktClearBuf(Execute := FALSE, Socket := Socket);
    state := 30;
END_IF;

IF state = 30 THEN
    iSktClearBuf(Execute := TRUE, Socket := Socket);
    IF NOT iSktClearBuf.Busy THEN
            state := 40;
    END_IF;
END_IF;

IF state = 40 THEN
    iSktReceive(Execute := FALSE, Socket := Socket, Size := 2000, RcvDat := RcvPacket[0]);
    state := 50;
END_IF;

IF state = 50 THEN
    iSktReceive(Execute := TRUE, Socket := Socket, TimeOut := 20, Size := 2000, RcvDat := RcvPacket[0]);
    IF NOT iSktReceive.Busy THEN
            IF iSktReceive.Done THEN
                    AryByteTo(In := RcvPacket[0], Size := iSktReceive.RcvSize, OutVal := RcvStr);
                    IF RcvStr = 'Hello World' THEN
                            Success := TRUE;
                    ELSE
                            Success := FALSE;
                    END_IF;
                    state := 60;
            END_IF;

            IF iSktReceive.Error THEN
                    state := 9000;
            END_IF;
    END_IF;
END_IF;
```

```
58
59      IF state = 60 THEN
60          SendStr := 'Hello World';
61          SendSize := ToAryByte(In := SendStr, AryOut := Packet[0]);
62          iSktSend(Execute := FALSE, Socket := Socket, SendDat := Packet[0], Size := SendSize);
63          state := 70;
64      END_IF;
65
66      IF state = 70 THEN
67          iSktSend(Execute := TRUE, Socket := Socket, SendDat := Packet[0], Size := SendSize);
68          IF NOT iSktSend.Busy THEN
69                      IF iSktSend.Done THEN
70                              state := 20;
71                      END_IF;
72
73                      IF iSktSend.Error THEN
74                              state := 9000;
75                      END_IF;
76          END_IF;
77      END_IF;
78
79      IF state = 9000 THEN
80          IF Socket.Handle <> 0 THEN
81                      state := 9010;
82          ELSE
83                      state := 0;
84          END_IF;
85      END_IF;
86
87      IF state = 9010 THEN
88          iSktClose(Execute := FALSE, Socket := Socket);
89          state := 9020;
90      END_IF;
91
92      IF state = 9020 THEN
93          iSktClose(Execute := TRUE, Socket := Socket);
94
95          IF NOT iSktClose.Busy THEN
96                      state := 0;
97          END_IF;
98      END_IF;
```

1-6-2.Functions

1-6-2-1.FindAvailablePrinter

1-6-2-1-1.Variables

Name	Data Type	Edge	Initial Value	AT	Retain	Constant	Comment
VAR							
index	INT				False	False	Index for iterating
printerNr	INT				False	False	Available printer number
VAR_INPUTOUTPUT							
EN	BOOL	No Edge			False	False	
statusArray	ARRAY[0..19] OF ST_3DPrinterStatus	No Edge			False	False	
RETURN							
FindAvailablePrinter	INT						The array number for the available printer

1-6-2-1-2.Program Body

```
1       //Set default return value to -1, since this cannot naturally be achieved.
2       //Should the function return a -1, then there is no available printer to use.
3       //Usage of this function should thus be followed by an IF statement to check for -1.
4       printerNr := -1;
5
6       //Iterate over the array containing all printer statuses.
7       FOR index := SizeOfAry(statusArray)-1 TO 0 BY -1 DO
8           //Check to see if there is a printer that is ready for a new print and isn't in the "finished" state where the
print is yet to be retrieved.
9           IF (statusArray[index].ready = 'True') AND (statusArray[index].finished = 'False') THEN
10                  // This printer is ready to use, choose it as printer to assign work
11                  printerNr := index;
12          END_IF;
13      END_FOR;
14
15      //Return either available printer or -1.
16      FindAvailablePrinter := printerNr;
```

1-6-2-2.ProcessGcodeName

1-6-2-2-1.Variables

Name	Data Type	Edge	Initial Value	AT	Retain	Constant	Comment
VAR							
cursor	UINT				False	False	The current position in the string that is being worked from
productID	STRING[256]				False	False	The extracted product ID text
number	STRING[256]				False	False	The extracted number of products being printed
tempStr	STRING[256]				False	False	Temporary string used for processing
VAR_INPUTOUTPUT							
EN	BOOL	No Edge			False	False	
gcodeName	STRING[256]	No Edge			False	False	.GCODE name
RETURN							
ProcessGcode Name	STRING[256]						Returned string containing product ID and number. -1 if unknown. Separate into a struct with SubDelimiter afterwards

1-6-2-2-2.Program Body

```
1       // EXPECTED FORMAT: "P[ID]_ProductName_[number]pcs.gcode"
2       // For example:
3       //   - P1_BracketPair_1pcs.gcode
4       //   - P192_WaluigiBust_10000pcs.gcode
5       // Function is able to detect letters inside the numbers, and can remove a single comma or period, but no
other symbols.
6
7       //Check the name of the gcode to see if it can be recognized. It must pass these tests:
8       // - First letter is 'P'
9       // - Last 9 letters is 'pcs.gcode'
10      // - There exists an underscore in the gcode name
11      // - There exists a second underscore in the remaining text after the end of the last underscore.
12      // - There does not exist a third underscore in the remaining text after the end of the second underscore
13      //If it does not pass, return "-1;-1".
14      cursor := FIND(In1:=gcodeName, In2:='_');
15      IF LEFT(In:=gcodeName, L:=1)='P' AND RIGHT(In:=gcodeName, L:=9)='pcs.gcode' AND NOT(cursor=0) AND
NOT(FIND(In1:=RIGHT(In:=gcodeName, L:=LEN(gcodeName)-cursor), In2:='_')=0) AND FIND(In1:=RIGHT
(In:=RIGHT(In:=gcodeName, L:=LEN(gcodeName)-cursor), L:=LEN(RIGHT(In:=gcodeName, L:=LEN(gcodeName)-
cursor))-FIND(In1:=RIGHT(In:=gcodeName, L:=LEN(gcodeName)-cursor), In2:='_')), In2:='_')=0 THEN
16          //A recognized part is being printed, update the status of this printer.
17
18          //Find product ID of what is being printed.
19          //Find first underscore
20          cursor := FIND(In1:=gcodeName, In2:='_');
21          //Remove P at the start and everything after (and including) the underscore.
22          tempStr := MID(In:=gcodeName, L:=(cursor-2), P:=2);
23          //Check if there are any letters in the remaining text. If yes, returning -1.
24          IF (tempStr = ToLCase(In:=tempStr)) AND (tempStr = ToUCase(In:=tempStr)) THEN
25                  //No letter in the remaining numbers. Check for period or comma, and if one is detected,
remove it.
26                  IF NOT(FIND(In1:=tempStr, In2:=',')=0) THEN
27                          productID := DELETE(In:=tempStr, L:=1, P:=FIND(In1:=tempStr, In2:=','));
28                  ELSIF NOT(FIND(In1:=tempStr, In2:='.')=0) THEN
29                          productID := DELETE(In:=tempStr, L:=1, P:=FIND(In1:=tempStr, In2:='.'));
30                  ELSE
31                          productID := tempStr;
32                  END_IF;
33          ELSE
34                  productID := '-1';
35          END_IF;
36
37          tempStr := RIGHT(In:=gcodeName, L:=(LEN(gcodeName)-cursor));
                            //Extracts everything after the underscore
38          cursor := FIND(In1:=tempStr, In2:='_');
                                                                                    //Find next
underscore
39          tempStr := MID(In:=tempStr, L:=(LEN(tempStr)-cursor-LEN('pcs.gcode')), P:=(cursor+1));     //Removes
everything up until and including underscore, and pcs.gcode.
40          IF (tempStr = ToLCase(In:=tempStr)) AND (tempStr = ToUCase(In:=tempStr)) THEN
41                  IF NOT(FIND(In1:=tempStr, In2:=',')=0) THEN
42                          number := DELETE(In:=tempStr, L:=1, P:=FIND(In1:=tempStr, In2:=','));
43                  ELSIF NOT(FIND(In1:=tempStr, In2:='.')=0) THEN
44                          number := DELETE(In:=tempStr, L:=1, P:=FIND(In1:=tempStr, In2:='.'));
45                  ELSE
46                          number := tempStr;
47                  END_IF;
```

```
48      ELSE
49              number := '-1';
50      END_IF;
51
52      //Combine into a format that SubDelimiter can split up
53      ProcessGcodeName := CONCAT(In1:=productID, In2:=';', In3:=number);
54  ELSE
55      //Name is not recognized. Return unattainable value for ID and number.
56      ProcessGcodeName := '-1;-1';
57  END_IF;
```

1-7.Data

1-7-1.Data Types

Name	Type				Comment
ST_OrderDetails	STRUCT	NJ	Offset Byte	Offset Bit	
orderReceived	BOOL				
productID	UINT				
name	STRING[256]				
ST_MoviconCommands	STRUCT	NJ	Offset Byte	Offset Bit	
startDxfScript	BOOL				
startPrinterScript	BOOL				
ST_MoviconStatus	STRUCT	NJ	Offset Byte	Offset Bit	
ID1BracketPairNumber	INT				
ID1BracketPairPrinting	INT				
ID1BracketPairBuffer	INT				
ID2KeychainNumber	INT				
ID2KeychainPrinting	INT				
ID2KeychainBuffer	INT				
masterState	INT				
laserCutter00	INT				
mobileRobot00	INT				
mobileRobot01	INT				
mobileRobot02	INT				
cobot00	INT				
cobot01	INT				
cobot02	INT				
quattro00	INT				
viper00	INT				
viper01	INT				
printer00	INT				
printer01	INT				
printer02	INT				
printer03	INT				
printer04	INT				
printer05	INT				
printer06	INT				
printer07	INT				
printer08	INT				
printer09	INT				
printer10	INT				
printer11	INT				
printer12	INT				
printer13	INT				
printer14	INT				
printer15	INT				
printer16	INT				
printer17	INT				
printer18	INT				
printer19	INT				
ST_CsvTools	STRUCT	NJ	Offset Byte	Offset Bit	
writeCutterDxfStart	BOOL				
writeCutterDxfDone	BOOL				
writePrinterCommandStart	BOOL				

	writePrinterCommandDone	BOOL				
	readCutterStatusStart	BOOL				
	readCutterStatusDone	BOOL				
	readPrinterStatusStart	BOOL				
	readPrinterStatusDone	BOOL				
ST_CutterDxfExport		STRUCT	NJ	**Offset Byte**	**Offset Bit**	
	separator	STRING[256]				
	finaFileName	STRING[256]				
	templateName	STRING[256]				
	logoFileName	STRING[256]				
	xInsertPointLogo	STRING[256]				
	yInsertPointLogo	STRING[256]				
	xPixelSize	STRING[256]				
	yPixelSize	STRING[256]				
	xSizeInMm	STRING[256]				
	ySizeInMm	STRING[256]				
	rotationLogo	STRING[256]				
	textToInsert	STRING[256]				
	textStyle	STRING[256]				
	xInsertPointText	STRING[256]				
	yInsertPointText	STRING[256]				
	rotationText	STRING[256]				
	alignmentText	STRING[256]				
	textWidth	STRING[256]				
	textHeight	STRING[256]				
	strNameLen	STRING[256]				
ST_CutterStatus		STRUCT	NJ	**Offset Byte**	**Offset Bit**	
	working	STRING[256]				
	done	STRING[256]				
	error	STRING[256]				
ST_LDStatus		STRUCT	NJ	**Offset Byte**	**Offset Bit**	
	connected	BOOL				
	busy	BOOL				
	error	BOOL				
ST_LDTools		STRUCT	NJ	**Offset Byte**	**Offset Bit**	
	fleetManager	STRING[256]				
	connect	BOOL				
	taskList	ARRAY[0..10] OF STRING[256]				
	getFromKanbanStart	BOOL				
	getFromCutterStart	BOOL				
	getFromPrinterStart	BOOL				
	getFromKanbanDone	BOOL				
	getFromCutterDone	BOOL				
	getFromPrinterDone	BOOL				
	deliverProductStart	BOOL				
	deliverProductDone	BOOL				
	printerNr	INT				
	productID	INT				
	connectError	BOOL				
	connectSuccessful	BOOL				
ST_TMTools		STRUCT	NJ	**Offset Byte**	**Offset Bit**	
	connect	BOOL				
	connectSuccessful	BOOL				

	connectError	BOOL				
	assemblyStart	BOOL				
	assemblyDone	BOOL				
	productID	INT				
	armLD_jobStart	BOOL				
	armLD_jobDone	BOOL				
	arm1_jobStart	BOOL				
	arm1_jobDone	BOOL				
	arm2_jobStart	BOOL				
	arm2_jobDone	BOOL				
	armLD_jobNr	INT				
	arm1_jobNr	INT				
	arm2_jobNr	INT				
ST_3DPrinterCommand		STRUCT	NJ	**Offset Byte**	**Offset Bit**	
	separator	STRING[256]				
	IP	STRING[256]				
	argument	STRING[256]				
	command	STRING[256]				
	number	STRING[256]				
ST_3DPrinterStatus		STRUCT	NJ	**Offset Byte**	**Offset Bit**	
	IP	STRING[256]				
	connected	STRING[256]				
	printing	STRING[256]				
	ready	STRING[256]				
	operational	STRING[256]				
	pausing	STRING[256]				
	paused	STRING[256]				
	finished	STRING[256]				
	nozzleTemp	STRING[256]				
	bedTemp	STRING[256]				
	printJob	STRING[256]				
	rack	STRING[256]				
	posX	STRING[256]				
	posY	STRING[256]				
ST_3DPrinterCurrentPrintInfo		STRUCT	NJ	**Offset Byte**	**Offset Bit**	
	productID	INT				
	number	INT				
sHMI		STRUCT	NJ	**Offset Byte**	**Offset Bit**	
	Button	sHMI_Buttons				
	Lamp	sHMI_Lamps				
sHMI_Buttons		STRUCT	NJ	**Offset Byte**	**Offset Bit**	
	Auto	BOOL				
	Start	BOOL				
	Pause	BOOL				
	Resume	BOOL				
	Stop	BOOL				
	Home	BOOL				
	Connect	BOOL				
sHMI_Lamps		STRUCT	NJ	**Offset Byte**	**Offset Bit**	
	ModbusOK	BOOL				
	Connected	BOOL				
	RobotReady	BOOL				

1-7-2.Global Variables

Name	Data Type	Initial Value	AT	Retain	Constant	Network Publish	Comment
VAR_GLOBAL							
HMI_Comm	sHMI			True	False	Do not publish	
masterState	INT	0		False	False	Do not publish	Used to keep track of current production state
firstBackgroundRunDone	BOOL			False	False	Do not publish	Goes true when initial equipment status information has been obtained
customerOrder	ST_OrderDetails			False	False	Publish Only	Structure that holds order info coming from Movicon
csvBools	ST_CsvTools			False	False	Do not publish	Toolbox for .CSV-related commands
cutterDxfInfo	ST_CutterDxfExport			False	False	Do not publish	Structure for storing .DXF cutting information
cutterProductInfo	ARRAY[1..2] OF ST_CutterDxfExport	[(separator := 'sep=', finalFileName := 'RLtestPlate.dxf', templateName := 'NTNU_Plate_Template_v1.dxf', logoFileName := 'logo_ntnu_manulab.png', xInsertPointLogo := '1', yInsertPointLogo := '60', xPixelSize := '1157', yPixelSize := '456', xSizeInMm := '98', ySizeInMm := '38.6', rotationLogo := '0', textToInsert := 'Hans Christian H. Finnson', textStyle := 'NTNU-DIN', xInsertPointText := '50', yInsertPointText := '55', rotationText := '0', alignmentText := '2', textWidth := '78', textHeight := '7', strNameLen := '40'), (sep		False	True	Do not publish	Array holding default .DXF cutting information for each product

		arator := 'sep=', finaFileName := 'RLtestPlate.dxf', templateName := 'NTNU_KeyChain_Templace_v2.dxf', logoFileName := 'ntnu_logo_svart.png', xInsertPointLogo := '1.505', yInsertPointLogo := '18.7732', xPixelSize := '260', yPixelSize := '348', xSizeInMm := '22', ySizeInMm := '29.5', rotationLogo := '0', textToInsert := 'H C F', textStyle := 'NTNU-DIN', xInsertPointText := '12.5', yInsertPointText := '14', rotationText := '0', alignmentText := '2', textWidth := '25', textHeight := '2.5', strNameLen := '8')]					
cutterStatus	ST_CutterStatus			False	False	Do not publish	Structure containing the status of the cutters
printerCommand	ST_3DPrinterCommand	(separator := 'sep=;RLIP_Address;Command', IP := 'Argument$R$L192.168.250.200', argument := 'print', command := '/api/files/local/test/home.gcode', number := '1')		False	False	Do not publish	Structure holding text that is written as a command for the printers

	printerStatus	ARRAY[0..19] OF ST_3DPrinterStatus	[20((IP := '', connected := 'False', printing := '', ready := '', operational := '', pausing := '', paused := '', finished := '', nozzleTemp := '', bedTemp := '', printJob := '', rack := '', posX := '', posY := ''))]		False	False	Do not publish	Array that contains status of all 20 printers
	printerActivePrints	ARRAY[0..19] OF ST_3DPrinterCurrentPrintInfo			False	False	Do not publish	Array that contains the ID and number of parts being printed at all the different printers
	printerProductInfo	ARRAY[1..2] OF STRING[256]	['P1_BracketPair_1pcs.gcode', 'P2_KeychainFrame_1pcs.gcodc']		False	True	Do not publish	Array containing .GCODE filepaths for all product IDs
	LDCommands	ST_LDTools			False	False	Do not publish	Toolbox containing commands for LD communication
	TMCommands	ST_TMTools			False	False	Output	Toolbox containing commands for TM communication
	moviconSendStatus	ST_MoviconStatus			False	False	Publish Only	Structure holding all status information that is transmitted to Movicon
	moviconSendCommand	ST_MoviconCommands			False	False	Publish Only	Structure holding commands that are sent to Movicon. Used for starting Python scripts
	backgroundWritingCommand	BOOL			False	False	Do not publish	Bool used for preventing simultaneous .CSV writing in both Master and Background
	LDTakloStatus	ST_LDStatus			False	False	Do not publish	Structure holding status information for the LD-130CT, nicknamed Taklo
	LDFinnsonStatus	ST_LDStatus			False	False	Do not publish	Structure holding status information for the first LD-90, nicknamed Finnson
	LDRingstadStatus	ST_LDStatus			False	False	Do not publish	Structure holding status information for the second LD-90, nicknamed Ringstad
	TM_ArmLD	OEN\nRobot\nTM\sRobot			False	False	Do not publish	Structure used for communicating with the TM5M located ontop of the LD-130CT, nicknamed Olur
	TM_Arm1	OEN\nRobot\nTM\sRobot			False	False	Do not publish	Structure used for communicating with the first TM5, nicknamed Dolur
	TM_Arm2	OEN\nRobot\nTM\sRobot			False	False	Do not publish	Structure used for communicating with the second TM5, nicknamed Doffur