

Master Program_5

Author: HC Finnson, HC Ringstad, Fredrik Taklo

Created: 20/02/2020 15:08:02

Last Modified: 12/05/2020 18:44:54

Comment:

Sysmac Studio Module Version: 1.29.1.0

Table of Contents

Master Program_5	3
1.IPC	3
1-3.Controller Setup	3
1-3-2.Built-in EtherNet/IP Port Settings	3
1-5.Task Settings	5
1-6.POUS	6
1-6-1.Programs	6
1-6-1-1.Master	6
1-6-1-1-1.Variables	6
1-6-1-1-2.Program Body	8
1-6-1-2.Background	16
1-6-1-2-1.Variables	16
1-6-1-2-2.Program Body	17
1-6-1-3.commsCSV	22
1-6-1-3-1.Variables	22
1-6-1-3-2.writeCutterDxf	23
1-6-1-3-3.writePrinterCommand	24
1-6-1-3-4.readCutterStatus	25
1-6-1-3-5.readPrinterStatus	26
1-6-1-4.commsLD	29
1-6-1-4-1.Variables	29
1-6-1-4-2.ConnectToLDs	31
1-6-1-4-3.getFromPrinter	33
1-6-1-4-4.getFromKanban	37
1-6-1-4-5.getFromCutter	39
1-6-1-4-6.deliverProduct	41
1-6-1-5.commsTM	43
1-6-1-5-1.Variables	43
1-6-1-5-2.ModbusTCP_Setup	45
1-6-1-5-3.doTask	52
1-6-1-6.UDP_Socket	53
1-6-1-6-1.Variables	53
1-6-1-6-2.Program Body	54
1-6-2.Functions	56
1-6-2-1.FindAvailablePrinter	56
1-6-2-1-1.Variables	56
1-6-2-1-2.Program Body	57
1-6-2-2.ProcessGcodeName	58
1-6-2-2-1.Variables	58
1-6-2-2-2.Program Body	59
1-7.Data	61
1-7-1.Data Types	61
1-7-2.Global Variables	65

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 Exceede d Detectio n	Task Timeout Detection Time	Executio n 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
printerPartsInStock	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		
cutterDxInfo	ST_CutterDxInfoExport			False		
printerStatus	ARRAY[0..19] OF ST_3DPrinterStatus			False		
printerCommand	ST_3DPrinterCommand			False		
moviconSendStatus	ST_MoviconStatus			False		
LDCCommands	ST_LDTools			False		
moviconSendCommands	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_CutterDxInfoExport			True		
printerProductInfo	ARRAY[1..2] OF STRING[256]			True		

backgroundWritingCommand	BOOL				False	
productStatus	ARRAY[1..2] OF ST_ProductStatus				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;
```

```

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 -----
65 // Idle, awaiting order
66 IF masterState = 1000 THEN
67
68     IF customerOrder.orderReceived THEN
69         //Order is received from Movicon, copy information to local variable for processing and reset
70         customerOrder to prevent a loop
71         orderCurrent := customerOrder;
72         customerOrder.orderReceived := FALSE;
73         customerOrder.productID := 0;
74         customerOrder.name := "";
75         masterState := 2000;
76
77     ELSIF NOT(customerOrder.orderReceived) AND customerQueue[1].orderReceived THEN
78         // There is no new order from Movicon, but the first position in the queue is filled with an order.
79         // Copy the order from the first spot in the queue into the local variable for processing. Signify
80         that this was an order started from the queue.
81         orderCurrent := customerQueue[1];
82         startedFromQueue := TRUE;
83         masterState := 2000;
84
85     END_IF;
86
87 END_IF;
88
89 // Add order to the back of the queue when a new order is received while the program is busy processing
90 another.
91
92 // Reset customerOrder after completing to open up for a new order if one comes
93 // Note, this always runs in the background.
94 IF NOT(masterState = 1000) AND customerOrder.orderReceived THEN
95
96     customerQueueLength := customerQueueLength + 1;
97     customerQueue[customerQueueLength] := customerOrder;
98     customerOrder.orderReceived := FALSE;
99     customerOrder.productID := 0;
100    customerOrder.name := "";
101
102 END_IF;
103
104
105 //2000-----
106 //Process customer order
107 // 2000 = setup
108 // 2100 = process dxf for cutter
109 // 2200 = check kanban storage for printer
110
111 IF masterState = 2000 THEN
112     // Setup
113     masterState := 2100;
114
115 END_IF;
116 IF masterState = 2100 THEN
117     // Cutter
118     // Depending on the product ID, either apply the preset for the nameplate or the keychain for export to
119     Dxf, and add the customer's name to it.
120     IF orderCurrent.productID > 0 AND orderCurrent.productID <= SizeOfAry(cutterProductInfo) THEN
121         cutterDxflInfo := cutterProductInfo[orderCurrent.productID];

```

```

111      cutterDxflInfo.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      IF productStatus[orderCurrent.productID].number > 0 THEN
122          //Product ID exists and there is at least 1 part in storage
123          printerPartsInStore := TRUE;
124          masterState := 3000;
125      ELSIF productStatus[orderCurrent.productID].number = 0 THEN
126          //Product ID exists but there are no parts in storage
127          printerPartsInStore := FALSE;
128          masterState := 3000;
129      ELSIF productStatus[orderCurrent.productID].number < 0 THEN
130          //Fault in number tracking, reset to 0 and continue as if there are no parts in store.
131          printerPartsInStore := FALSE;
132          masterState := 3000;
133      ELSE
134          //Non-existent product ID
135          masterState := 9220;
136      END_IF;
137  END_IF;
138
139
140 //3000-----
141 // Assign work to cutter, printer, robots, etc
142 // 3000 = setup
143 // 3100 = assign cutter task
144 // 3200 = assign printer task
145 // 3300 = assign LD robots
146 // 3400 = assign TM robots
147 IF masterState = 3000 THEN
148     //Setup
149     orderChosenPrinterNr := -1;
150     masterState := 3100;
151 END_IF;
152 IF masterState = 3100 THEN
153     //Cutter
154     //With the information added to the dxflInfo variable and ready for export, start the csv conversion
program
155     //After that is done, tell Movicon to start the Python script
156     IF NOT(csvBools.writeCutterDxfStart) AND NOT(csvBools.writeCutterDxfDone) THEN
157         csvBools.writeCutterDxfStart := TRUE;
158     END_IF;
159     IF csvBools.writeCutterDxfStart AND csvBools.writeCutterDxfDone THEN
160         csvBools.writeCutterDxfStart := FALSE;
161         moviconSendCommand.startDxfScript := TRUE; //Note, needs to be turned off at some point.
Could do in cleanup
162         masterState := 3200;
163     END_IF;
164 END_IF;
165 IF masterState = 3200 THEN
166     //Printer
//Regardless of whether needed part is or is not in store, print a new one.
167

```

```

168 //Iterate through the array of printer statuses to find the first printer that is ready and available to use.
169 orderChosenPrinterNr := FindAvailablePrinter(statusArray:=printerStatus);
170 IF orderChosenPrinterNr = -1 THEN
171     //If no available printer was found, go to error.
172     masterState := 9320;
173 ELSE
174     //Otherwise, continue as planned
175     masterState := 3210;
176 END_IF;
177 END_IF;
178 IF masterState = 3210 AND NOT(backgroundWritingCommand) THEN
179     //Now that an available printer has been found, create a command to send to the printer.
180     printerCommand.separator := 'sep='; //Ensures that the file can be opened in excel if required
181     printerCommand.IP := CONCAT(ln1:="$R$L", ln2:=printerStatus[orderChosenPrinterNr].IP); //$R$L is
newline
182     printerCommand.command := 'print';
183     printerCommand.number := '1';
184     printerCommand.argument := printerProductInfo[orderCurrent.productID];
185     printerActivePrints[orderChosenPrinterNr].productID := orderCurrent.productID;
186     printerActivePrints[orderChosenPrinterNr].number := STRING_TO_UINT(printerCommand.number);
187     productStatus[orderCurrent.productID].printing := productStatus[orderCurrent.productID].printing + 1;
188     masterState := 3220;
189 END_IF;
190 IF masterState = 3220 AND NOT(backgroundWritingCommand) THEN
191     //Run the csv conversion program to put the previously created printer command into a csv, then tell
Movicon to run the python script
192     IF NOT(csvBools.writePrinterCommandStart) AND NOT(csvBools.writePrinterCommandDone) THEN
193         csvBools.writePrinterCommandStart := TRUE;
194     END_IF;
195     IF csvBools.writePrinterCommandStart AND csvBools.writePrinterCommandDone THEN
196         csvBools.writePrinterCommandStart := FALSE;
197         moviconSendCommand.startPrinterScript := TRUE; //Note, needs to be turned off at some point,
could do in cleanup
198         masterState := 3300;
199     END_IF;
200 END_IF;
201 IF masterState = 3300 THEN
202     // LD robots
203     IF printerPartsInStore THEN
204         //Command LD robot to pick up parts
205         //May use CASE if rack is too large to only pick from one spot
206         //Could push to 4300?
207         LDCommands.productID := orderCurrent.productID;
208         LDCommands.getFromKanbanStart := TRUE;
209         masterState := 4000;
210     ELSIF NOT(printerPartsInStore) THEN
211         //Part needed is not in store
212         masterState := 9330; //Temporary error state, wait until printing is done and kanban has the
given part.
213     END_IF;
214 END_IF;
215
216 //4000-----
217 // Wait for processes to complete, then continue production
218 // Pick up the different parts and move to assembly table
219 // 4000 initial setup, wait until part is retrieved from kanban
220 // 4100 wait for cutter to be done, then flag it as done
221 // 4200 no printer jobs
222 // 4300 Retrieve item from cutter

```

```

223 IF masterState = 4000 THEN
224     IF LDCommands.getFromKanbanDone THEN
225         orderPrinterDone := TRUE;
226         LDCommands.getFromKanbanStart := FALSE;
227         productStatus[orderCurrent.productID].number := productStatus
[orderCurrent.productID].number - 1;
228
229         masterState := 4100;
230     END_IF;
231 END_IF;
232 IF masterState = 4100 THEN
233     //Cutter
234     //Cutter status csv is being continuously read/refreshed in Background. Wait until it reads that the cutting
is done
235     IF cutterStatus.done='True' THEN
236         //Cutter is done. Move to next task
237         orderCutterDone := TRUE;
238         masterState := 4200;
239     ELSIF cutterStatus.error='True' THEN
240         //Error happened, go to appropriate state.
241         masterState := 9410;
242     END_IF;
243 END_IF;
244 IF masterState = 4200 THEN
245     //No more necessary printer tasks to do
246     masterState := 4300;
247 END_IF;
248 IF masterState = 4300 THEN
249     //LD robots
250     masterState := 4310;
251 END_IF;
252 IF masterState = 4310 THEN
253     //Retrive item from cutter
254     IF orderCutterDone AND NOT(LDCommands.getFromKanbanStart) AND NOT
(LDCommands.getFromCutterDone) THEN
255         LDCommands.productID := orderCurrent.productID;
256         LDCommands.getFromCutterStart := TRUE;
257     END_IF;
258     IF orderCutterDone AND LDCommands.getFromCutterStart AND LDCommands.getFromCutterDone
THEN
259         LDCommands.getFromCutterStart := FALSE;
260         masterState := 5000;
261     END_IF;
262 END_IF;
263
264 //5000-----
265 // Order assembly and delivery
266 IF masterState = 5000 THEN
267     TMCommands.productID := orderCurrent.productID;
268
269     TMCommands.arm1_jobNr := orderCurrent.productID;
270     TMCommands.arm1_jobStart := TRUE;
271
272     TMCommands.arm2_jobNr := orderCurrent.productID;
273     TMCommands.arm2_jobStart := TRUE;
274
275     masterState := 5100;
276 END_IF;
277 IF masterState = 5100 THEN

```

```

278  IF TMCommands.arm1_jobDone AND TMCommands.arm2_jobDone THEN
279      TMCommands.arm1_jobStart := FALSE;
280      TMCommands.arm2_jobStart := FALSE;
281      TMCommands.productID := 0;
282      LDCommands.productID := orderCurrent.productID;
283      LDCommands.deliverProductStart := TRUE;
284      masterState := 5200;
285  END_IF;
286 END_IF;
287
288 IF masterState = 5200 THEN
289     IF LDCommands.deliverProductDone THEN
290         LDCommands.deliverProductStart := FALSE;
291         masterState := 6000;
292     END_IF;
293 END_IF;
294
295
296 //6000-----
297 // Update status and clean up
298
299 IF masterState = 6000 THEN
300     //Reset local variables
301     printerPartsInStore := FALSE;
302     orderCutterDone := FALSE;
303     orderPrinterDone := FALSE;
304     moviconSendCommand.startDxfScript := FALSE;
305     moviconSendCommand.startPrinterScript := FALSE;
306     orderCurrent.orderReceived := FALSE;
307     orderCurrent.productID := 0;
308     orderCurrent.name := "";
309
310     IF startedFromQueue THEN
311         masterState := 6050;
312     ELSIF NOT(startedFromQueue) THEN
313         masterState := 6100;
314     END_IF;
315 END_IF;
316 IF masterState = 6050 THEN
317     //If the current process was started from a queue, then move all spots in the queue down one level and
318     //reset the last.
319     FOR index := 1 TO customerQueueLength-1 DO
320         customerQueue[index] := customerQueue[index+1];
321     END_FOR;
322     customerQueue[customerQueueLength].orderReceived := FALSE;
323     customerQueue[customerQueueLength].productID := 0;
324     customerQueue[customerQueueLength].name := "";
325     customerQueueLength := customerQueueLength - 1;
326     startedFromQueue := FALSE;
327
328     //Queue process complete, continue with order cleanup
329     masterState := 6100;
330 END_IF;
331 IF masterState = 6100 THEN
332     //Cleanup complete, return to idle
333     masterState := 1000;
334 END_IF;
335
336 //9000-----

```

```
337 // Error handling
338 // Actual errors, handling "no printers/cutters available"/"can't complete task", etc
339 // 9X00, replace X with whatever X000 master state the program was in when error occurred.
340 // 90Y0, replace Y with whatever 0Y00 master state the program was in when error occurred.
341 // 900Z, replace Z with whatever 00Z0 master state the program was in when error occurred.
342
343 IF masterState = 9010 THEN
344     //Cutter is unable to connect
345     //Send to Movicon?
346     //Wait a while, then set master state to 100 to try again.
347     //Maybe send "Unrecoverable error" status to Movicon if this happens too many times
348     IF NOT(refreshTimer.In) THEN
349         refreshTimer.In := TRUE;
350         refreshTimer.PT := T#5s;
351     END_IF;
352     IF refreshTimer.In AND refreshTimer.Q THEN
353         refreshTimer.In := FALSE;
354         masterState := 100;
355     END_IF;
356 END_IF;
357 IF masterState = 9020 THEN
358     //No printers available in rack
359     //Send to movicon?
360     //Wait a while, then set master state to 200 to try again
361     //Maybe send "Unrecoverable error" status to Movicon if this happens too many times
362     IF NOT(refreshTimer.In) THEN
363         refreshTimer.In := TRUE;
364         refreshTimer.PT := T#5s;
365     END_IF;
366     IF refreshTimer.In AND refreshTimer.Q THEN
367         refreshTimer.In := FALSE;
368         masterState := 200;
369     END_IF;
370 END_IF;
371 IF masterState = 9030 THEN
372     //Unable to connect to LDs
373     //Send to movicon?
374     //Wait a while, then set master state to 300.
375     //Maybe send "Unrecoverable error" status to Movicon if this happens too many times
376     IF NOT(refreshTimer.In) THEN
377         refreshTimer.In := TRUE;
378         refreshTimer.PT := T#5s;
379         LDCommands.connect := FALSE;
380     END_IF;
381     IF refreshTimer.In AND refreshTimer.Q THEN
382         refreshTimer.In := FALSE;
383         masterState := 300;
384     END_IF;
385 END_IF;
386 IF masterState = 9040 THEN
387     //Unable to connect to TMs
388     //Send to movicon?
389     //Wait a while, then set master state to 300.
390     //Maybe send "Unrecoverable error" status to Movicon if this happens too many times
391     IF NOT(refreshTimer.In) THEN
392         refreshTimer.In := TRUE;
393         refreshTimer.PT := T#5s;
394         TMCommands.connect := FALSE;
395     END_IF;
```

```
396 IF refreshTimer.In AND refreshTimer.Q THEN
397     refreshTimer.In := FALSE;
398     masterState := 400;
399 END_IF;
400 END_IF;
401
402 IF masterState = 9210 THEN
403     //Invalid product ID from cutter, return to idle.
404     masterState := 1000;
405 END_IF;
406 IF masterState = 9220 THEN
407     //Invalid product ID from printer, return to idle.
408     masterState := 1000;
409 END_IF;
410
411
412 IF masterState = 9320 THEN
413     //No available printer is found, do something about this.
414     //Set printer text to "no available printer"? Prompt asking whether to cancel or wait?
415     //For now, loop printer search until a printer becomes available.
416 IF NOT(refreshTimer.In) THEN
417     refreshTimer.In := TRUE;
418     refreshTimer.PT := T#1s;
419 END_IF;
420 IF refreshTimer.In AND refreshTimer.Q THEN
421     refreshTimer.In := FALSE;
422     orderChosenPrinterNr := FindAvailablePrinter(statusArray:=printerStatus);
423     IF NOT(orderChosenPrinterNr = -1) THEN
424         masterState := 3210;
425     END_IF;
426 END_IF;
427 END_IF;
428 IF masterState = 9330 THEN
429     //Part required to pick up is not in stock. Wait until part has finished printing and added to storage
430     //Ask in movicon whether the customer wants to wait or cancel in the meantime?
431 IF orderCurrent.productID = 1 AND moviconSendStatus.ID1BracketPairNumber > 0 THEN
432     LDCommands.getFromKanbanStart := TRUE;
433     LDCommands.productID := orderCurrent.productID;
434     printerPartsInStore := TRUE;
435     masterState := 4000;
436 ELSIF orderCurrent.productID = 2 AND moviconSendStatus.ID2KeychainNumber > 0 THEN
437     LDCommands.getFromKanbanStart := TRUE;
438     LDCommands.productID := orderCurrent.productID;
439     printerPartsInStore := TRUE;
440     masterState := 4000;
441 END_IF;
442 END_IF;
443
444 IF masterState = 9410 THEN
445     //Cutter has given off an error before completing.
446     //Cannot be fixed without access to physical hardware to know what I can and cannot do about it .
447     masterState := 9410;
448 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
printerTempArra y	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
firstCutterRead Done	BOOL			False	False	Cutter status has been successfully read for the first time.
firstPrinterRead Done	BOOL			False	False	Printer status has been successfully read for the first time.
VAR EXTERNAL						
printerStatus	ARRAY[0..19] OF ST_3DPrinterStatus				False	
moviconSendSta tus	ST_MoviconStatus				False	
printerActivePri nts	ARRAY[0..19] OF ST_3DPrinterCurrentPrintInfo				False	
printerCommand	ST_3DPrinterCom mand				False	
moviconSendCo mmand	ST_MoviconComm ands				False	
csvBools	ST_CsvTools				False	
firstBackground RunDone	BOOL				False	
printerProductIn fo	ARRAY[1..2] OF STRING[256]				True	
masterState	INT				False	
LDCommands	ST_LDTools				False	
LDTakloStatus	ST_LDStatus				False	
LDFinnsonStatu s	ST_LDStatus				False	
LDRingstadStat us	ST_LDStatus				False	
backgroundWrit ingCommand	BOOL				False	
TMCommands	ST_TMTools				False	
cutterStatus	ST_CutterStatus				False	
productStatus	ARRAY[1..2] OF ST_ProductStatus				False	

1-6-1-2.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),
38             OutStruct:=printerActivePrints[index], Delimiter:=_eDELIMITER#_SEMICOLON);
39         ELSIF printerStatus[index].printing='False' AND printerStatus[index].finished='False' THEN
40             //Nothing is being printed here, update status.
41             printerActivePrints[index].productID := 0;
42             printerActivePrints[index].number := 0;
43         END_IF;
44     END_FOR;
45
46 //Confirm that buffer doesn't use unrealistic values, correct if it does
47 FOR index := 1 TO SizeOfAry(productStatus) DO
48     IF productStatus[index].buffer > 20 THEN
49         productStatus[index].buffer := 20;
50     ELSIF productStatus[index].buffer < 0 THEN
51         productStatus[index].buffer := 0;
52     END_IF;
53 END_FOR;
54
55 // Check if printer's bed and nozzle temperatures aren't too high. If yes, tell printer to do an emergency
56 // shutdown.
57 // Protect against malicious or stupid intentions causing high enough temperatures to damage or

```

```

destroy equipment
57      FOR printerNr := 0 TO SizeOfAry(printerStatus)-1 DO
58          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
59              //Printer with too large temperature detected
60              printerCommand.separator := 'sep=';
61              printerCommand.IP := CONCAT(In1:='$R$L', In2:=printerStatus[printerNr].IP);
62              printerCommand.command := 'shutdown';
63              //printerCommand.argument := 'emergency';
64              csvBools.writePrinterCommandStart := TRUE;
65          END_IF;
66      END_FOR;
67
68 // Check if a printer has finished printing. If yes, bring the part to kanban storage
69      FOR printerNr := SizeOfAry(printerStatus)-1 TO 0 BY -1 DO
70          IF (printerStatus[printerNr].finished = 'True') AND NOT(LDCommands.getFromPrinterStart) AND
NOT(LDCommands.getFromPrinterDone) THEN
71              // This printer has finished a print and hasn't been retrieved.
72              LDCommands.getFromPrinterStart := TRUE;
73              LDCommands.printerNr := index;
74          END_IF;
75          IF (printerStatus[printerNr].finished = 'True') AND LDCommands.getFromPrinterStart AND
LDCommands.getFromPrinterDone THEN
76              //The print has been picked up
77              productStatus[printerActivePrints[printerNr].productID].number := productStatus
[printerActivePrints[printerNr].productID].number + printerActivePrints[printerNr].number;
78              productStatus[printerActivePrints[printerNr].productID].printing := productStatus
[printerActivePrints[printerNr].productID].printing - printerActivePrints[printerNr].number;
79
80          //Tell printer that the part is picked up and that the printer is ready for use again.
Doing this will set finished to False.
81          printerCommand.separator := 'sep=';
82          printerCommand.IP := CONCAT(In1:='$R$L', In2:=printerStatus[printerNr].IP);
83          printerCommand.command := 'retrievedPrint';
84          csvBools.writePrinterCommandStart := TRUE;
85          LDCommands.getFromPrinterStart := FALSE;
86
87          //To prevent issues for another finished print, exit for-loop to prevent further finished
prints from being found.
88          //This way, it's guaranteed to only work on one print at a time.
89          EXIT;
90
91      END_IF;
92  END_FOR;
93
94
95 //Check buffers, print up to 5 new parts (per printer) if current stock is smaller than buffer
96  FOR index := 1 TO SizeOfAry(productStatus) DO
97      IF (productStatus[index].number+productStatus[index].printing) < productStatus[index].buffer
AND NOT(masterState=3210) AND NOT(masterState=3220) THEN
98          //Current number of parts in store + parts printing is not enough to fill the buffer
99          IF NOT(csvBools.writePrinterCommandStart) AND NOT
(csvBools.writePrinterCommandDone) THEN
100              //IF statements ensure that no other part of the program is currently trying to
write to the same file
101              printerNr := -1; //Set unreachable default value to check for no available
printer
102              printerNr := FindAvailablePrinter(statusArray:=printerStatus);
103              IF NOT(printerNr = -1) THEN

```

```

104 //If there is an available printer, use start sending write commands to
it
105 printerCommand.separator := 'sep=';
106 printerCommand.IP := CONCAT(ln1:='$R$L', ln2:=printerStatus
[printerNr].IP);
107 printerCommand.command := 'print';
108 printerCommand.argument := printerProductInfo[index];
109 //Find number of parts required left to print, counting current in
storage and currently printing
110 partsToPrint := productStatus[index].buffer - productStatus
[index].number - productStatus[index].printing;
111 //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.
112 printerCommand.number := INT_TO_STRING(LIMIT(MN:=INT#1,
In:=partsToPrint, MX:=INT#5));
113 csvBools.writePrinterCommandStart := TRUE;
114 backgroundWritingCommand := TRUE;
115 END_IF;
116 END_IF;
117 IF csvBools.writePrinterCommandStart AND csvBools.writePrinterCommandDone
THEN
118 //Note, method for adding to local database of prints removed because it
now updates in the top.
119 productStatus[index].printing := productStatus[index].printing +
STRING_TO_INT(printerCommand.number);
120 moviconSendCommand.startPrinterScript := TRUE; //Note, must be turned off
eventually
121 csvBools.writePrinterCommandStart := FALSE;
122 backgroundWritingCommand := FALSE;
123 END_IF;
124 EXIT;
125 END_IF;
126 END_FOR;
127
128 -----
129 //Send updated statuses to Movicon
130 //Master state
131 moviconSendStatus.masterState := masterState;
132
133 //Product statuses
134 moviconSendStatus.ID1BracketPairNumber := productStatus[1].number;
135 moviconSendStatus.ID1BracketPairPrinting := productStatus[1].printing;
136 moviconSendStatus.ID2KeychainNumber := productStatus[2].number;
137 moviconSendStatus.ID2KeychainPrinting := productStatus[2].printing;
138
139 //Update buffer sizes in program
140 productStatus[1].buffer := moviconSendStatus.ID1BracketPairBuffer;
141 productStatus[2].buffer := moviconSendStatus.ID2KeychainBuffer;
142
143 //LD robots
144 IF LDFinnsonStatus.connected AND NOT(LDFinnsonStatus.busy) THEN
145     moviconSendStatus.mobileRobot01 := 1;
146 END_IF;
147 IF LDFinnsonStatus.busy THEN
148     moviconSendStatus.mobileRobot01 := 2;
149 END_IF;
150 IF LDFinnsonStatus.error OR NOT(LDFinnsonStatus.connected) THEN

```

```

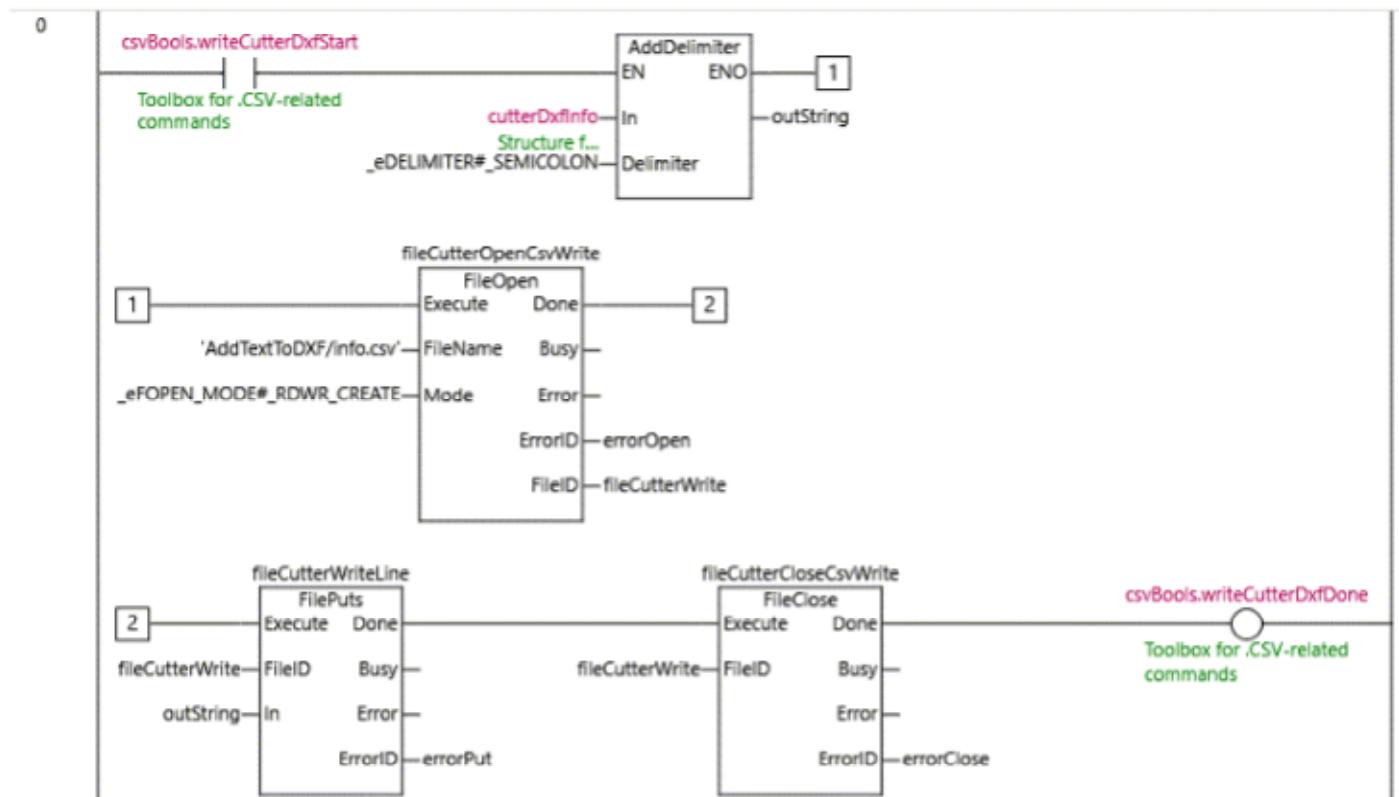
153         moviconSendStatus.mobileRobot01 := 0;
154     END_IF;
155
156     IF LDTakloStatus.connected AND NOT(LDTakloStatus.busy) THEN
157         moviconSendStatus.mobileRobot00 := 1;
158     END_IF;
159     IF LDTakloStatus.busy THEN
160         moviconSendStatus.mobileRobot00 := 2;
161     END_IF;
162     IF LDTakloStatus.error OR NOT(LDTakloStatus.connected) THEN
163         moviconSendStatus.mobileRobot00 := 0;
164     END_IF;
165
166     IF LDRingstadStatus.connected AND NOT(LDRingstadStatus.busy) THEN
167         moviconSendStatus.mobileRobot02 := 1;
168     END_IF;
169     IF LDRingstadStatus.busy THEN
170         moviconSendStatus.mobileRobot02 := 2;
171     END_IF;
172     IF LDRingstadStatus.error OR NOT(LDRingstadStatus.connected) THEN
173         moviconSendStatus.mobileRobot02 := 0;
174     END_IF;
175
176 //TM Arms
177 IF TMCommands.connectSuccessful AND NOT(TMCommands.armLD_jobStart) THEN
178     moviconSendStatus.cobot00 := 1;
179 END_IF;
180 IF TMCommands.connectSuccessful AND NOT(TMCommands.arm1_jobStart) THEN
181     moviconSendStatus.cobot01 := 1;
182 END_IF;
183 IF TMCommands.connectSuccessful AND NOT(TMCommands.arm2_jobStart) THEN
184     moviconSendStatus.cobot02 := 1;
185 END_IF;
186 IF TMCommands.armLD_jobStart THEN
187     moviconSendStatus.cobot00 := 2;
188 END_IF;
189 IF TMCommands.arm1_jobStart THEN
190     moviconSendStatus.cobot01 := 2;
191 END_IF;
192 IF TMCommands.arm2_jobStart THEN
193     moviconSendStatus.cobot02 := 2;
194 END_IF;
195 IF TMCommands.connectError OR NOT(TMCommands.connectSuccessful) THEN
196     moviconSendStatus.cobot00 := 0;
197     moviconSendStatus.cobot01 := 0;
198     moviconSendStatus.cobot02 := 0;
199 END_IF;
200
201 //Cutter
202 IF cutterStatus.done='True' AND cutterStatus.working='False' THEN
203     moviconSendStatus.laserCutter00 := 1;
204 END_IF;
205 IF cutterStatus.working='True' THEN
206     moviconSendStatus.laserCutter00 := 2;
207 END_IF;
208 IF cutterStatus.error='True' THEN
209     moviconSendStatus.laserCutter00 := 0;
210 END_IF;

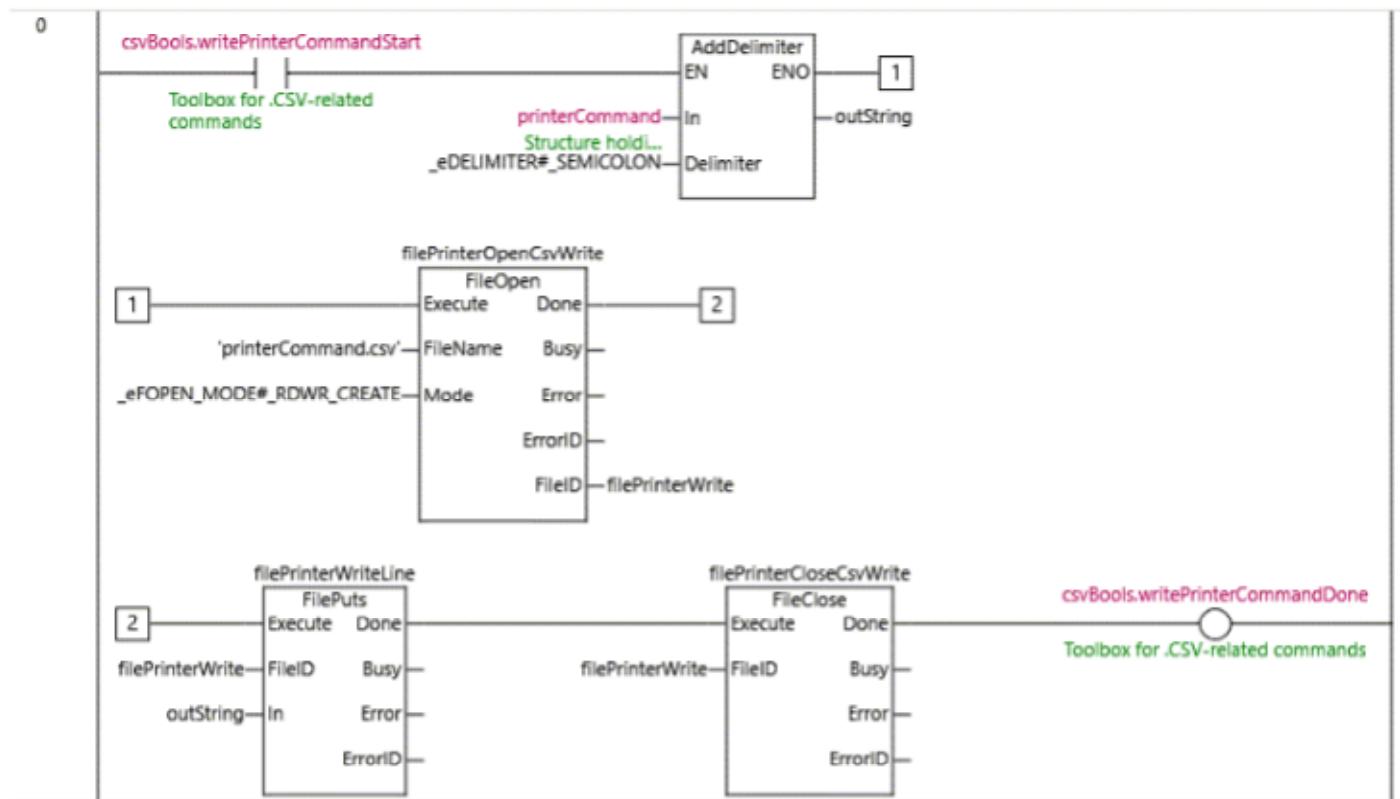
```

```
211
212 //Printers
213 FOR index :=0 TO SizeOfAry(printerStatus)-1 DO
214     //Update printer status
215     // 0 = error
216     // 1 = idle
217     // 2 = printing
218     IF printerStatus[index].ready='True' THEN
219         printerTempArray[index] := 1;
220     END_IF;
221     IF printerStatus[index].printing='True' OR printerStatus[index].finished='True' THEN
222         printerTempArray[index] := 2;
223     END_IF;
224     IF printerStatus[index].connected='False' OR printerStatus[index].operational='False' OR
printerStatus[index].pausing='True' OR printerStatus[index].paused='True' THEN
225         printerTempArray[index] := 0;
226     END_IF;
227 END_FOR;
228 moviconSendStatus.printer00 := printerTempArray[0]; moviconSendStatus.printer10 := printerTempArray
[10];
229 moviconSendStatus.printer01 := printerTempArray[1]; moviconSendStatus.printer11 := printerTempArray
[11];
230 moviconSendStatus.printer02 := printerTempArray[2]; moviconSendStatus.printer12 := printerTempArray
[12];
231 moviconSendStatus.printer03 := printerTempArray[3]; moviconSendStatus.printer13 := printerTempArray
[13];
232 moviconSendStatus.printer04 := printerTempArray[4]; moviconSendStatus.printer14 := printerTempArray
[14];
233 moviconSendStatus.printer05 := printerTempArray[5]; moviconSendStatus.printer15 := printerTempArray
[15];
234 moviconSendStatus.printer06 := printerTempArray[6]; moviconSendStatus.printer16 := printerTempArray
[16];
235 moviconSendStatus.printer07 := printerTempArray[7]; moviconSendStatus.printer17 := printerTempArray
[17];
236 moviconSendStatus.printer08 := printerTempArray[8]; moviconSendStatus.printer18 := printerTempArray
[18];
237 moviconSendStatus.printer09 := printerTempArray[9]; moviconSendStatus.printer19 := printerTempArray
[19];
238 // ^Thanks Movicon for neither accepting array inputs nor structs inside other structs, would use the
earlier for loop for assigning if it did.
239 END_IF;
```

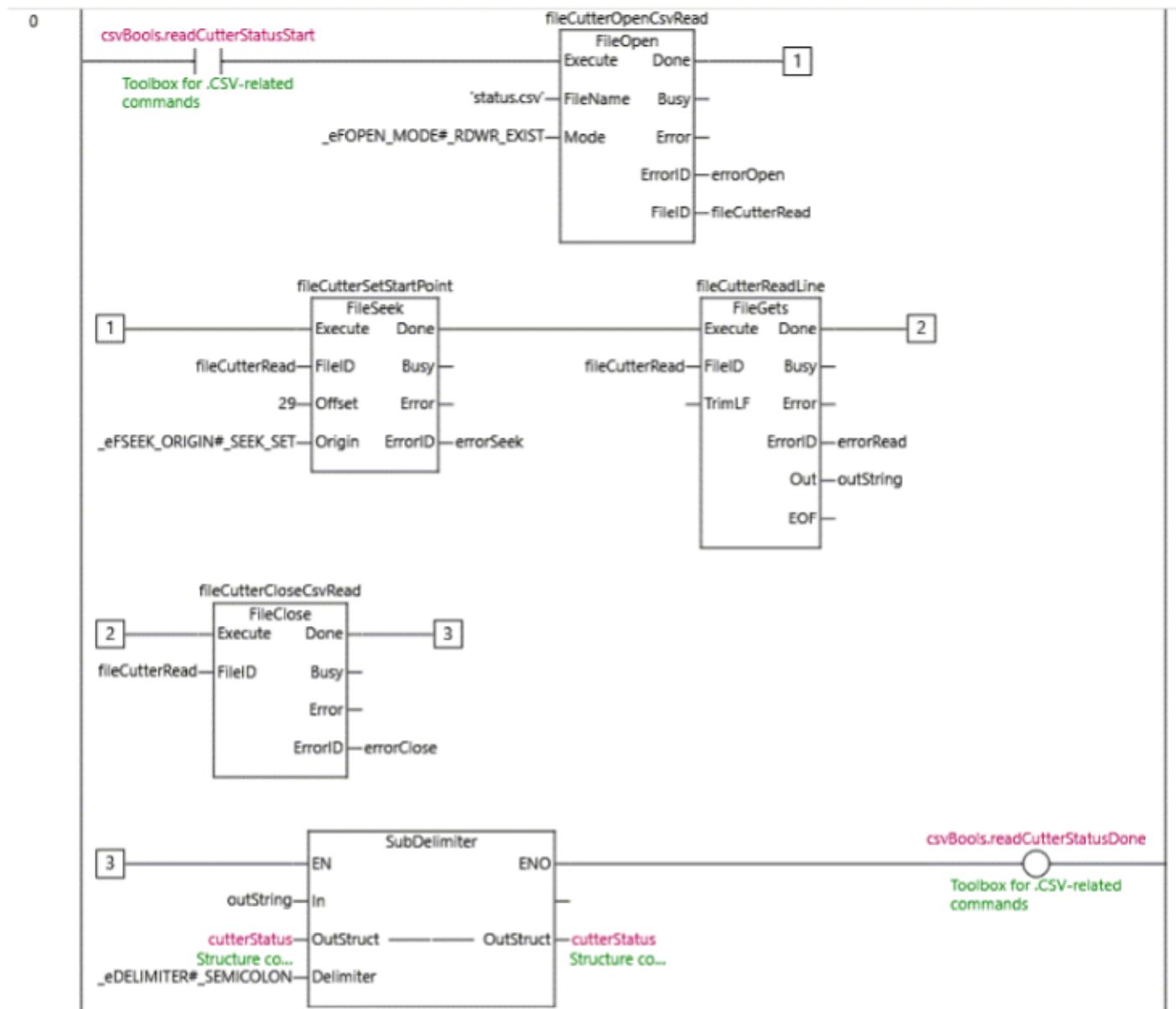
1-6-1-3.commsCSV**1-6-1-3-1.Variables**

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	
errorSck	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						
cutterDxInfo	ST_CutterDxExport				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      //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,
22   fileName:='printerStatus.csv',Mode:=_RDWR_EXIST, ErrorID=>errorOpen,
23   FileID=>filePrinterRead);
24   IF NOT(filePrinterOpenCsvRead.Busy) THEN
25     IF filePrinterOpenCsvRead.Done THEN
26       // Open success
27
28       state := 21;
29     END_IF;
30   IF filePrinterOpenCsvRead.Error THEN
31     // Open failed
32     state := 110;
33   END_IF;
34 END_IF;
35 // Set the start point where reading will happen using the FileSeek function
36 // block
37 // State 20 = Category encompassing all FileSeek functions
38 // State 21 = Initiate function block
39 // State 22 = Execute function block
40 IF state = 21 THEN
41   filePrinterSeek(Execute:=FALSE);
42   state := 22;
43 END_IF;
44 IF state = 22 THEN
45   filePrinterSeek(Execute:=TRUE, FileID:=filePrinterOpenCsvRead.FileID,
46   Offset:=offset, Origin:=_SEEK_SET);
47   IF NOT(filePrinterSeek.Busy) THEN
48     IF filePrinterSeek.Done THEN
49       // Seek success
50       state := 31;
51     END_IF;
52   IF filePrinterSeek.Error THEN
53     // Seek failed
54     state := 120;

```

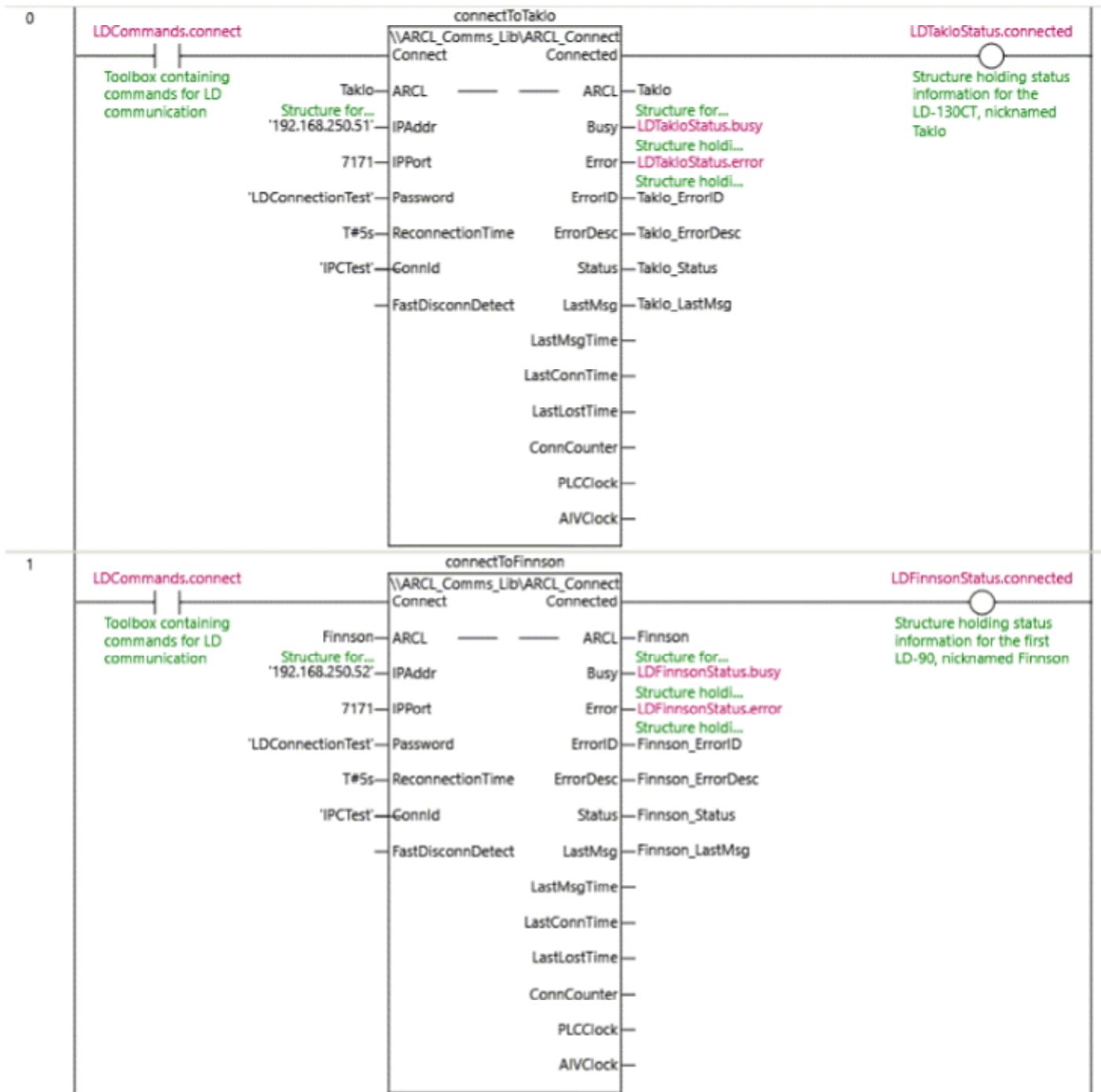
```
53     END_IF;
54     END_IF;
55     END_IF;
56
57 // Read the rest of the line from the start point set in state 20
58 // State 30 = Category encompassing all FileGets functions
59 // State 31 = Initiate function block
60 // State 32 = Execute function block
61 IF state = 31 THEN
62     filePrinterReadLine(Execute:=FALSE);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

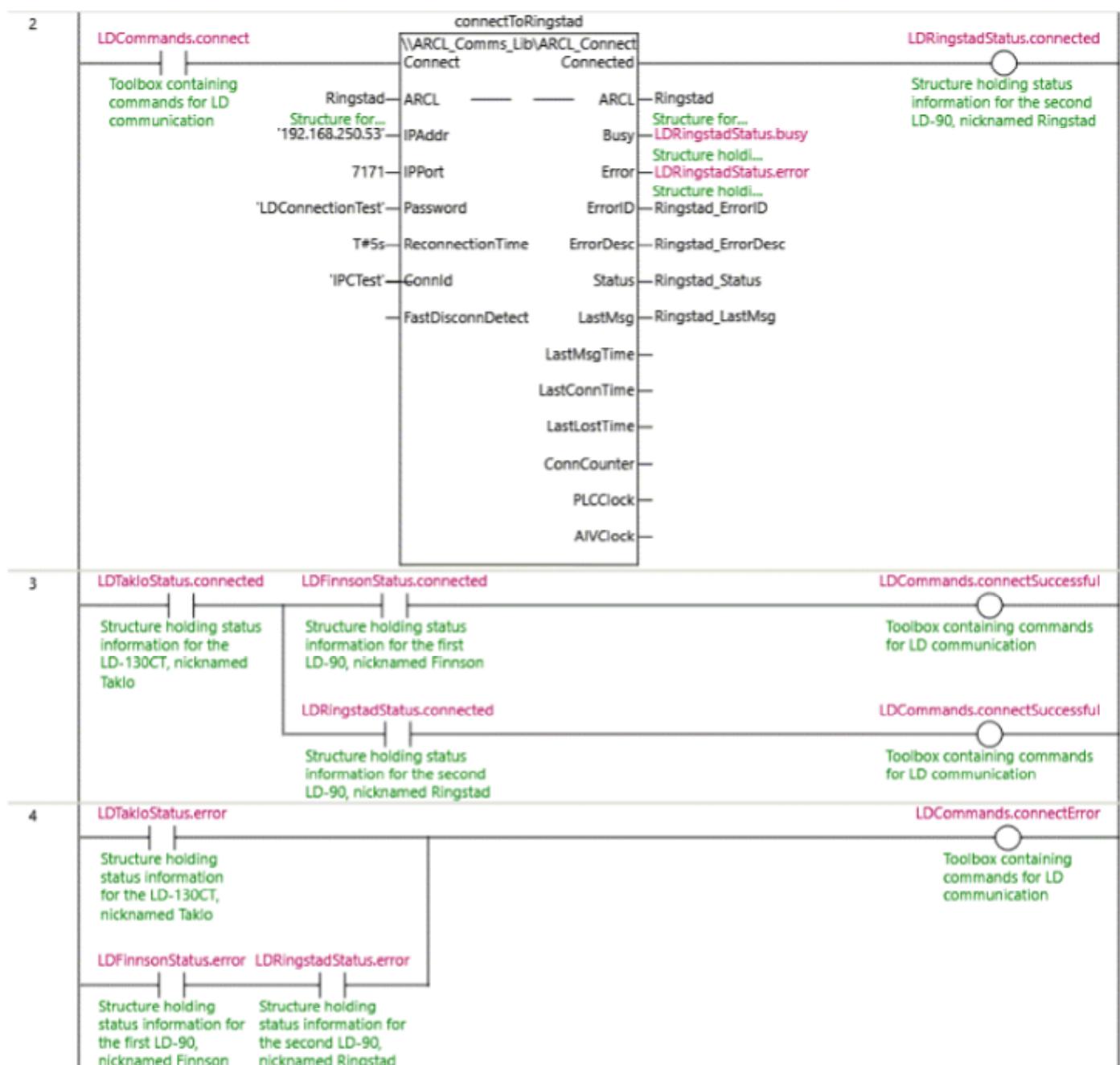
```
Done=>csvBools.readPrinterStatusDone);
109  IF NOT(filePrinterCloseCsvRead.Busy) THEN
110    IF filePrinterCloseCsvRead.Done THEN
111      // Closed successfully
112      state := 60;
113    END_IF;
114    IF filePrinterCloseCsvRead.Error THEN
115      // Close failed
116      state := 150;
117 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

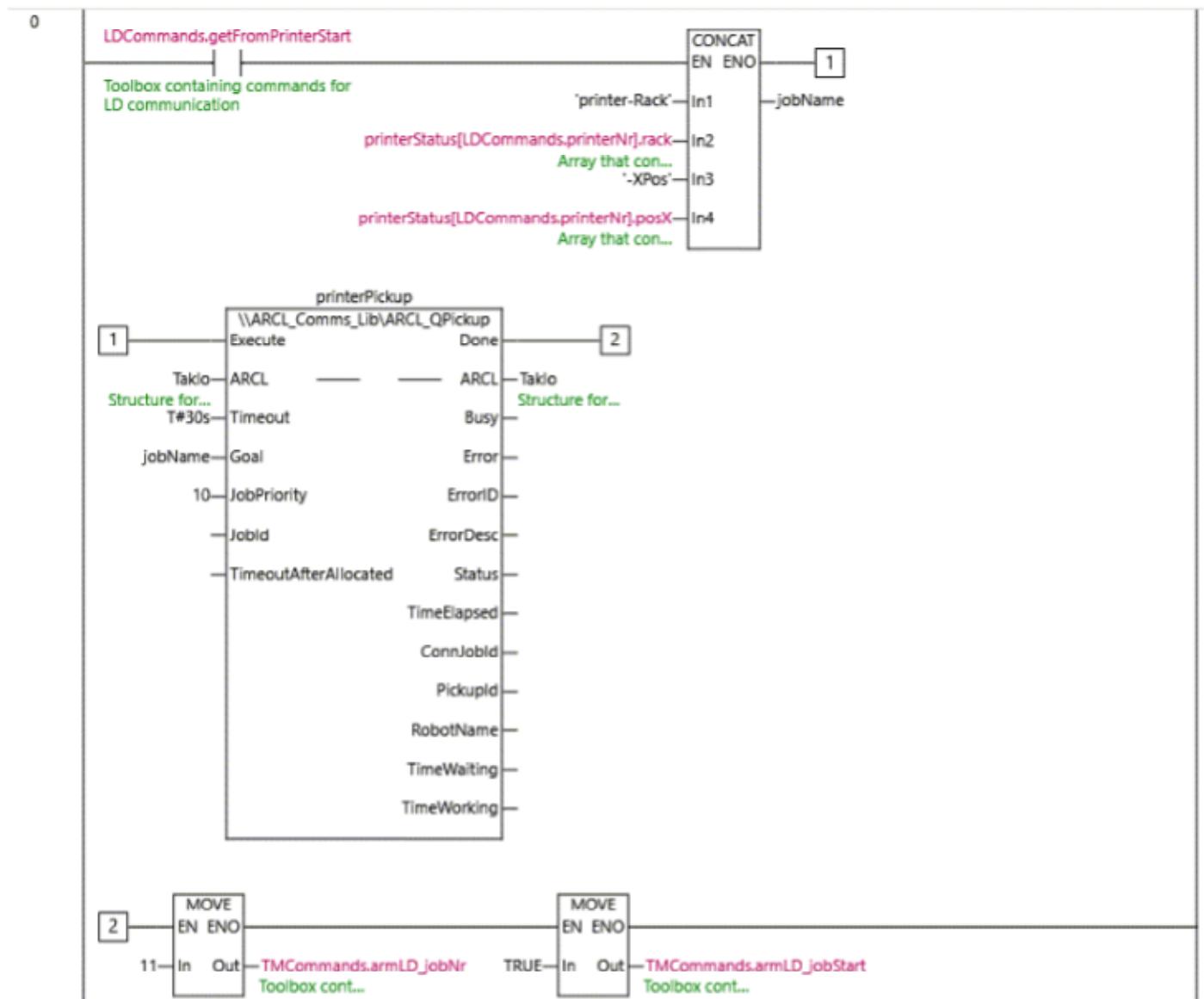
1-6-1-4.commsLD**1-6-1-4-1.Variables**

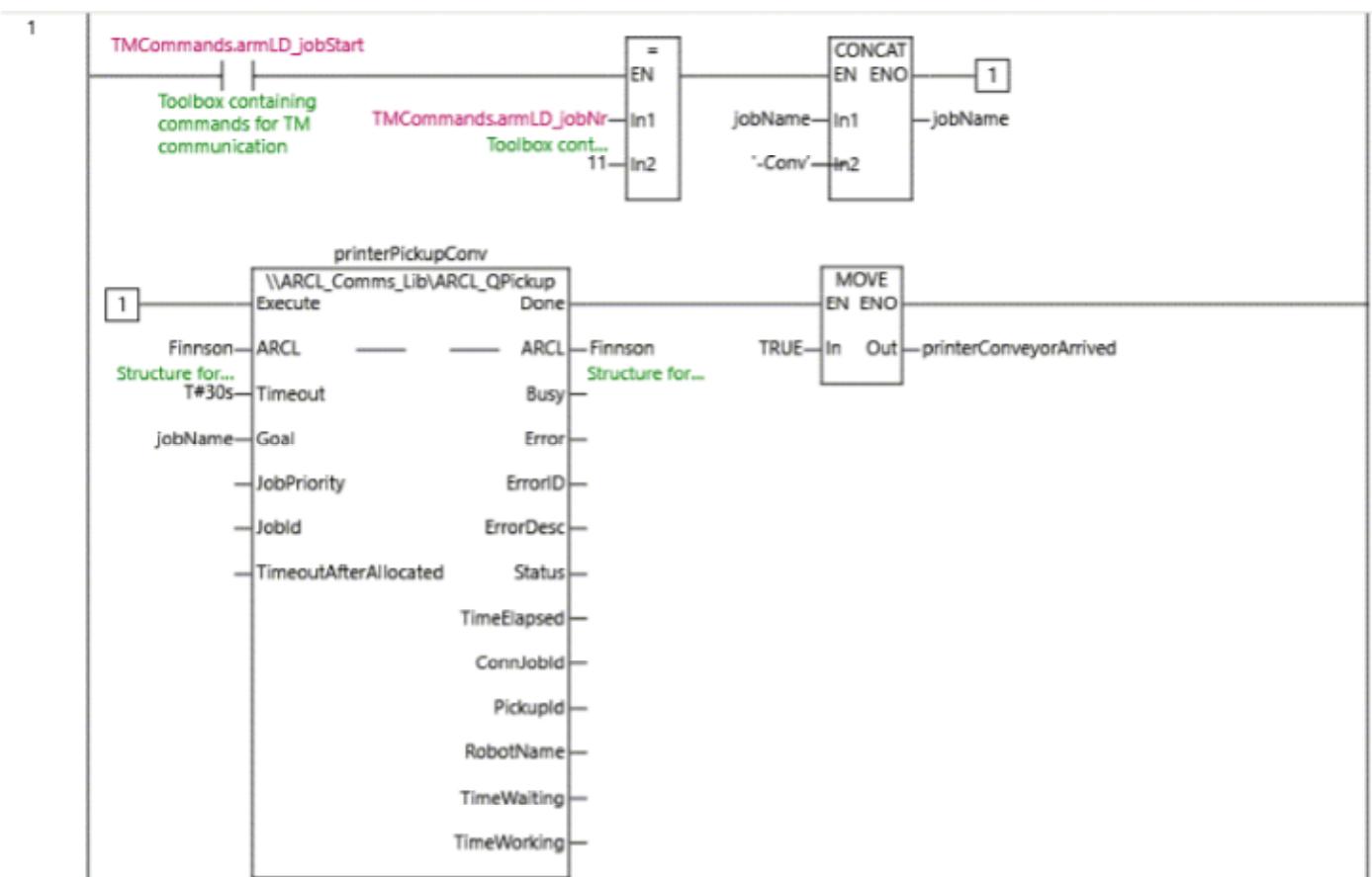
Name	Data Type	Initial Value	AT	Retain	Constant	Comment
VAR						
Taklo	ARCL_Comms_Lib\ARCL_ST_Connection			False	False	
Taklo_Status	STRING[30]			False	False	
Taklo_ErrorID	WORD			False	False	
Taklo_ErrorDes	STRING[30]			False	False	
Taklo_LastMsg	STRING[200]			False	False	
connectToTaklo	ARCL_Comms_Lib\ARCL_Connect			False	False	
connectToFinns	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	

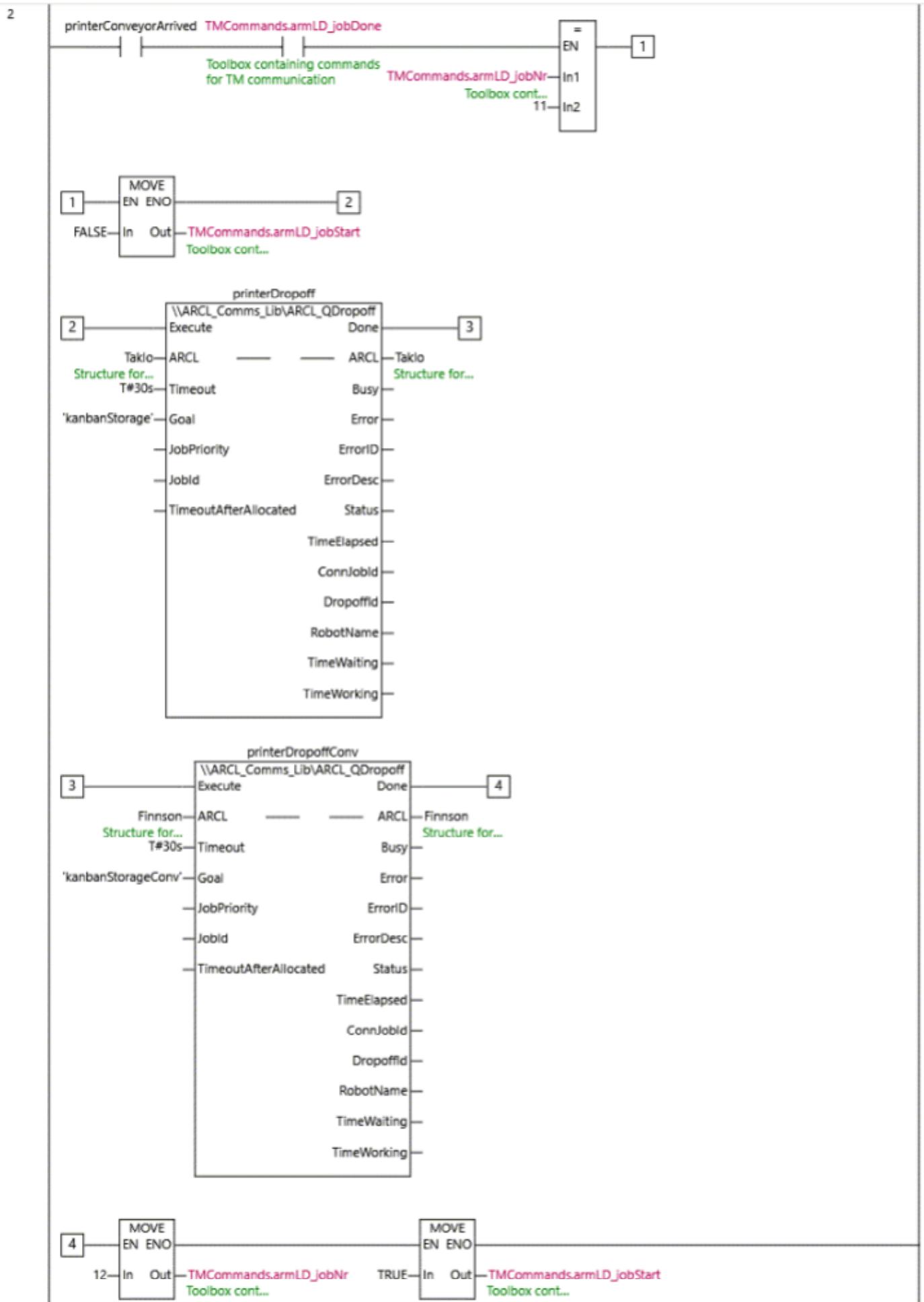
printerStatus	ARRAY[0..19] OF ST_3DPrinterStatus				False	
LDTakloStatus	ST_LDStatus				False	
LDFinnsonStatu s	ST_LDStatus				False	
LDRingstadStat us	ST_LDStatus				False	

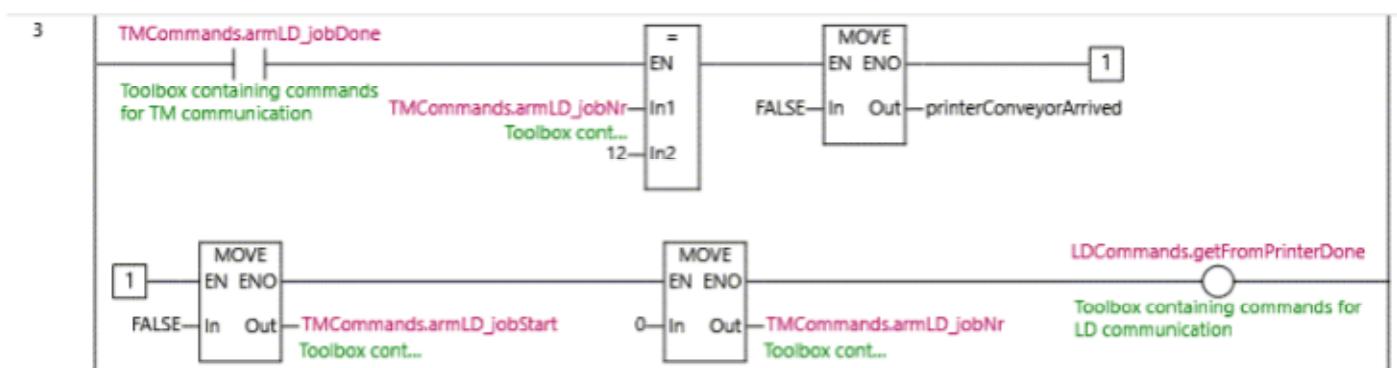
1-6-1-4-2.ConnectToLDs

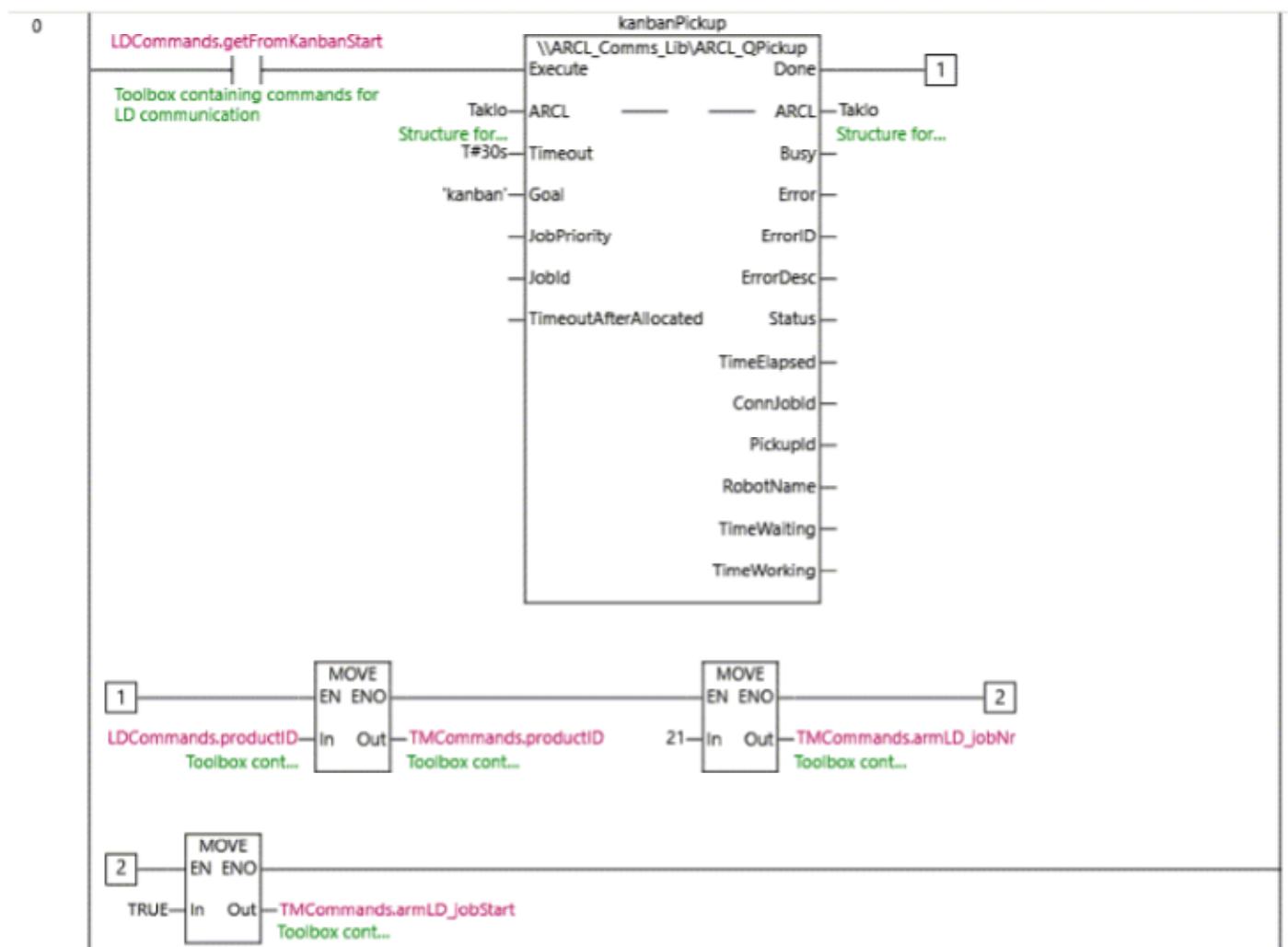


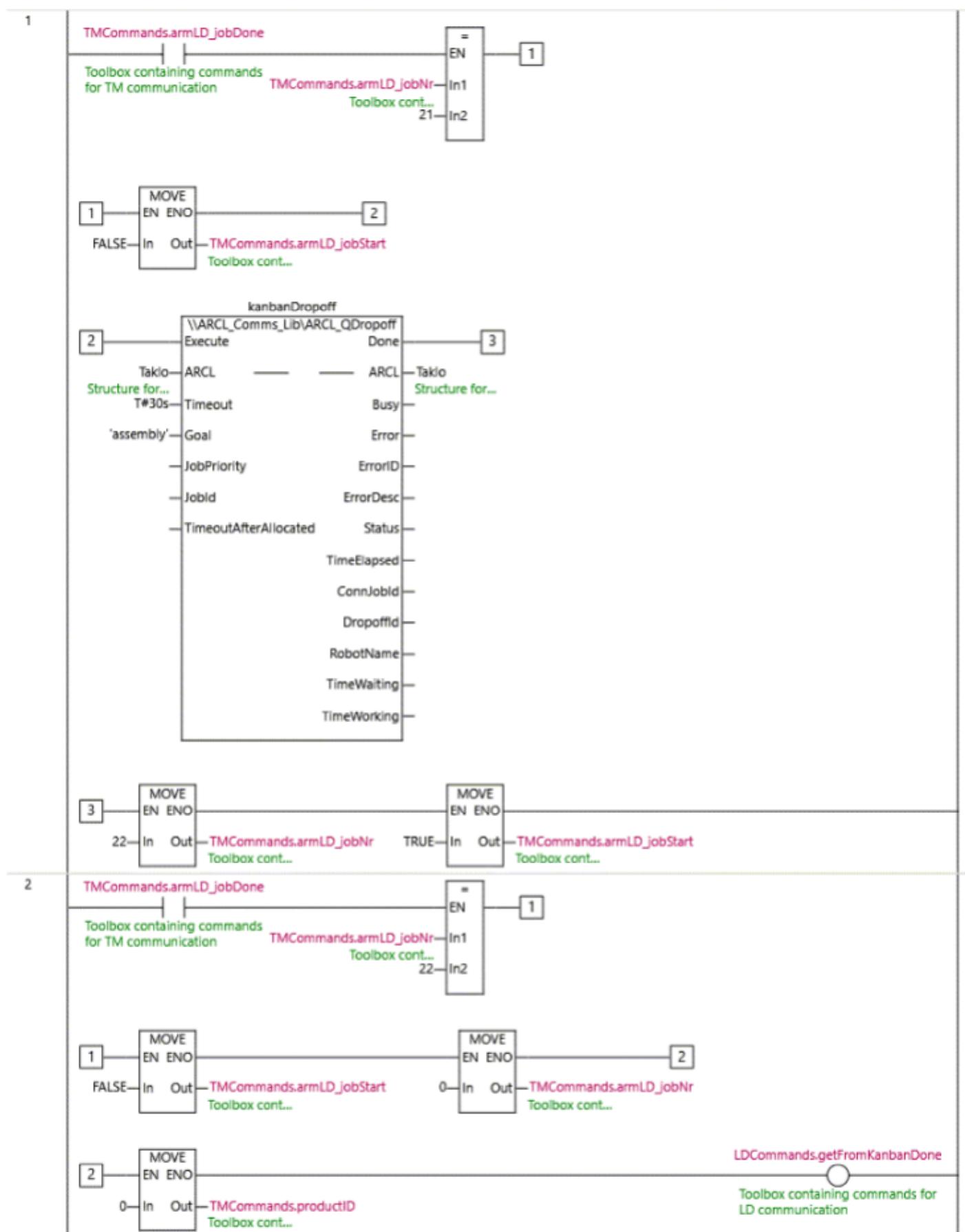
1-6-1-4-3.getFromPrinter

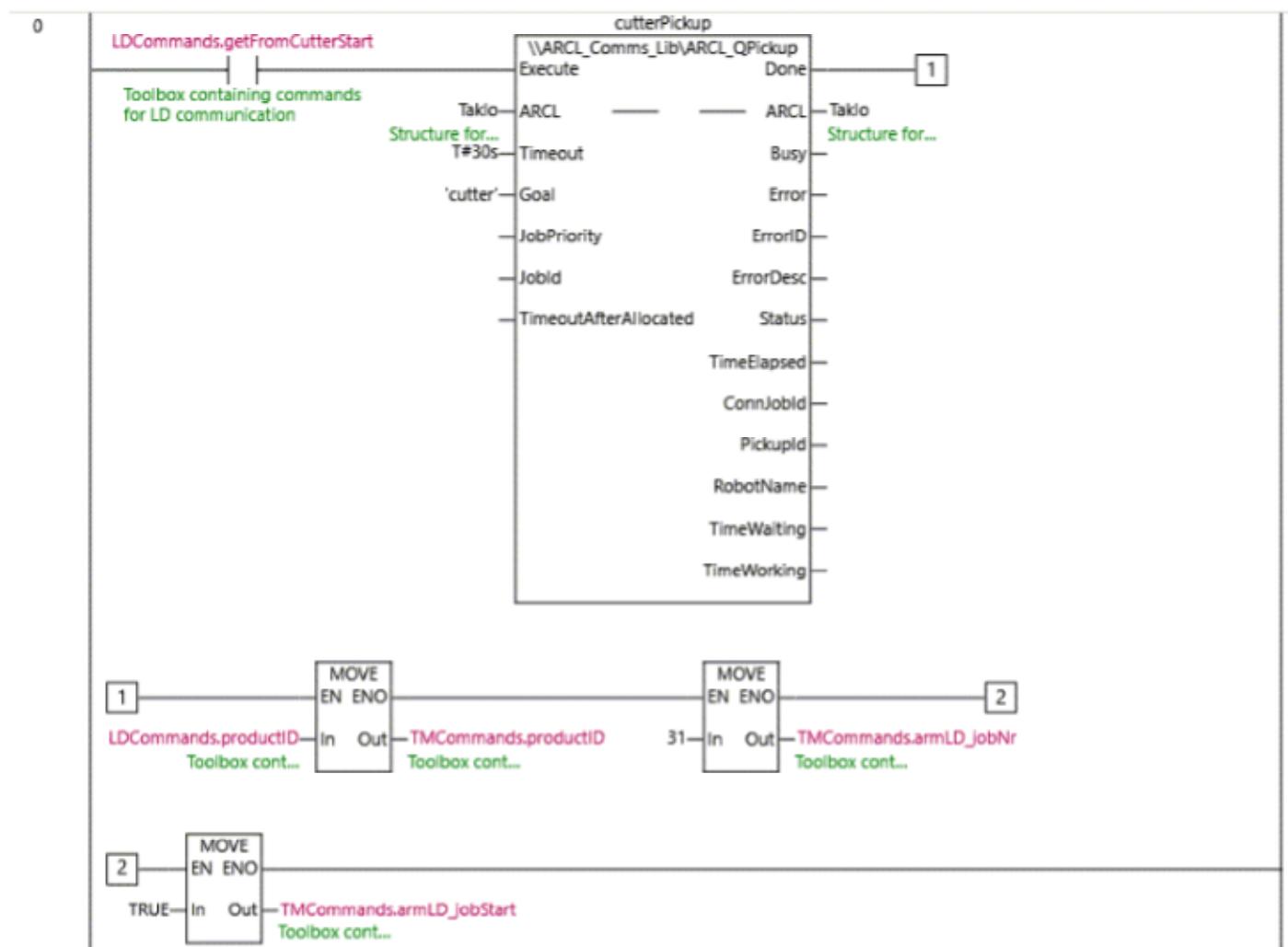


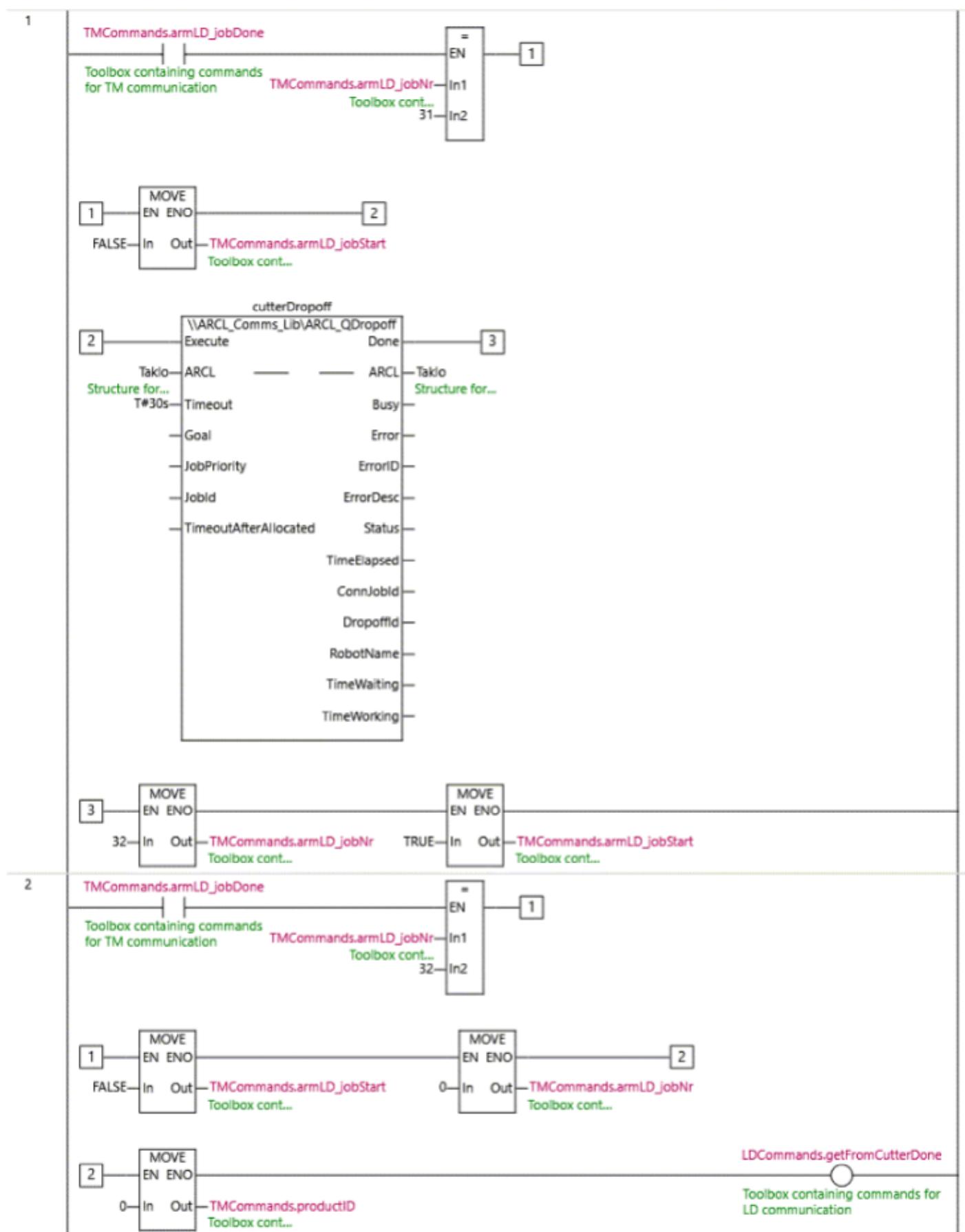


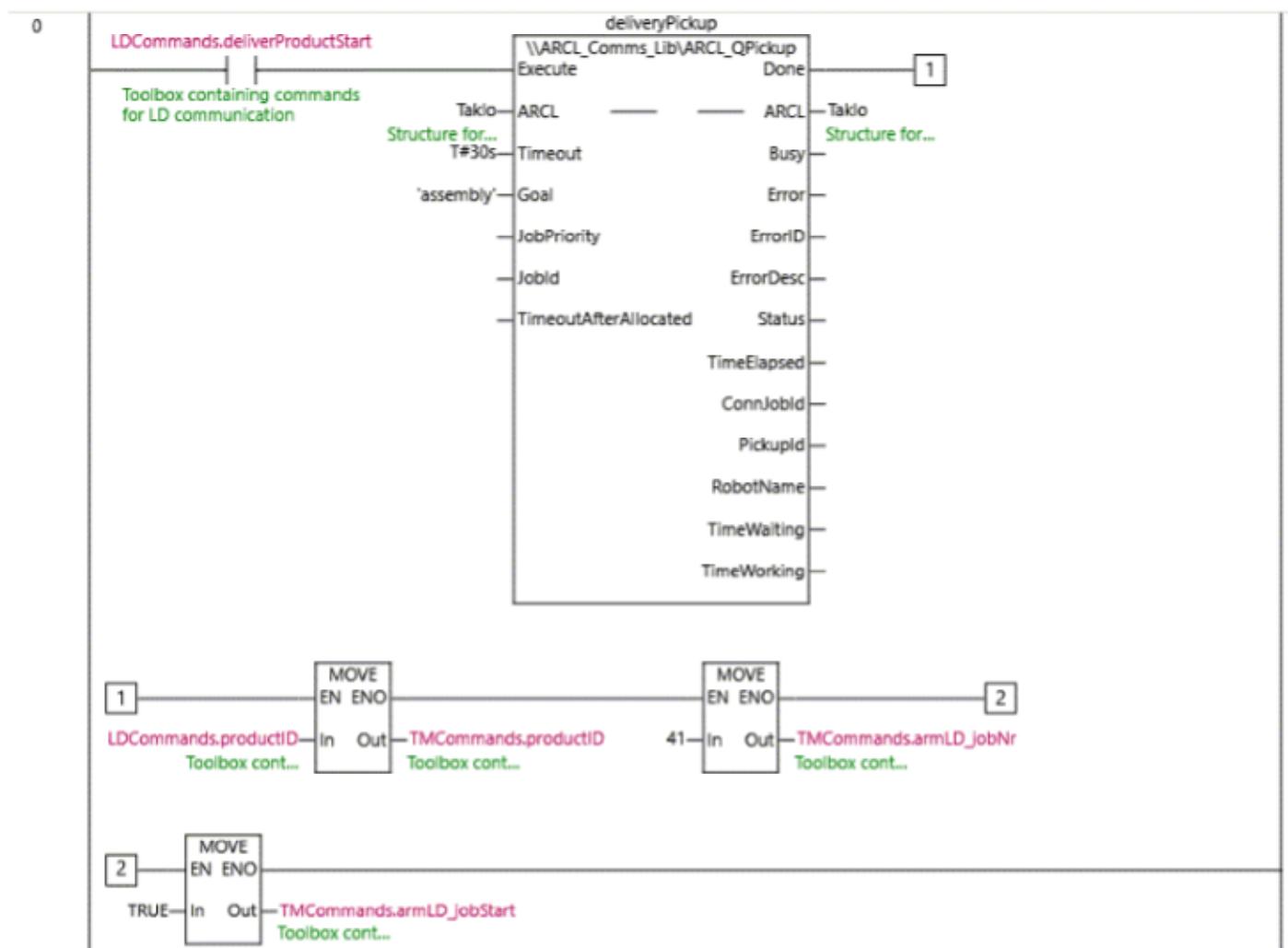


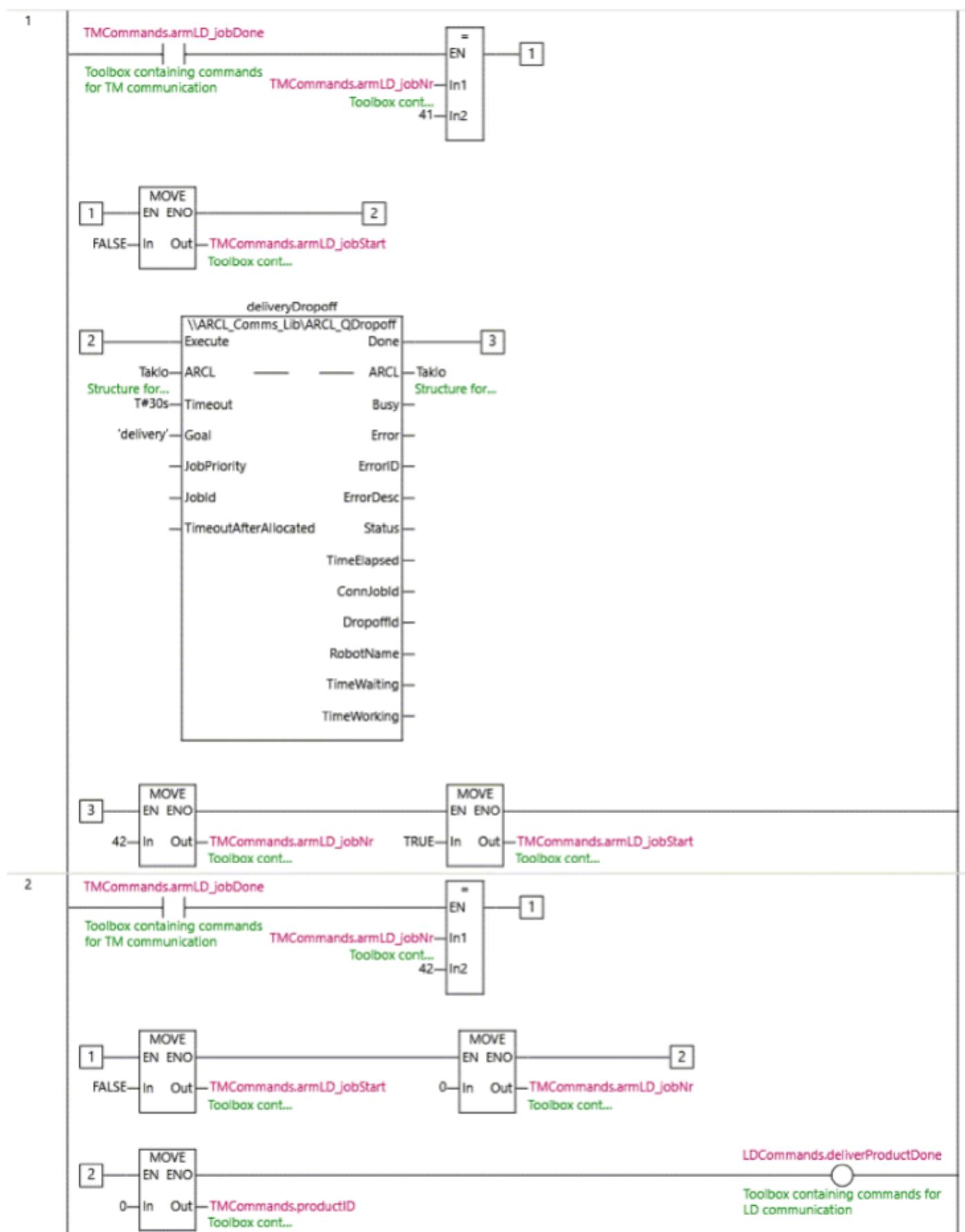
1-6-1-4.getFromKanban



1-6-1-4-5.getFromCutter



1-6-1-4-6.deliverProduct



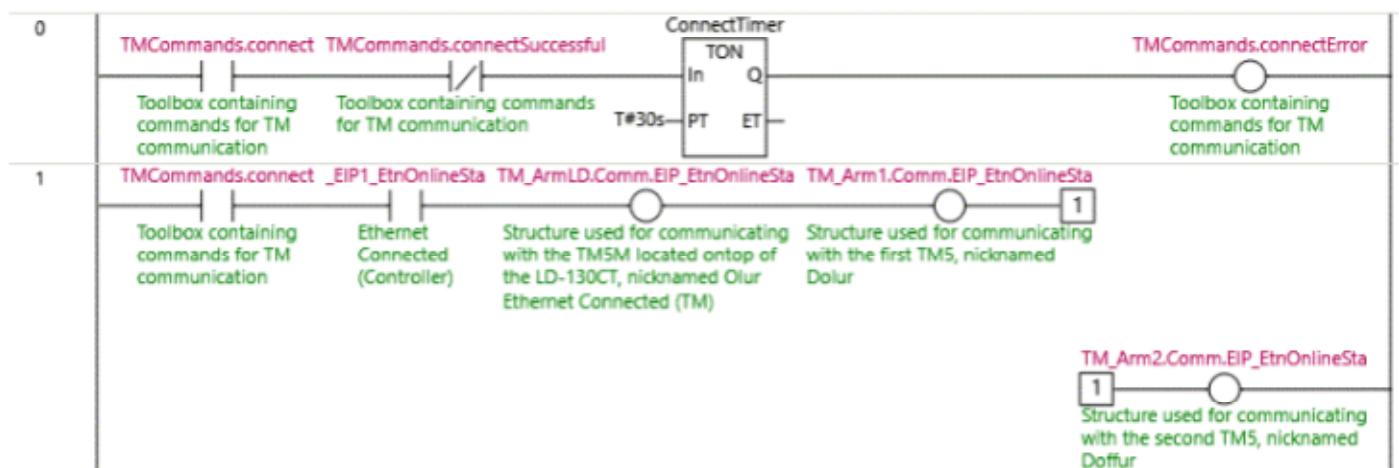
1-6-1-5.commsTM**1-6-1-5-1.Variables**

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_ErrorIDtx	STRING[100]			False	False	
FB_TM_ArmLD	OEN\Robot\TM\TM_ModbusTCP			False	False	
TM_testconnect	BOOL			False	False	
FB_TMCControl	OEN\Robot\TM\TM_Control			False	False	
ArmLD_PlayPause	BOOL			False	False	
ArmLD_Stop	BOOL			False	False	
ArmLD_ToggleAuto	BOOL			False	False	
ArmLD_Mode	OEN\nRobot\TM\cMode			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_TMCControl_ArmLD	OEN\Robot\TM\TM_Control			False	False	
FB_TMCControl_Arm1	OEN\Robot\TM\TM_Control			False	False	
Arm1_PlayPause	BOOL			False	False	
Arm1_Stop	BOOL			False	False	
Arm1_ToggleAuto	BOOL			False	False	
Arm1_Mode	OEN\nRobot\TM\cMode			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\T M_ModbusTCP			False	False	
Arm2_PlayPause	BOOL			False	False	
Arm2_Stop	BOOL			False	False	
Arm2_ToggleAuto	BOOL			False	False	
Arm2_Mode	OEN\nRobot\nTM\neMode			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\nsRobot				False	
TMCommands	ST_TMTools				False	
TM_Arm1	OEN\nRobot\nTM\nsRobot				False	
TM_Arm2	OEN\nRobot\nTM\nsRobot				False	

1-6-1-5-2.ModbusTCP_Setup

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

```

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 except UserINT and UserFloat
25
26 ArmLD_AttemptConnect := TRUE;
27
28
  
```

3 TM_Arm1.Comm.EIP_EtnOnlineSta

Structure used for communicating with the first TM5, nicknamed Dolur

```

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.JointTorqu
e:=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 except UserINT and UserFloat
25
26 Arm1_AttemptConnect := TRUE;
27
28

```

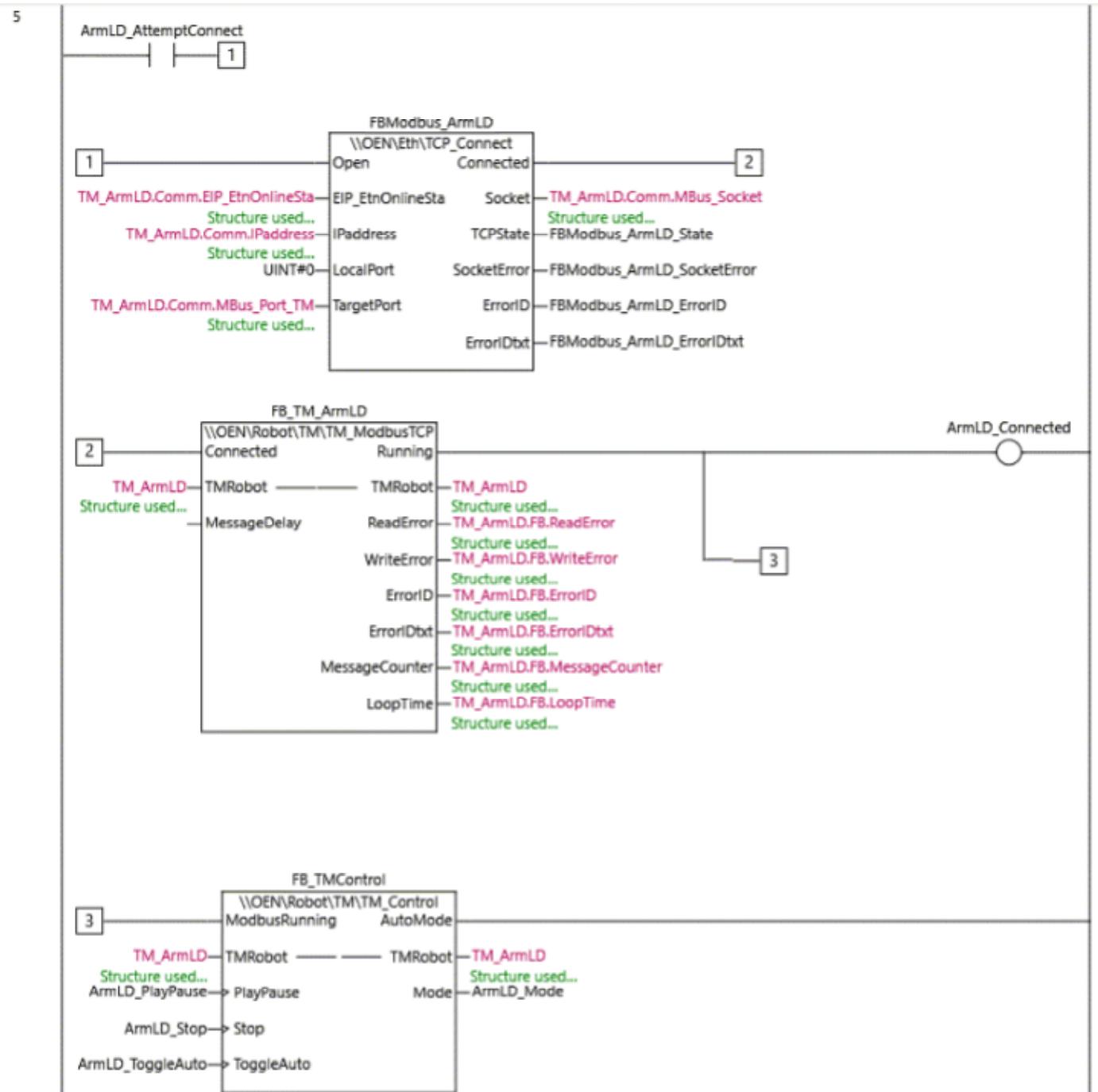
4 TM_Arm2.Comm.EIP_EtnOnlineSta

Structure used for communicating with the second TM5, nicknamed Doffur

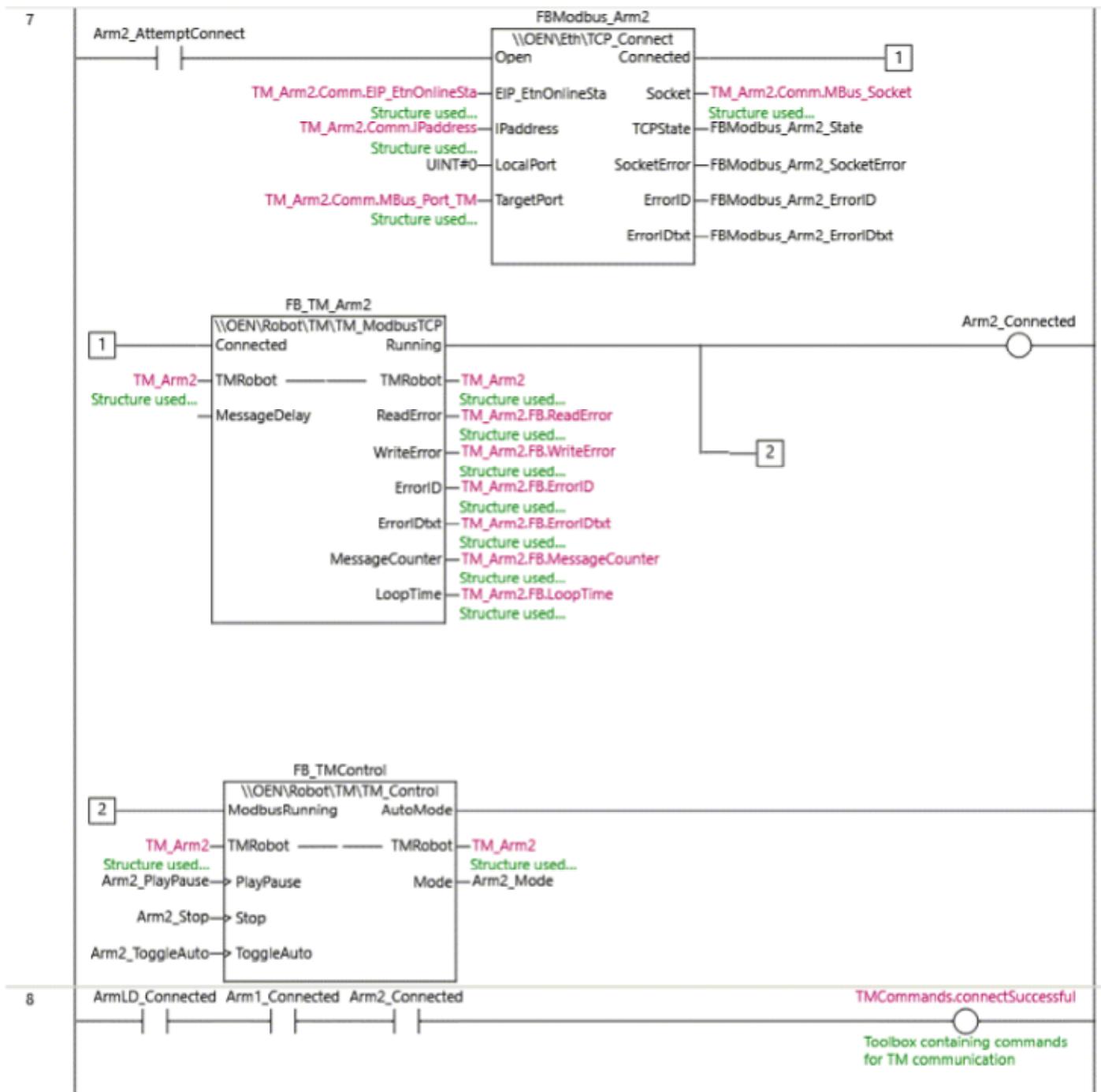
```

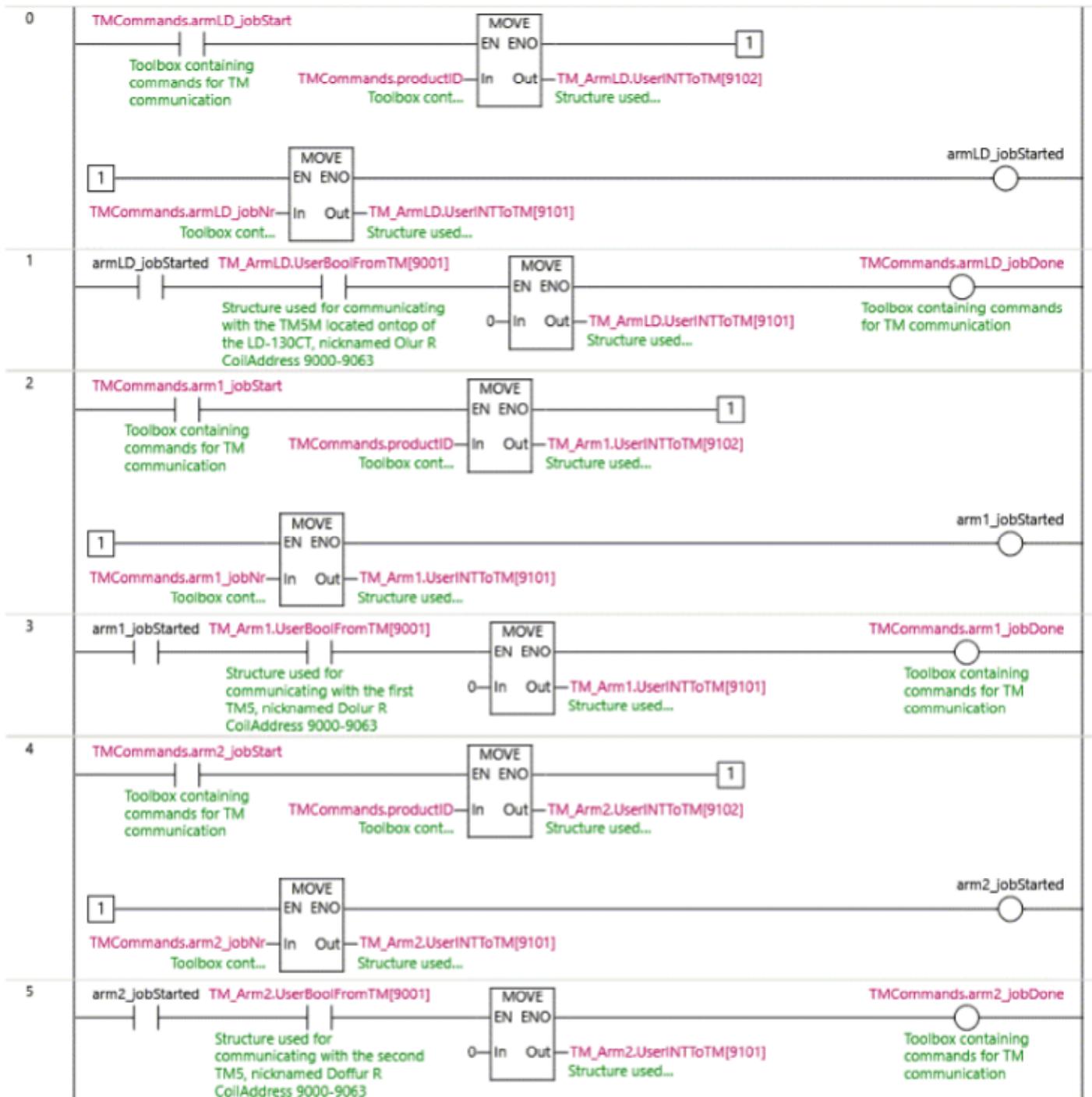
1 TM_Arm2.Comm.MBus_Port_TM:=UINT#502;           //Robot TCP-port for
Modbus communication
2 TM_Arm2.Comm.IPAddress:='192.168.250.152';
3
4 //Activate Modbus Reading
5 TM_Arm2.DataSelect.Coordinates_Reg1B59thru1B94:=TRUE; //Activate reading
of coordinates
6 TM_Arm2.DataSelect.Others_Reg1CACthru1CB7:=TRUE; // 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 except UserINT and UserFloat
25
26 Arm2_AttemptConnect := TRUE;
27
28

```







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

```

1  IF state = 0 THEN
2      IF _EIP_EtnOnlineSta THEN
3          iSktCreate(Execute := FALSE, SrcUdpPort := 6000, Socket => Socket);
4          state := 10;
5      END_IF;
6  END_IF;
7
8  IF state = 10 THEN
9      iSktCreate(Execute := TRUE, SrcUdpPort := 6000, Socket => Socket);
10     IF NOT iSktCreate.Busy THEN
11         IF iSktCreate.Done THEN
12             Socket.DstAddr.IpAddr := '192.168.250.200';
13             Socket.DstAddr.PortNo := 6000;
14             state := 20;
15         END_IF;
16
17         IF iSktCreate.Error THEN
18             state := 9000;
19         END_IF;
20     END_IF;
21 END_IF;
22
23 IF state = 20 THEN
24     iSktClearBuf(Execute := FALSE, Socket := Socket);
25     state := 30;
26 END_IF;
27
28 IF state = 30 THEN
29     iSktClearBuf(Execute := TRUE, Socket := Socket);
30     IF NOT iSktClearBuf.Busy THEN
31         state := 40;
32     END_IF;
33 END_IF;
34
35 IF state = 40 THEN
36     iSktReceive(Execute := FALSE, Socket := Socket, Size := 2000, RcvDat := RcvPacket[0]);
37     state := 50;
38 END_IF;
39
40 IF state = 50 THEN
41     iSktReceive(Execute := TRUE, Socket := Socket, TimeOut := 20, Size := 2000, RcvDat := RcvPacket[0]);
42     IF NOT iSktReceive.Busy THEN
43         IF iSktReceive.Done THEN
44             AryByteTo(In := RcvPacket[0], Size := iSktReceive.RcvSize, OutVal := RcvStr);
45             IF RcvStr = 'Hello World' THEN
46                 Success := TRUE;
47             ELSE
48                 Success := FALSE;
49             END_IF;
50             state := 60;
51         END_IF;
52
53         IF iSktReceive.Error THEN
54             state := 9000;
55         END_IF;
56     END_IF;
57 END_IF;

```

```
58
59  IF state = 60 THEN
60      SendStr := 'Hello World';
61      SendSize := ToAryByte(ln := 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
9     //print is yet to be retrieved.
10    IF (statusArray[index].ready = 'True') AND (statusArray[index].finished = 'False') THEN
11        // This printer is ready to use, choose it as printer to assign work
12        printerNr := index;
13    END_IF;
14 END_FOR;
15
16 //Return either available printer or -1.
17 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							
ProcessGcodeName	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(ln1:=gcodeName, ln2:='_');
15 IF LEFT(ln1:=gcodeName, L:=1)='P' AND RIGHT(ln1:=gcodeName, L:=9)='pcs.gcode' AND NOT(cursor=0) AND
NOT(FIND(ln1:=RIGHT(ln1:=gcodeName, L:=LEN(gcodeName)-cursor), ln2:='_'))=0) AND FIND(ln1:=RIGHT
(ln1:=RIGHT(ln1:=gcodeName, L:=LEN(gcodeName)-cursor), L:=LEN(RIGHT(ln1:=gcodeName, L:=LEN(gcodeName)-
cursor))-FIND(ln1:=RIGHT(ln1:=gcodeName, L:=LEN(gcodeName)-cursor), ln2:='_')), ln2:='_'))=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(ln1:=gcodeName, ln2:='_');
21 //Remove P at the start and everything after (and including) the underscore.
22 tempStr := MID(ln1:=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(ln1:=tempStr)) AND (tempStr = ToUCase(ln1:=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(ln1:=tempStr, ln2:=';'))=0) THEN
27     productID := DELETE(ln1:=tempStr, L:=1, P:=FIND(ln1:=tempStr, ln2:=';'));
28 ELSEIF NOT(FIND(ln1:=tempStr, ln2:='.'))=0) THEN
29     productID := DELETE(ln1:=tempStr, L:=1, P:=FIND(ln1:=tempStr, ln2:='.'));
30 ELSE
31     productID := tempStr;
32 END_IF;
33 ELSE
34     productID := '-1';
35 END_IF;
36
37 tempStr := RIGHT(ln1:=gcodeName, L:=(LEN(gcodeName)-cursor));
//Extracts everything after the underscore
38 cursor := FIND(ln1:=tempStr, ln2:='_'); //Find next
underscore
39 tempStr := MID(ln1:=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(ln1:=tempStr)) AND (tempStr = ToUCase(ln1:=tempStr)) THEN
41     IF NOT(FIND(ln1:=tempStr, ln2:=';'))=0) THEN
42         number := DELETE(ln1:=tempStr, L:=1, P:=FIND(ln1:=tempStr, ln2:=';'));
43     ELSEIF NOT(FIND(ln1:=tempStr, ln2:='.'))=0) THEN
44         number := DELETE(ln1:=tempStr, L:=1, P:=FIND(ln1:=tempStr, ln2:='.'));
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(ln1:=productID, ln2:=';', ln3:=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	NJ	Offset Byte	Offset Bit	Comment
ST_OrderDetails	STRUCT	NJ			
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				
ST_ProductStatus	STRUCT	NJ	Offset Byte	Offset Bit	
number	INT				
printing	INT				
buffer	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
customerOrderList	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	<pre>[separator := 'sep', finalFileName := '\$R\$LtestPlate.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</pre>		False	True	Do not publish	Array holding default .DXF cutting information for each product

		<pre> arator := 'sep=', finalFileName := 'e := '\$R\$LtestPlate.dxf', templateName := 'NTNU_Key Chain_Temp lace_v2.dxf', logoFileName := 'e := 'ntnu_logo_s vart.png', xInsertPoint Logo := '1.505', yInsertPoint Logo := '18.7732', xPixelSize := '260', yPixelSize := '348', xSizeInMm := '22', ySizeInMm := '29.5', rotationLogo := '0', textToInsert := 'H C F', textStyle := 'NTNU- DIN', xInsertPoint Text := '12.5', yInsertPoint Text := '14', rotationText := '0', alignmentText := '2', textWidth := '25', textHeight := '2.5', strNameLen := '8')]</pre>				
cutterStatus	ST_CutterStatus		False	False	Do not publish	Structure containing the status of the cutters
printerCommand	ST_3DPrinterCommand	(separator := 'sep=\$R\$LI P_Address;C ommand', IP := 'Argument\$R\$L192.168. 250.200', argument := 'print', command := '%api/files/loc al/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_1pcsgcode']		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
productStatus	ARRAY[1..2] OF ST_ProductStatus			False	False	Do not publish	Structure holding the current status for number of a procuts in store, being printed and its buffer
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\nTMsRobot			False	False	Do not publish	Structure used for communicating with the TM5M located ontop of the LD-130CT, nicknamed Olur
TM_Arm1	OEN\nRobot\nTMsRobot			False	False	Do not publish	Structure used for communicating with the first TM5, nicknamed Dolur

TM_Arm2	OEN\nRobot\nT M\sRobot			False	False	Do not publish	Structure used for communicating with the second TM5, nicknamed Doffur
---------	---------------------------	--	--	-------	-------	----------------	---