TCP (AHP) Integration Guide

Send orders to the cutting machines and receive the results over TCP with the AHP telegram protocol

Interface	TCP (AHP)
Document version	1.1
Date	October 2026

Contents

1. Introduction 4
1.1 What the TCP interface is 4
1.2 How it works: two connections 4
1.3 What you need from the Link administrator 5
1.4 Conventions in this guide 5
2. Quick start 6
3. The protocol 7
3.1 The frame 7
3.2 Sequences, duplicates and keep-alive 7
3.3 Acknowledgements 7
3.4 Timing and retries of the Link 8
4. Telegrams you send 9
4.1 Type 10: order line 9
4.2 Type 11: cancel 10
4.3 Type 00: keep-alive 10
5. Telegrams the Link sends 11
5.1 Examples 11
5.2 Delivery rules 12
6. The T dialect (TK / Jungheinrich) 13
7. Errors and behaviour on bad input 14
7.1 Error codes in telegram 78 14
7.2 Malformed telegrams 14
8. Debugging and troubleshooting 15
8.1 A method: follow the order step by step 15
8.2 Where the Link leaves its traces 15
8.3 My line does not start 16
8.4 Common problems with this interface 17
8.5 Tracing the traffic 17
8.6 What to send to LOIMEX support 18
9. Testing without a machine 19
9.1 Set up the test machine 19
9.2 Useful simulator options 20
10. Going to production 21
Appendix A. Example client (Python) 22
Appendix B. Reference of all telegrams 25

1. Introduction

1.1 What the TCP interface is

The **TCP interface** speaks the **AHP** telegram protocol, the one used between automatic warehouse systems and machines. Your system and the Link exchange short **fixed-width telegrams** over two TCP connections. It is the right choice if your system already speaks AHP (for example a Jungheinrich / TK warehouse), or if you prefer a plain socket to HTTP or SQL.

- **Your telegrams to the Link:** an order line (`10`), a cancellation (`11`) and keep-alives (`00`). Each one is acknowledged at once.
- **The Link's telegrams to you:** the line was accepted (`60`), done (`70`), canceled (`71`), rejected (`78`) and the state of the machines (`80`). You acknowledge each one.
- **Two dialects:** the **Link dialect** (`L`), described in this guide, and the **T dialect** of the TK / Jungheinrich warehouse module (chapter 6).

1.2 How it works: two connections

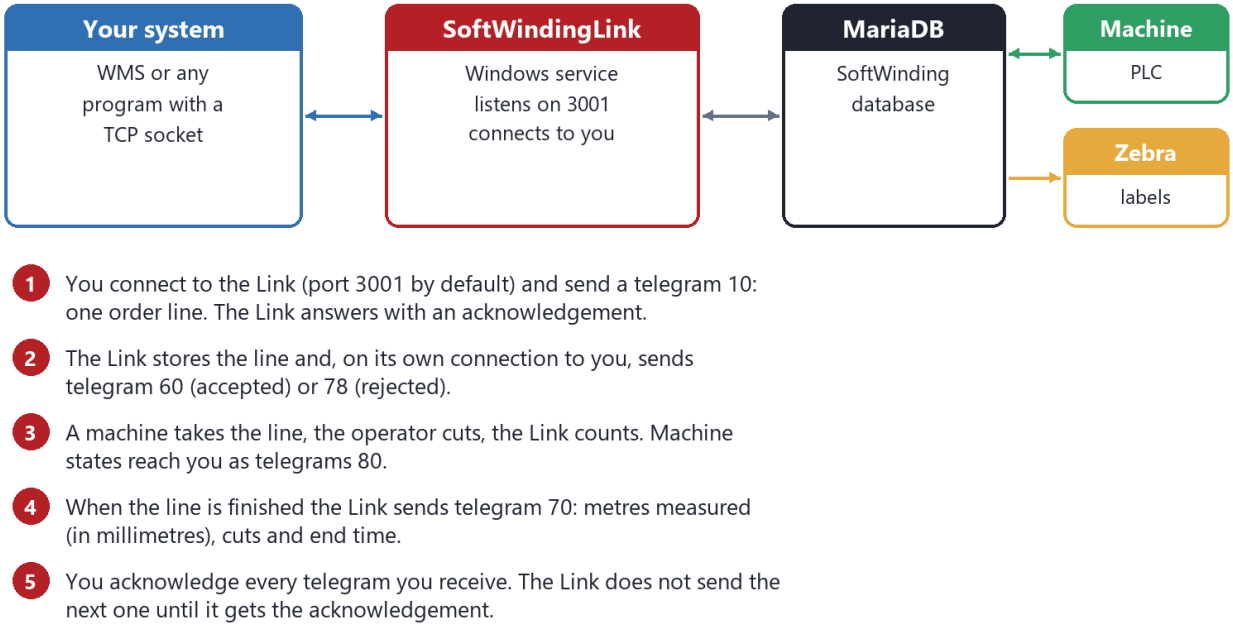


Figure 1. The path of an order through the TCP interface.

Table 1. The two channels. Each connection is opened by the side that sends.

Channel	Who connects	To where	What travels
Sending channel	You	The Link, port `[Ahp] ListenPort` (3001)	Your telegrams `10`, `11`, `00` and the Link's acknowledgements of them.
Receiving channel	The Link	Your host and port registered in the client (`TcpHost`, `TcpPort`)	The Link's telegrams `60`, `70`, `71`, `78`, `80` and your acknowledgements.

> **Warning.** **You must listen before you send.** The Link answers your order on its own connection to your receiving port. Start your listener first; if it is not there, the Link keeps the telegrams and retries until it can deliver them (section 5.2).

1.3 What you need from the Link administrator

Table 2. Data to ask the administrator for.

You need	Example	Where it is set
The Link host and port	`link-server:3001`	Server name or IP and `[Ahp] ListenPort` in `SoftWindingLink.ini`. Switched **off by default**: `[Ahp] Enabled=1`, then restart the service.
A client with the TCP channel	`WMS`	Administration window, **Clients** tab, or `SoftWindingLinkSvc.exe --newclient=WMS --channels=TCP --dialect=L --host=<your IP> --port=3200`.
Your receiving port	`3200`	The `Port` of your client: the Link connects to `TcpHost:TcpPort`.
Your IP address	`10.0.0.20`	The Link **identifies you by your IP address** (`TcpHost`, or `AllowedIPs` if filled). Only one TCP client per IP address.
Acknowledgement format	`S` or `C`	`AckFormat` of your client (section 3.3). `S` for new clients.
Machine identifiers	`M1`, `M2`	Administration window, **Machines** tab. At most **20 characters** in AHP.

Allow the connections in your firewalls: your system to the Link port, and the Link to your receiving port.

1.4 Conventions in this guide

- Telegrams are shown as they travel. `<STX>` and `<ETX>` are the bytes 0x02 and 0x03. In long telegrams the padding spaces are shown as `·` and the line is wrapped; on the wire it is a single line.
- The examples were captured on a test system with two simulated machines (`M1`, `M2`) and the client `WMS`. The Python code is in `docs\examples\ahp\ahp_client.py` and the `ahpclient.exe` tool is in the installation folder ("Simulators and diagnostic tools" component).
- Lengths in AHP are **integer millimetres**: 12.5 m is `0000012500`.

2. Quick start

A complete cycle with the example client (`ahp_client.py`, Python 3, no packages). It listens on your receiving port, sends a keep-alive, one order, one invalid order, and waits for the result.

```
set LINK_HOST=link-server
set LINK_PORT=3001
set LISTEN_PORT=3200
python ahp_client.py
```

The first part: the keep-alive and the order, with the acknowledgement of each (`send` = you to the Link, `recv` = the Link to you):

```
send  <STX>0000100<ETX>
recv  <STX>00001<ETX>
send  <STX>0000210···························WMS-1001·····················00010M
      1···············H07V-K-1.5-BK··············R-77881···········000001250
      0000020000H07V-K·1.5·mm2·black···············ACME·Cables·Ltd···········
      ····························<ETX>
recv  <STX>00002<ETX>

-- the Link connects to you:
recv  <STX>0023560WMS-1001···················00010M1···············<ETX>
send  <STX>00235<ETX>
```

Telegram `10` carries the order line. The Link answered `<STX>00002<ETX>` (acknowledgement of sequence 2) and, a moment later, **its own** telegram `60` (line accepted) on the receiving channel, which you acknowledge with `<STX>00235<ETX>`.

When the line is finished the Link sends telegram `70`:

```
recv  <STX>0024070WMS-1001···················00010M1···············R-77881····
      ·········00000002500000002202610281320 30<ETX>
```

Read it with the table of section 5: order `WMS-1001`, line 10, machine `M1`, reel `R-77881`, **25000 mm** measured (2 cuts of 12.5 m), 2 cuts, ended `2026-10-28 13:20:30`.

3. The protocol

3.1 The frame

```
<STX> SSSSS TT data... <ETX>
 0x02   5     2            0x03
```

Table 3. Parts of a telegram.

Part	Meaning
`<STX>` / `<ETX>`	Start (0x02) and end (0x03) of the telegram. There is **no length field**: read until `<ETX>`.
`SSSSS`	Sequence number: 5 digits with leading zeros, 00001 to 99999, then back to 00001.
`TT`	Telegram type: 2 digits (`10`, `11`, `00`, `60`, `70`, ...).
data	Fixed-width fields. **CHAR(n)**: left aligned, padded with spaces on the right. **NUM(n)**: right aligned, padded with zeros on the left; a negative number has its `-` in the first position.

Text is Latin-1 (one byte per character). Identifiers allow printable ASCII without spaces; a `ProductID` longer than 30 characters cannot be sent by AHP (use REST or the database).

3.2 Sequences, duplicates and keep-alive

- **Your sequence** numbers your own telegrams. Use any increasing counter; the acknowledgement repeats it. A telegram with the **same sequence as the previous one** is acknowledged but **not processed again**: this is how a retry is made safe.
- **The Link's sequence** is consecutive per client (00001 to 99999) and restarts at 1 when the Link process restarts. Do not assume it never goes back.
- **Idempotency by key:** sending the same order line again with a **new** sequence is also safe: the line (client, order, item) exists once. The Link answers `60` again.
- **Keep-alive `00`:** acknowledged and **reflected** on the receiving channel with the same sequence. The Link closes a receiving connection that is idle for 10 minutes: send a keep-alive from time to time.

```
send  <STX>0000100<ETX>          <- your keep-alive
recv  <STX>00001<ETX>          <- its acknowledgement
recv  <STX>0000100<ETX>          <- reflected on the receiving channel
send  <STX>00001<ETX>          <- you acknowledge it
```

3.3 Acknowledgements

Every telegram is acknowledged on the connection where it arrived. There are two formats:

Table 4. Acknowledgement formats.

AckFormat	Format	Length	Used by
`S`	`<STX> SSSSS <ETX>`	7 bytes	The AHP specification and the new clients (default).
`C`	`<STX> SSSSS TT <ETX>`	9 bytes	The current LOIMEX / TK module, which repeats the type.

The Link **accepts both** when it receives an acknowledgement from you, and **sends** the one set in `AckFormat` of your client. It checks that the acknowledgement **repeats the sequence** of the telegram. Answer in the same format the Link uses to you.

3.4 Timing and retries of the Link

Table 5. Link behaviour.

Situation	What the Link does
You do not acknowledge a telegram within 1.5 s	It sends it again, up to 3 attempts, each one on a **new connection**.
The 3 attempts fail	It waits 8 seconds and tries again, **without skipping any event**.
Your receiving port is closed	It logs a warning (`AHP > ... connect ... refused`) and keeps trying. Telegrams wait in order.
You connect from an address that is not yours	The connection is refused and logged: `AHP connection refused: 127.0.0.2 is not an authorised TCP client`.

4. Telegrams you send

4.1 Type 10: order line

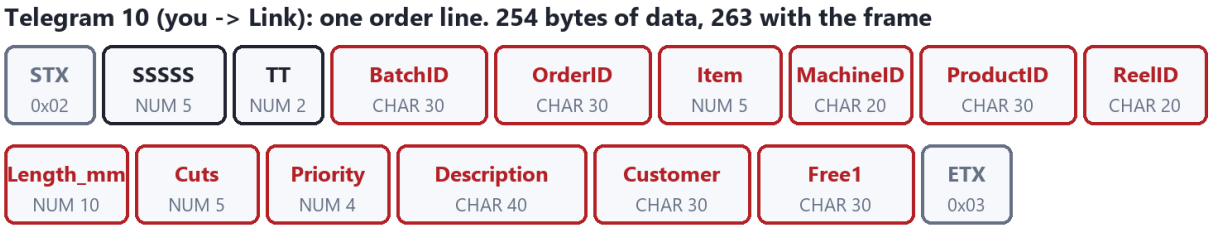


Figure 2. Structure of telegram 10.

Table 6. Fields of telegram 10.

Field	Type	Meaning
`BatchID`	CHAR(30)	Work package. Blank = the default batch of your client (`<ClientID>_<date>`).
`OrderID`	CHAR(30)	Your order number. Required.
`Item`	NUM(5)	Line number (1 to 99999). With the order it identifies the line.
`MachineID`	CHAR(20)	The machine. Blank = any machine that accepts general lines.
`ProductID`	CHAR(30)	Product code. Required.
`ReelID`	CHAR(20)	Reel (label and traceability).
`Length_mm`	NUM(10)	Length of **one** cut in **millimetres**. 12.5 m = `0000012500`. Must be > 0.
`Cuts`	NUM(5)	Number of cuts (1 to 99999).
`Priority`	NUM(4)	Lower goes first. 0 by default.
`Description`	CHAR(40)	Text for the label and the machine screen.
`Customer`	CHAR(30)	Customer name.
`Free1`	CHAR(30)	A free field for the label template.

One line is one telegram `10`; an order with several lines is several telegrams with the same `OrderID`. A real telegram (padding shown as `·`):

```
<STX>0000210···························WMS-1001·····················00010M1·········
·········H07V-K-1.5-BK···············R-77881·············0000012500000020000H07V-K·1.5
·mm2·black·····················ACME·Cables·Ltd·····································
··<ETX>
```

The Link answers with the acknowledgement, and later with `60` (accepted) or `78` (rejected). If you send a telegram `10` again with the same key you get `60` again and the line is not duplicated:

```
send  <STX>0000210···························WMS-1001····················· ...
recv  <STX>00002<ETX>
send  <STX>0000310···························WMS-1001····················· ...
recv  <STX>00003<ETX>
recv  <STX>0023860WMS-1001                    00010M1                 <ETX>
send  <STX>00238<ETX>
```

The first telegram repeats sequence 2 (a retry): it is acknowledged and **not processed again**. The second has a new sequence, 3, but the same key: it is acknowledged, **no new line is created** and the Link sends one `60`.

4.2 Type 11: cancel

Table 7. Fields of telegram 11 (44 bytes, 74 with the machine).

Field	Type	Meaning
`OrderID`	CHAR(30)	The order to cancel. **Blank**: cancel what a machine has on its screen.
`Item`	NUM(5)	The line. **0** = the whole order.
`MachineID`	CHAR(30)	Optional, at position 36 of the data. With a blank `OrderID` it names the machine whose screen is cleared.

Cancel the whole order `WMS-2001` (still pending): the acknowledgement, then a `71` (canceled):

```
send  <STX>0024771WMS-2001····················00010<ETX>
recv  <STX>00247<ETX>
...
recv  <STX>0024771WMS-2001·····················00010<ETX>
```

Rules are those of the other interfaces: a pending line is deleted at once (`71`); a line that has not started to measure is released and deleted; a line that is **being measured** is not canceled (`78`, code 4011); a finished line cannot be canceled (`78`, 4011). Clear what a machine shows (blank order and machine):

```
send  <STX>0000911···························00000M1·····························<ETX>
recv  <STX>00009<ETX>
recv  <STX>0024871WMS-2000·····················00010<ETX>          <- 71: the line that was on the screen
```

If the machine is **running** the request is refused with a `78` and code 4014. This telegram does not exist in the T dialect.

4.3 Type 00: keep-alive

No data. See section 3.2.

5. Telegrams the Link sends

You receive them on your receiving channel and **must acknowledge each one**.

Table 8. Telegrams from the Link.

Type	Meaning	Fields (width)	Frame
`60`	Line accepted	`OrderID` CHAR(30), `Item` NUM(5), `MachineID` CHAR(20)	64
`70`	Line done	`OrderID` CHAR(30), `Item` NUM(5), `MachineID` CHAR(20), `ReelID` CHAR(20), `Measured_mm` NUM(12), `Cuts` NUM(5), `EndTime` CHAR(14) `yyyymmddhhnnss`	115
`71`	Line canceled	`OrderID` CHAR(30), `Item` NUM(5)	44
`78`	Order or line rejected / error	`OrderID` CHAR(30), `Item` NUM(5), `Code` NUM(4), `Text` CHAR(40)	88
`80`	Machine status	`MachineID` CHAR(20), `State` CHAR(1), `Counter_mm` NUM(12), `CutNr` NUM(5)	47
`00`	Keep-alive (reflected)	-	10

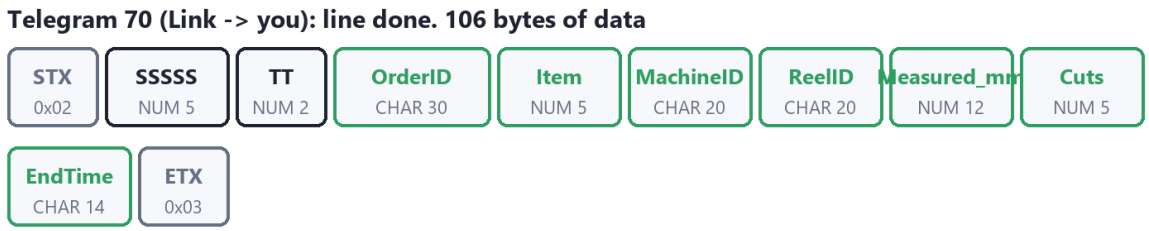


Figure 3. Structure of telegram 70.

5.1 Examples

5.1.1 60 and 80: the line was accepted and the machine prepared it

```
<STX>0023560WMS-1001····················00010M1···············<ETX>
<STX>0023680M1···············P00000000000000001<ETX>
<STX>0023780M1···············P00000001250000000<ETX>
```

`80` carries the machine: `M1`, state `P` (prepared), counter 12 500 mm (12.5 m), cut number 0. States are `O` offline, `I` free, `P` prepared, `R` running, `S` waiting for confirmation, `E` error, `D` disabled. They are sent at most every 5 seconds per machine.

5.1.2 70: the line is done

```
<STX>0024070WMS-1001····················00010M1···············R-77881···········0
00000025000000022026102813203O<ETX>
```

5.1.3 78: a rejection

```
<STX>0023978WMS-E1····················000104001unknown·machine·····················<ETX>
<STX>0025178··························000004004invalid·field·························<ETX>
```

The first: order `WMS-E1`, line 10, code **4001** `unknown machine`. The second, for a telegram `10` that was too short: blank order, code **4004** `invalid field`.

5.2 Delivery rules

- Telegrams arrive **in order** and **at least once**. The Link does not send the next telegram until you acknowledge the previous one, and **it does not advance your cursor until it receives the acknowledgement**.
- If your receiver was down, the Link delivers everything you missed when you come back, in the original order. A **new client** starts with its cursor at zero: the first connection can bring a long backlog of `80` machine-status telegrams. Acknowledge them (you can ignore their content); the administrator can move the cursor to the present (`sw_link_clients.LastEventID`) if the backlog is not wanted.
- You receive the telegrams of **your** orders, and the machine status `80` of the machines.
- The telegram `70` is the one that matters for your records. The `label.printed` and waste events do not exist in AHP.

6. The T dialect (TK / Jungheinrich)

For integrations with the TK / Jungheinrich warehouse module the Link understands the telegrams of that specification **unchanged**, with `--dialect=T` and `AckFormat C`. This is not needed for new integrations: use the Link dialect.

Table 9. Telegrams of the T dialect.

Type	Direction	What the Link does
`00`	both	Keep-alive.
`32`	WMS -> Link	Cut request. It becomes one line: `DeliveryID` = TaskID, product = cut type (`ROPES`, `CABLES`, `MANUAL`), length = QPick/1000, one cut, any machine. Batch `LMX_dd-mm-yyyy`. Nothing is deleted.
`92`	Link -> WMS	Cut done (only the TaskID).
`05`	WMS -> Link	Cancel. The Link answers `65`.
`65`	Link -> WMS	Canceled.
`58`	Link -> WMS	Error, with the TK codes 1001 to 3000.
`03` / `63`	both	Move platform: **not implemented**. The telegram is acknowledged and logged; no `63` is sent.

Ask LOIMEX for the specification of the warehouse module if you integrate this dialect; it is outside the scope of this guide.

7. Errors and behaviour on bad input

7.1 Error codes in telegram 78

Table 10. Error codes.

Code	Text	What to do
4001	unknown machine	`MachineID` does not exist. Maximum 20 characters in AHP.
4002	machine disabled or offline	Retry later or use another machine.
4003	duplicate or unchangeable order	The line was already taken or finished and differs.
4004	invalid field	Length not > 0, cuts < 1, empty product or order, or a telegram that is too short to be an order.
4010	invalid character in an identifier	A space or forbidden character in an identifier.
4011	the line cannot be canceled	It is being measured or already done.
4012	machine not allowed / client not allowed	Ask the administrator.
4014	the machine is running	Clear-screen refused: wait until it stops.

7.2 Malformed telegrams

What the Link does with input it cannot use (captured):

Table 11. Behaviour on bad input.

You send	The Link answers
A telegram `10` that is too short	Acknowledges it (`<STX>00900<ETX>`) and sends a `78` with blank order and code 4004 `invalid field`.
An unknown type (`99`)	Acknowledges it and logs a warning: `AHP telegram type 99 not supported`.
A telegram without `<ETX>`	Nothing: the Link keeps waiting for the end of the telegram. Always terminate with `<ETX>`.
A connection from another address	Refused and logged.

8. Debugging and troubleshooting

8.1 A method: follow the order step by step

When "nothing happens", do not guess: follow the order along its path, from your software to the machine, and find the first step where it stops.

Table 12. The five steps of a diagnosis.

Step	Question	Where to look
1	**Did the Link read my telegram?** Every telegram is acknowledged on the same connection. A telegram `78` reports a rejected order.	The acknowledgement and the `AHP <` / `AHP >` lines of the Link log.
2	**Is the order stored?**	Administration window, **Orders** tab (filter by order number), or the view `sw_link_v_orders` in the database.
3	**Is a machine working on it?**	Administration window, **Machines** tab: state and current order. In the database: the view `sw_link_v_machines`.
4	**What did the Link do?**	Administration window, **Log** and **Events** tabs; the daily log file `logs\link_YYYYMMDD.log`.
5	**What did the machine answer?**	The log lines tagged with the machine, for example `[M1] ...`, and the machine state text.

8.2 Where the Link leaves its traces

Table 13. Diagnostic tools.

Place	What it shows
Administration window	**Status**: engine, database and licence. **Machines**: state, counter and current line of each machine, with a connection test. **Orders**: every line with its status, searchable. **Clients**: your client, its channels, allowed IPs and machines. **Events**: the events your system receives. **Log**: the Link log with a level filter and a button that opens the log folder.
Log file `logs\link_YYYYMMDD.log`	One line per fact: starts, connections, every request of the channel, every order accepted, every line sent to the PLC, every cut confirmed, warnings and errors. One file per day, kept 30 days.
Table `sw_link_log`	The engine part of the log, queryable with SQL: `SELECT * FROM sw_link_log ORDER BY LogID DESC LIMIT 50`.
Events (`sw_link_events`, API, files, messages)	The business-level history: `line.done`, `line.error`, `line.canceled`, `label.printed`, `machine.status`, `line.waste`.
Views `sw_link_v_orders`, `_cuts`, `_waste`, `_machines`	The current state of lines, cuts made, wastes and machines, as plain SQL.
`SoftWindingLinkSvc.exe --console`	Runs the engine in a window instead of as a service, to watch it live (stop with Enter). Combine with `--ini=<path>` to use a test configuration.

Typical engine lines for an order that goes well (excerpt of the log of the test system):

```
12:46:16 INF REST order SO-100234 from ACME: 2 line(s) accepted
12:46:17 INF [M1] claimed ACME_20261028|SO-100234|H07V-K-1.5-BK|10
12:46:17 INF [M1] line ACME_20261028|SO-100234|H07V-K-1.5-BK|10 sent to the PLC:
              H07V-K-1.5-BK x 3 cuts of 12.500 m (preselection 1250)
12:46:25 INF [M1] cut 1/3 confirmed: 12.500 m
```

```
12:46:38 INF [M1] line ACME_20261028|SO-100234|H07V-K-1.5-BK|10 DONE: 3 cuts, 37.500 m
```

8.3 My line does not start

The line is stored (`pending`) but no machine takes it, or the machine takes it and nothing is cut. Check, in this order:

Table 14. Why a line does not start.

What you see	Probable cause	What to do
Machine state `O` (offline) or `D` (disabled), or a heartbeat older than 30 s	The Link cannot reach the PLC, the machine is switched off in the Link, or **the Link service is not running**.	Check the service, the machine network and the **Machines** tab (use the connection test).
The line stays `pending`; the machine is free (`I`)	The line is for another machine, or has no `machineId` and the machine accepts only addressed lines, or an earlier line is still ahead of it (batch priority, `priority`, `item`).	Compare `machineId` with the machine; look at the **Orders** tab ordering.
The machine is `P` (`prepared ...`) and nothing moves	The line was sent to the PLC and the machine waits for the **operator to start it**. The Link does not start machines.	Ask the operator to start. This is normal.
The machine shows the order list on its screen and waits	The machine works in **screen mode**: the operator chooses the order on the machine panel.	The operator picks and starts the order on the screen.
The machine is `S` (`waiting for confirmation`)	The machine is set to external confirmation and waits for your system.	Confirm the cut (REST `confirm`, mailbox action `K`...).
The machine is `E`	Emergency stop, open guard or a PLC error.	The operator clears the cause; the Link continues.
The line is `error`	The PLC refused it or the data were invalid (see `code` and the `line.error` event).	Read the code in chapter 7 (REST guide) / the error table of this guide and the Link log.
The first line works but the next one never starts	The machine waits for the operator after each line (reset, next order) or a previous line is still `claimed`.	Look at the machine state and current order.

8.4 Common problems with this interface

Table 15. Common problems.

Symptom	Cause	Solution
`connect refused` when I connect to the Link	The AHP channel is off (`[Ahp] Enabled=0`), the port is wrong, the service is not running, or a firewall blocks it.	Check `[Ahp] Enabled`, `ListenPort` and the firewall. The log says `AHP listening on <ip>:<port>`.
I connect but nobody answers	The connection comes from an IP that is not your client's (`TcpHost` / `AllowedIPs`), or the telegram has no `<ETX>`.	Compare the source IP with the client. Look for `AHP connection refused` in the log.
I get the acknowledgement but never a `60`	The Link cannot connect to your receiving port: nothing is listening, the port differs from `TcpPort`, or a firewall blocks it.	Start the listener first. Check the log: `AHP > <ip>:<port> type 60: connect ... refused`.
The Link repeats the same telegram	You do not acknowledge in time (1.5 s) or the acknowledgement does not repeat the sequence.	Acknowledge immediately, with the same `SSSSS`, in the format of your `AckFormat`.
I receive hundreds of `80` telegrams	It is the backlog of a new client (cursor 0) or a receiver that was down.	Acknowledge them; ask the administrator to move your cursor if you do not want them.
`78` code 4001 for a machine that exists	The `MachineID` is longer than 20 characters, or has trailing garbage.	Check the field width: CHAR(20) in AHP.
Lengths are 1000 times too small	You sent metres instead of millimetres.	`Length_mm` is millimetres: 4 m = `0000004000`.
Strange characters	Encoding. The wire is Latin-1.	Encode texts as Latin-1, not UTF-8.
Two systems use the same IP	The Link identifies the client by IP: with two clients on one IP it uses the first and warns in the log.	One TCP client per IP address.

8.5 Tracing the traffic

Because AHP is a plain TCP stream you can watch it with any packet tool (**Wireshark**, filter `tcp.port == 3001`). The Link log also shows each telegram at debug level:

```
13:20:42 INF AHP WMS: order WMS-2001 item 10 accepted
13:20:43 DBG AHP > WMS event 185 sent as 60
13:20:45 DBG AHP < 127.0.0.1 type 11 seq 00008
13:20:52 WRN AHP telegram type 99 not supported (127.0.0.1)
13:20:56 WRN AHP connection refused: 127.0.0.2 is not an authorised TCP client
```

The tool **`ahpclient.exe`** (installation folder, "Simulators and diagnostic tools" component) sends telegrams and listens, to check your side or the Link without writing code:

```
ahpclient serve --port=3200 --log=received.txt [--ack=S|C] [--run=SECONDS]      listen and log
TYPE|SEQUENCE|DATA
ahpclient send  --port=3001 --seq=00001 --kind=keepalive
ahpclient send  --port=3001 --seq=00002 --kind=order --order=SO-1 --item=10 --machine=M1 --product=P1 -
-len=6500 --cuts=2
ahpclient send  --port=3001 --seq=00003 --kind=cancel --order=SO-1 --item=0
```

`send` prints the acknowledgement it received (`ACK <2>00002<3>`) and `serve` writes each received telegram as `TYPE|SEQUENCE|DATA`, with `.` for the padding spaces:

```
00|00077|
60|00252|TOOL-1.....................00010M2.................
80|00253|M2.................P0000000000000000
70|00254|TOOL-1.....................00010M2.....................................00000000300000001
20261028132517
```

8.6 What to send to LOIMEX support

- Link version (`GET /health` or the window title) and Windows version.
- The order number, the line (`item`) and the machine.
- The exact request and the **complete answer** (headers and body), or the row / file / telegram / message you sent.
- The log file of the day (`logs\link_YYYYMMDD.log`) and, if possible, an export of the **Events** tab.
- What you expected and what happened, with the time (local time of the server).

9. Testing without a machine

You do not need a real machine to develop. LOIMEX provides **simulators** that behave like a cutting machine and like a customer system. With them you can run every example in this guide at your desk, repeat a failing case as many times as you like and see events appear in the Link window.

Table 16. Simulators (installed in the Link folder with the "Simulators and diagnostic tools" component of the installer).

Program	What it is	Typical use
`SoftWindingLinkSim.exe`	Machine simulator (console). Speaks the PLC protocols of the Omron NX/CP (FINS over UDP, port 9600) and of PLCs without FINS (SFW over TCP, port 63000) and simulates a Zebra printer on TCP 9100. With `--auto` an automatic "operator" starts, measures and resets the counter by itself.	Unattended tests and scripts.
`SoftWindingLink_MachineSimulator.exe`	The same simulator with a **window**: counter, preselection and cut count, with buttons to start, stop, reset the counter, reach the length, cut and edit values. The right panel shows the order the Link writes to the machine.	You play the operator by hand.
`SoftWindingLink_ServerSimulator.exe`	Customer-system simulator with a window: sends orders by database, REST, TCP (AHP) and MQTT and shows the answers.	Compare your integration with a known good client.

9.1 Set up the test machine

1. **Start the machine simulator** in a command prompt: `SoftWindingLinkSim.exe --auto --speed=400 --start-delay=1500 --cut-delay=2000 --printer-port=9100`. It listens on `127.0.0.1`. `--speed` is the counter speed in counter units per second (hundredths of a metre: 400 is 4 m/s; default 2000), `--start-delay` the time before the automatic operator starts, `--cut-delay` the time the operator takes to cut and reset. Add `--noconsole` to run it in the background (otherwise it waits for commands typed in its console) and `--run=SECONDS` to make it stop by itself.
2. **Add the machine to the Link.** In the administration window, **Machines** tab, create `M1`: PLC type *Omron NX1 (FINS)*, address `127.0.0.1`, port `9600`, profile `OMRON_CP`, confirm mode *counter*, cut mode *cut by cut*, enabled. Use the connection test: it must answer. (The database equivalent is the `INSERT INTO sw_link_machines ...` shown in the Link manual, section 4.)
3. **Create your client** in the **Clients** tab (or `SoftWindingLinkSvc.exe --newclient=ACME --channels=TCP --dialect=L --host=<your IP> --port=<port you listen on>`) and keep the access data it shows.
4. **Check the channel** is enabled in the **Settings** tab and that the engine is running (Status tab).
5. **Run the examples.** The line goes `pending`, `claimed` and `done` in a few seconds; the simulated machine counts every cut and the simulated Zebra receives the labels.

> **Warning.** **Never point a test client at a Link that drives real machines**, unless the machine identifiers you use are reserved for tests: on a real machine an operator could start your test line. Use a separate test Link, or machines that exist only for testing.

9.2 Useful simulator options

Table 17. Machine simulator options.

Option	Effect
`--auto`	Automatic operator: starts, measures up to the preselection, confirms the cut and resets.
`--speed=N`, `--start-delay=MS`, `--cut-delay=MS`	Counter speed and operator delays. Slow values let you watch a line in progress.
`--fins-port=N`, `--sfw-port=N`	Ports. Use different ports to run several simulated machines on one PC (`M1` on 9600, `M2` on 9601).
`--printer-port=9100`, `--printer-file=labels.txt`	Simulated Zebra: the received ZPL is appended to the file.
`--waste=N`, `--wastes=K`	The cut is interrupted at N counter units and the operator resets the counter: a waste (a reset before the preselection). K = how many times (1 by default).
`--noconsole`, `--run=SECONDS`	Without them the simulator waits for typed commands in its console. Use `--noconsole` to run it in the background, and `--run` to make it stop after that many seconds.
`--listen=IP`	Address to listen on (default `127.0.0.1`).

Then run `ahp_client.py` or `ahpclient.exe` as shown above. The customer simulator `SoftWindingLink_ServerSimulator.exe` also has a TCP mode (Link dialect and T dialect): `--selftest=tcp` runs a complete cycle.

10. Going to production

Table 18. Production checklist.

Topic	Recommendation
Listener first	Keep your receiving server always up. Start it before the first send and restart it automatically.
Acknowledge at once	Acknowledge inside the 1.5 s window, before doing slow work: process the telegram afterwards.
Duplicates	Expect the same telegram twice (retries). Use the order and item as key; ignore a repeated sequence.
Persist what matters	Store `70` (measured, cuts, reel, time) as you receive it; the Link considers it delivered when you acknowledge.
Your sequence	Keep a counter per connection; after a restart any value is fine, but do not reuse the previous one for a different telegram.
Keep-alive	Send `00` every minute or so on the sending channel. If it is not acknowledged, reconnect.
Reconnect	Reconnect with a growing delay (1 s ... 30 s) after a failure. The Link will not lose events while you are away.
Units	Millimetres, integers. Convert on your side.
Firewall and IP	Fixed IP for your system; open both directions; one client per IP.
Monitoring	Alert if no `00` reflection arrives, or if `78` telegrams appear.

Appendix A. Example client (Python)

Telegram building and parsing, the sending channel and the receiving channel in one file (standard library only). The `main()` of the file runs the demo of chapter 2.

```
import os
import queue
import socket
import socketserver
import sys
import threading
import time


STX, ETX = b'\x02', b'\x03'



# ---------------------------------------------------------------------------------------------- telegram building
def char(value, width):
    """CHAR(n): left aligned, padded with spaces, cut at n. Latin-1 on the wire."""
    return str(value).encode('latin-1', 'replace')[:width].ljust(width)



def num(value, width):
    """NUM(n): right aligned, padded with zeros (a negative number keeps its sign in the first position)."""
    n = int(round(value))
    s = ('-' if n < 0 else '') + str(abs(n)).rjust(width - (1 if n < 0 else 0), '0')
    return s.encode('ascii')



def frame(seq, ttype, data=b''):
    """<STX> SSSSS TT data <ETX>"""
    return STX + ('%05d%02d' % (seq, ttype)).encode('ascii') + data + ETX



def order_telegram(seq, order_id, item, machine, product, length_m, cuts, reel='', batch='', priority=0, description='',
customer='', free1=''):
    """Type 10: one order line. Length in MILLIMETRES. 254 bytes of data."""
    data = (char(batch, 30) + char(order_id, 30) + num(item, 5) + char(machine, 20) + char(product, 30) + char(reel, 20) +
            num(length_m * 1000, 10) + num(cuts, 5) + num(priority, 4) + char(description, 40) + char(customer, 30) +
char(free1, 30))
    return frame(seq, 10, data)



def cancel_telegram(seq, order_id, item=0, machine=''):
    """Type 11: cancel (item 0 = the whole order). With order_id empty and a machine: clear what that machine shows."""
    data = char(order_id, 30) + num(item, 5) + (char(machine, 30) if machine else b'')
    return frame(seq, 11, data)



def keepalive_telegram(seq):
    return frame(seq, 0)



# ---------------------------------------------------------------------------------------------- telegram reading
def read_frame(sock, timeout=10):
    """Reads one frame (everything between STX and ETX) from a socket. Returns bytes without STX/ETX, or None on close."""
    sock.settimeout(timeout)
    buf = b''
    while True:
        ch = sock.recv(1)
        if not ch:
            return None
        if ch == STX:
            buf = b''
        elif ch == ETX:
            return buf
        else:
            buf += ch



def parse(body):
    """body = SSSSS TT data  ->  (seq, type, data text). An acknowledgement has no type: (seq, None, '')."""
    seq = int(body[:5])
    if len(body) == 5:
        return seq, None, ''
```

```
    return seq, int(body[5:7]), body[7:].decode('latin-1')



def describe(ttype, d):
    """Human-readable fields of the telegrams the Link sends."""
    t = lambda a, b: d[a:b].strip()
    if ttype == 60:
        return dict(order=t(0, 30), item=int(d[30:35]), machine=t(35, 55))
    if ttype == 70:
        return dict(order=t(0, 30), item=int(d[30:35]), machine=t(35, 55), reel=t(55, 75), measured_mm=int(d[75:87]),
cuts=int(d[87:92]), end=t(92, 106))
    if ttype == 71:
        return dict(order=t(0, 30), item=int(d[30:35]))
    if ttype == 78:
        return dict(order=t(0, 30), item=int(d[30:35]), code=int(d[35:39]), text=t(39, 79))
    if ttype == 80:
        return dict(machine=t(0, 20), state=d[20:21], counter_mm=int(d[21:33]), cut_nr=int(d[33:38]))
    return {}



# ---------------------------------------------------------------------------------------------- the two channels
class Link:
    """Sending channel: you connect to the Link, send a telegram and read the acknowledgement."""


    def __init__(self, host, port, log=None):
        self.host, self.port, self.seq, self.log = host, port, 0, log or (lambda *a: None)
        self.sock = socket.create_connection((host, port), timeout=10)


    def send(self, make):
        """make(seq) -> telegram bytes. Returns the acknowledgement (seq, type or None)."""
        self.seq = self.seq % 99999 + 1
        data = make(self.seq)
        self.sock.sendall(data)
        self.log('>', data)
        body = read_frame(self.sock)
        self.log('<', STX + (body or b'') + ETX)
        ack = parse(body)
        if ack[0] != self.seq:
            raise RuntimeError('acknowledgement for sequence %d, expected %d' % (ack[0], self.seq))
        return ack


    def close(self):
        self.sock.close()



class Receiver:
    """Receiving channel: the Link connects to LISTEN_PORT and sends telegrams; each one is acknowledged here.
    Received telegrams are put in .inbox as (seq, type, data text)."""


    def __init__(self, port, ack_type=False, host='0.0.0.0', log=None):
        self.inbox = queue.Queue()
        outer = self


        class Handler(socketserver.BaseRequestHandler):
            def handle(self):
                last = None
                while True:
                    try:
                        body = read_frame(self.request, timeout=900)
                    except (socket.timeout, OSError):
                        return
                    if body is None:
                        return
                    seq, ttype, data = parse(body)
                    # acknowledge: <STX> SSSSS <ETX>   (clients created with AckFormat C expect <STX> SSSSS TT <ETX>)
                    ack = STX + ('%05d' % seq).encode() + (('%02d' % ttype).encode() if ack_type and ttype is not None else
b'') + ETX
                    self.request.sendall(ack)
                    if log:
                        log('<', STX + body + ETX)
                        log('>', ack)
                    if seq != last and ttype is not None:         # a repeated sequence is a retry: acknowledge but do not
process again
                        outer.inbox.put((seq, ttype, data))
                    last = seq


        socketserver.ThreadingTCPServer.allow_reuse_address = True
        self.server = socketserver.ThreadingTCPServer((host, port), Handler)
        self.server.daemon_threads = True
        threading.Thread(target=self.server.serve_forever, daemon=True).start()


    def wait_for(self, types, order_id=None, timeout=90):
        """Waits for a telegram of one of the types (for an order) and returns (seq, type, fields)."""
        end = time.time() + timeout
```

```
        while time.time() < end:
            try:
                seq, ttype, data = self.inbox.get(timeout=1)
            except queue.Empty:
                continue
            if ttype in types:
                f = describe(ttype, data)
                if order_id is None or f.get('order') == order_id:
                    return seq, ttype, f
        raise TimeoutError('no telegram of type %s' % (types,))
```

Appendix B. Reference of all telegrams

Table 19. All telegrams of the Link dialect. The frame length includes STX, sequence, type and ETX.

Type	Direction	Data fields	Frame length
10	you -> Link	BatchID CHAR(30), OrderID CHAR(30), Item NUM(5), MachineID CHAR(20), ProductID CHAR(30), ReelID CHAR(20), Length_mm NUM(10), Cuts NUM(5), Priority NUM(4), Description CHAR(40), Customer CHAR(30), Free1 CHAR(30)	263
11	you -> Link	OrderID CHAR(30), Item NUM(5), [MachineID CHAR(30)]	44 / 74
00	both	-	10
60	Link -> you	OrderID CHAR(30), Item NUM(5), MachineID CHAR(20)	64
70	Link -> you	OrderID CHAR(30), Item NUM(5), MachineID CHAR(20), ReelID CHAR(20), Measured_mm NUM(12), Cuts NUM(5), EndTime CHAR(14)	115
71	Link -> you	OrderID CHAR(30), Item NUM(5)	44
78	Link -> you	OrderID CHAR(30), Item NUM(5), Code NUM(4), Text CHAR(40)	88
80	Link -> you	MachineID CHAR(20), State CHAR(1), Counter_mm NUM(12), CutNr NUM(5)	47
ACK	both	S: `<STX> SSSSS <ETX>` (7); C: `<STX> SSSSS TT <ETX>` (9)	7 / 9