

TangoSTEP Intelligent Motion Control

White Paper

How to control the stepper motor driver from your PC:

The TangoSTEP is controlled with a 14 Byte command sent through RS-485 serial interface. There is no need for additional drivers or DLLs on your computer, except when using adapters from USB to RS-485. You can use common terminal emulation software like Docklight (www.docklight.de) to send commands to TangoSTEP controllers.

Communication parameters:

Set your serial port to the following basic parameters:

Baudrate: 57600
Data Bits: 8
Parity: None
Stop Bit: 1

Command string structure:

For each command, the structure of the string must be like following example:

1 2 3 4 5 6 7 8 9 10 11 12 13 14
/STX1/STX2/Adr/Pos0/Pos1/Pos2/Pos3/SpeedLo/SpeedHi/Ramp/Modus/Chksum/CR/LF

Description:

STX1: Start byte #1, always value "255"

STX2: Start byte #2, always value "1"

Adr: Address byte, defines controller address (01-15) or broadband (00, all controllers)

Pos0~Pos3: Position bytes, absolute number of microsteps, data type: long, values from -2.147.483.648 to +2.147.483.647, where negative numbers mean moving the motor CCW, positive numbers will move motor CW.

Annotation: Byte Pos0 is lowest byte, Pos3 is highest byte

Position movement is always relative to the actual position !

Example:

You have a stepper motor with 1.8 degrees step width resolution, which equals 200 full steps per rotation. TangoSTEP supports 16 microsteps per full step, so total steps for one rotation of the motor is 3200.

Steps_per_rotation = ((360 degrees/step width) * 16)

For moving CCW, the result must be negative, for CW, it must be positive:

For easier explanation, we convert the steps to HEX format:

CW: Dec 3200 equals HEX C80

CCW: Dec -3200 equals HEX FFFFF380

The 8 digit Hex string can now be split up to four two-digits values:

FFFFFF380 → FF FF F3 80

Where the beginning FF is the highest byte and 80 is the lowest byte
Since the sequence of the byte values must be inverted in the command
string, it will look as follows:

1	2	3	4	5	6	7	8	9	10	11	12	13	14
/STX1/STX2/Adr/Pos0/Pos1/Pos2/Pos3/SpeedLo/SpeedHi/Ramp/Modus/Chksum/CR/LF													
			80	F3	FF	FF							

For CW movement, the Hex value has only three digits in our example. We can
fill up the leading digits with zeros, which will have no effect to Hex
value:

C80 → 00 00 0C 80

After inverting sequence of the two-digit bytes, the command string will
look as follows:

1	2	3	4	5	6	7	8	9	10	11	12	13	14
/STX1/STX2/Adr/Pos0/Pos1/Pos2/Pos3/SpeedLo/SpeedHi/Ramp/Modus/Chksum/CR/LF													
			80	0C	00	00							

SpeedLo, SpeedHi: defines speed value from 10 to 25600. Data type: integer.
Annotation: Byte SpeedLo is lowest byte, SpeedHi is highest byte.

Similar to the above example for the position, we can use the hex
conversion also for the speed. Let's assume a desired speed value of 12000:

Dec 12000 → Hex 2EE0

Split into two bytes with two digits each:

2EE0 → 2E E0

After inverting sequence of the two-digit bytes, the command string will
look as follows:

1	2	3	4	5	6	7	8	9	10	11	12	13	14
/STX1/STX2/Adr/Pos0/Pos1/Pos2/Pos3/SpeedLo/SpeedHi/Ramp/Modus/Chksum/CR/LF													
							E0	2E					

Ramp: defines ramp byte, value from 0 to 255. Will accelerate and
decelerate the motor from zero to full speed and back to zero. The value of
the ramp is multiplied by ten and gives you the total microsteps for each
acceleration and deceleration.

Example:

Microsteps = ramp x 10

So if you select a ramp value of 50, the motor will perform 500 microsteps
before reaching full speed. Also deceleration will be 500 microsteps from
full speed to zero at the end of the travel. The TangoSTEP controller will
automatically deduct these values from total travel value to ensure correct
position:

Example for one full motor rotation with ramp value 50:

Step 1~500: acceleration to full speed
Step 501~2700: full speed
Step 2701~3200: deceleration to zero

The ramp is directly depending on speed value:

$$\text{Ramp_Speed} = ((\text{ramp step} * \text{final speed}) / \text{ramp length})$$

For example, when accelerating to speed 12000 with ramp 50, the speed for each step will look like this:

Step	Speed
1	$((1 * 12000) / (50 * 10)) = 24$
2	$((2 * 12000) / (50 * 10)) = 48$
.	.
.	.
.	.
499	$((499 * 12000) / (50 * 10)) = 11976$
500	$((500 * 12000) / (50 * 10)) = 12000$

However, the ramp time is not linearly, but follows a square function:

$$\text{time} = \text{ramp}^2 / 100$$

Modus: this value determines the operating modus of the controller, which can have the following values:

Value	Modus
1	execute command immediately (move modus)
2	store command into memory and do nothing (learn modus)
0	execute stored command (update modus)
11	motor current control

Sending any command with modus "1" will cause the controller to move the motor without delay.

When using modus "2", the transmitted parameters like position, speed and ramp are stored into the controller's memory, but motor will commit no action until you send any other command with modus "0". This will act as a trigger to the stored parameters and clear controller's memory after execution.

However, if controller's memory does not have any information stored, the trigger command will have no effect.

When sending another store-command to a controller's memory which has already stored information, the old information will be discarded and replaced by the new command.

When sending the trigger with modus "0", there is no need to send parameters like position, speed and ramp again. Just leave this values zero (or any other valid value, it will be ignored by the controller).

This feature is very useful when multiple motors should move at the same time with different position, speed and ramp values. You can send the trigger to the broadcast address "00", and every controller on the bus with stored commands will execute them simultaneously.

Modus "11" provides a special feature, which is a little different from the other modus parameters:

When defining modus "11", the maximum motor current can be limited from 0 to 3000 milliamps in 15 steps. In this case, the ramp byte will be used as current byte. The ramp byte must be within the range of 0 to 15 to provide proper function. Any other values like position and speed will be ignored by the controller. This modus simply defines the maximum motor current of the specified address, all other parameters in the command string will be discarded.

Maximum current = (3000mA * current_byte) / 15

Where current_byte is any integer value from 0 to 15

Example:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14
/STX1/STX2/Adr/Pos0/Pos1/Pos2/Pos3/SpeedLo/SpeedHi/Ramp/Modus/Chksum/CR/LF
                                07 11
```

This command will limit motor current to 1400 mA (47%)

Chksum: Checksum byte, not used and should be "1"

CR: carriage return, Asc value "13"

LF: line feed, Asc value "10"

Respond from controllers:

The TangoSTEP controllers are programmed so they will confirm every executed command with a data string which is sent back to the computer on the same bus. This data string is always identical to the address of the sending controller, e.g. controller #01 will answer with "01", controller #08 will answer with "08" and so on. You can use these answers in your application to check if the controller has committed the requested command. For example, the available TangoSTEP controller software uses this feature to find any errors while executing commands.

For example, when moving the motor to a certain position, the controller will operate the motor and after reaching final destination, the answer will be sent.

Exception: when sending commands in the learn modus (2), the controller will not send any response.

Auto correction: when two or more controllers finish their job at the same time, this may lead to a data corruption of their answers on the bus. The TangoSTEP controllers are programmed so they send out the answers with a very small delay. However, if there is an overlapping and data corruption, the controllers will correct this automatically. Each controller listens to the bus when he is sending out his response. If there is any data corruption, the controller will send his answer again until it is not corrupted anymore. This procedure will be done for a maximum of 15 times (you can only operate 15 controllers on each bus), so data correction is ensured.

End position detection:

TangoSTEP controllers support end position detection switches on your stage to protect the stage from being damaged. When TangoSTEP detects a switch

operation, it will immediately stop the motor and send back his answer. However, in some cases this can be confusing for your software. When moving the stage for example for 10 mm into one direction and receiving the response from the controller, one might assume that the stage has performed the 10 mm travel. But when the stage hits the end switch after 3 mm, the absolute position will differ by 7 mm. Our TangoSTEP Software has a built in feature, which eliminates this malfunction by defining the total travel range and prohibits any position value outside this range.

Command sequence

While the controller is busy with executing the move or update command, he cannot accept any new commands for next movement or storing. This means, while moving the motor, the controller is deaf to any new commands and will discard them. So when sending two move-commands (modus 1) to the same address directly in very short time, the first command will be executed, the second one will be erased. You should always wait for the response of the controller before sending out the next command to it.

Calculating Timeout

In your application it might be mandatory to ensure proper function of the stages. This also includes the interpretation of the responds from the controllers. However, if there are problems with your hardware like power supply failure or data bus interruption, no respond will come back from the controllers. It is very useful to integrate a timeout function into your application so you don't have to wait forever for response.

Our standard TangoSTEP software has this feature already built in. It is based on the following calculation:

execution_time= acceleration_time + full_speed_time + deceleration_time

since acceleration and deceleration time are identical, we can sum up the formula:

execution_time= (2 * ramp_time) + full_speed_time

So if the controller does not respond after this time, there might be a problem with it.

On the other hand, if controller responds earlier, the stage might have hit the end position detection switch.

Error messages

When the controller is powered on or off, it will send special characters to the bus (most likely all above value 15), so your application will not interpret this as a valid response. If you application detects such characters, this is a definite hint that the controller had a temporary power failure. Especially during movement, your absolute position might not be correct and a re-aligning is necessary.