rexroth
A Bosch Company

Reference Manual

Small Handling Walkthrough

V1.0

Bosch Rexroth Corporation

Small Handling Walkthrough & Reference Manual

Contents

Scope

This manual defines Bosch Rexroth's ASCII protocol and the device settings and commands for the Bosch Rexroth Small Handling linear modules running firmware version 7.28. Other products or firmware versions may support other commands or settings; see a product's specific protocol manual for the commands and settings it supports.

Conventions

Conventions used throughout this document:

- `Fixed width` type indicates ASCII characters communicated to and from a device.
- The ↲ symbol indicates a newline. When sending a command to a device, ↲ can be any combination of the Carriage Return (CR, `'\r'`, ASCII code 13) and/or Line Feed (LF, `'\n'`, ASCII code 10) characters. In responses from a device, ↲ includes both the Carriage Return and Line Feed characters.
- When describing command parameters:
    - Square brackets around a parameter indicate it is optional, e.g. **`[axis]`**.
    - Three periods following a parameter indicate that the parameter can be included more than once in a single command, e.g. **`axis`**`...`.
    - An italicized parameter with a < prefix and > suffix must be replaced with data, typically a number, when sending the command, e.g. *`<value>`*.
    - A pipe separating multiple parameters indicates that one of the listed parameters should be provided, e.g. **`abs|rel`**.
    -

Physical Units

Many of the setting values and command parameters described in this document relate to physical units, such as distance, voltage, and time. When a setting value is expressed in a real-world metric unit or in a scaled version of a unit, the units are described explicitly (such as a voltage specified in units of volts, or a current specified in 20 mA increments). In other cases, the setting value is unitless and some scaling factor is required to convert to a practical unit. Unitless values may be referred to in this document as the 'data' value.

In general, the documentation of each command or setting describes how to convert the relevant data values to real-world units. An exception is for values involving physical distances or positions (including speed/velocity and acceleration/deceleration), which are described here. This section is referenced whenever they're used in a command or setting.

Converting data values involving distance or position to real-world values requires a device-specific conversion factor.

For stepper motors, this conversion factor is called "Microstep Size". The value in the User Manual assumes the `resolution` setting is at the default value. If you have modified the resolution value, the Microstep Size can be converted:

$$(\text{Microstep Size})_{\text{new resolution}} = (\text{Microstep Size})_{\text{default resolution}} \times \text{default resolution} / \texttt{resolution}$$

Position

For stepper motor products, all position and distance data values are in units of stepper motor microsteps. The conversion from distance or position data values to real-world units is given by:

$$\text{position} = \text{data} \times (\text{Microstep Size})$$

Velocity

For stepper motor products, the conversion from velocity or speed data values to real-world units is given by:

$$\text{velocity} = \text{data} \times (\text{Microstep Size}) / (1.6384\ \text{s})$$

Acceleration

For stepper motor products, the conversion from acceleration or deceleration data values to real-world units is given by:

$$\text{acceleration} = \text{data} \times (\text{Microstep Size}) \times 10{,}000 / (1.6384\ \text{s}^2)$$

Quick Start

Connecting

Bosch Rexroth Small Handling devices support connecting to user equipment over standard serial connections using a human-readable, text-based protocol - Bosch Rexroth's ASCII protocol. This allows these devices to interface with a variety of equipment and software, including:

- Rexroth Dashboard
- Terminal Emulators
- User programs
- PLCs
- Automation and Instrumentation packages

Bosch Rexroth devices can be up and running in a matter of minutes, no matter what environment is being used.

Small Handling devices typically communicate over RS-232 at 9600 or 115200 baud, with 8 bits, 1 stop bit and no parity. Characters are not echoed by the device, so if a terminal emulator is being used, it is advisable to turn on local echo.

Talking to Bosch Rexroth Devices

Bosch Rexroth devices listen for commands sent to them over a serial port and then immediately respond with a reply. Commands always begin with a `/` and end with a newline. Some commands take parameters, which are separated by spaces. Two example commands are:

```
/1 warnings↵
/1 system reset↵
```

Replies begin with a `@`, end with a newline, and have several space-delimited parameters in between. For example, the most common reply is:

```
@01 0 OK IDLE -- 0
```

This can be broken down into:

```
@                  A reply.
 01                The address of the device sending the reply
    0              The reply scope: 0 for the device or all axes, 1 onwards for an
individual axis
      OK           The command succeeded.
         IDLE      The device isn't moving (or BUSY if it is moving).
              --   No faults or warnings are active.
                 0 The return value, typically 0
```

Devices can also send two other types of messages: Alerts, which start with `!`; and Info, which start with `#`.

Talking to an Individual Device

If there is more than one device in a chain, sending the command `/move abs 10000` will move all devices in the chain at once and return multiple responses. While this is a handy feature for initial setup, general use requires a way to instruct only an individual device to move.

Devices can be addressed by including their device number before the command. For example, the following command instructs only device 1 to move:

```
/1 move abs 10000↵
@01 0 OK BUSY -- 0
```

The valid device addresses are from 1 to 99 inclusive and can include a leading 0 for devices 1 to 9. For example, both 01 and 1 refer to device 1.

Making it Move

It is recommended that the first movement command sent to a device after it is connected and powered is the `home` command. The device will move to the home limit sensor and establish a reference position:

```
/home↵
@01 0 OK BUSY WR 0
```

If the device hasn't been homed, motion commands may not work as expected. The controller's position limits will not match the physical limits, absolute movements will not move to expected positions, and relative movements the device is capable of may be rejected for being outside the travel range. Replies from the device will include the ***No Reference Position*** (`WR`) flag:

```
/move rel 10000↵
@01 0 RJ IDLE WR BADDATA
```

Once the device has been homed, make the device move by sending a `move` command. For example, to move a distance of 10,000 forward from the current position:

```
/move rel 10000↵
@01 0 OK BUSY -- 0
```

To move to the absolute position 10,000, measured from the home position, regardless of the current position:

```
/move abs 10000↵
@01 0 OK BUSY -- 0
```

See the Physical Units section for information about unit conversions.

Changing a Device Setting

All of the device settings are read and modified using the `get` and `set` commands. For example, to query the device maxspeed:

```
/get maxspeed↵
@01 0 OK IDLE -- 153600
```

The maximum speed setting is currently 153,600.

See the Physical Units section for information about unit conversions.

To set the maximum speed, for example, to be twice what was read:

```
/set maxspeed 307200↵
@01 0 OK IDLE -- 0
```

Message Format

The ASCII protocol uses a command-reply model, such that:

- The user must initiate all communication by sending a device a command.
- Unless explicitly disabled, a device always responds with one reply immediately after a command has been received (see the Message IDs section for how to disable replies). Select commands will return additional Info messages.
- Unless explicitly enabled, a device never sends a message unless in response to a command (see Alerts for more information).

The content of the message is space-delimited, with consecutive spaces being treated as a single space. There is only one command or response per message. Sending multiple commands in a single message is not supported.

Reserved Characters

The following characters are reserved and their use is restricted:

- `/`, `@`, `#`, and `!` indicate the message type and must be the first byte of a message.
- Carriage Return (CR, `'\r'`, ASCII code 13), Line Feed (LF, `'\n'`, ASCII code 10), or any combination thereof, indicate a newline (collectively represented as ↵) and the end of a message. All newline characters must be the last bytes of a message.
- `:` indicates the start of a message checksum and must be followed by a two-byte checksum and ↵. See Checksums for more details.
- `\` indicates a line continuation for firmware versions 7.12 and above and must only appear immediately prior to ↵ or the `:` of a message checksum. It may appear in replies or info messages. In firmware versions 7.26 and above it may also appear in command messages. See Line Continuation for more details.
- Bytes 128 to 255 are reserved and may not be present anywhere in a message.

Any message that includes these characters in positions other than those specified above is malformed. If a device receives a malformed command it may not respond, respond with the reason `BADCOMMAND`, or interpret the command in unexpected ways. For example, including `!` as an argument to a command will not elicit a reply:

```
/tools echo hi!↲
```

Similarly, including newline characters, ↲, anywhere but the last bytes of a command will truncate the message, producing an unexpected reply:

```
/tools echo hello↲world↲
@01 0 OK IDLE -- hello
```

Commands

Users send commands to one or more devices, which always and immediately respond with a reply. See the Command Reference section for all the available commands. A command instructs the device to perform an operation. An example of a typical command message and associated fields is:

```
/1 1 move abs 10000↲
/n a xxxx yyy yyyyy[:CC]ff
```

/

Message Type

Size: 1 byte

The message type for a command is always `/`.

This field and the footer are the only required fields; all others are optional.

n

Device Address

Size: 1+ bytes

The address indicates which device number should perform the command. The address is optional: if it is not included, or set to 0, all devices on the chain execute the command. Device addresses range from 1 to 99 inclusive.

Examples of acceptable addresses are:

```
0, 00, 1, 01, 000001, 76, 99, 0x00, 0x01, 0x5A, 0x5a
```

Invalid addresses include:

```
100, -1, 0x65
```

The addresses are out of range and while the message may be valid, no device will respond.

a

Axis Number

Size: 1 byte

The axis number indicates which axis within a device should perform the command. The axis number is optional: if it is not included, or set to 0, the command is executed by all axes of the device. Axis numbers range from 0 to 9 inclusive.

xxxx...

Command

Size: Variable

The command message data contains the command information. The contents are space-delimited.

The Command Reference covers the available commands.

yyy...

Command Parameters

Size: Variable

This message data contains command parameters and data. The contents are space-delimited and case-sensitive. Any numerical values in the message data can either be in decimal format or in hexadecimal and prefixed with `0x`. Negative values are prefixed with '`-`'; positive values may optionally be prefixed with '`+`'.

The Command Reference covers the contents of the parameters field for the available commands.

cc

Message Checksum

Size: 3 bytes.

The message checksum is an optional parameter that, when provided, causes a device to reject a message that has been corrupted during transmission.

For more information about checksums, see Checksums.

ff

Message Footer

Size: 1 - 2 bytes

The footer indicates end-of-message. For convenience, the device accepts any ASCII combination of carriage return (CR, `\r`) and/or line feed (LF, `\n`) as a message footer. In a terminal emulator, this is included when the command is sent by pressing enter or return.

Smallest Command

The smallest valid command is `/↵`, which generates a response from all devices in the chain:

```
/↵
@01 0 OK IDLE -- 0
@03 0 OK IDLE -- 0
@02 0 OK IDLE -- 0
```

This can be used as a quick way to check the statuses of all devices and that they are communicating as expected.

Largest Command

The maximum command packet size is `comm.packet.size.max` characters, including the `/` and `↵`. Devices do not respond to command packets longer than this. In firmware versions 7.26 and above, Line Continuation can be used to send a larger command by splitting it into multiple packets.

Replies

The device sends a reply as soon as it has received a command and determined if it should respond. An example of a typical response message and associated fields is:

```
@01 0 OK IDLE -- 0↵
@nn a fl bbbb ww x[\][:CC]ff
```

`@`

Message Type

Size: 1 byte

The message type for a reply is always `@`.

`nn`

Device Address

Size: 2 bytes

The address indicates the address of the device sending the message, and is always formatted as two digits.

a

Axis Number

Size: 1 byte

The axis number limits the scope of the reply, and has a value in the range of 0 to 9. An axis number between 1 and 9 indicates the reply applies specifically to that axis. An axis number of 0 indicates that the reply applies to the whole device and all axes on it.

fl

Reply Flag

Size: 2 bytes

The reply flag indicates if the message was accepted or rejected and has one the following values:

OK

The command was valid and accepted by the device.

RJ

The command was rejected. The data field of the message will contain one of the following reasons:

AGAIN

The device cannot process the command right now. Send the command again. This response is only in replies to `stream` commands.

BADAXIS

The command's axis number was greater than the number of axes available.

BADCOMMAND

The command or setting was incorrect or invalid.

BADDATA

The command's data was incorrect or out of range.

BADMESSAGEID

The command included a message ID, but the ID was not either `--` or a number from 0 to 99.

BADSPLIT

The command was split into multiple packets in an incorrect way. Possible causes include the parts having different axis numbers or message IDs, an incorrect packet counter value, incorrect positioning of `\` characters, or the use of the line continuation feature in a `renumber` command. See Line Continuation for more details.

DEVICEONLY

An axis number was provided to a device-scope command.

DRIVERDISABLED

One of the drivers is disabled because the driver is or was in an over-temperature, over-voltage, or over-current condition or because the user disabled it with the `driver disable` command. This rejection reason persists until the driver is successfully re-enabled with the `driver enable` command.

FULL

The device cannot allocate more memory for information related to the command. See the documentation for the command that prompted this response for details of this rejection.

LONGWORD

A word in a message was longer than `comm.word.size.max` characters.

NOACCESS

The command or setting is not available at the current access level.

NOTSIN

The axis is not currently moving in response to a `move sin` command, so it cannot be stopped with the `move sin stop` command.

PARKED

The device cannot move because it is currently parked.

Use the `tools parking unpark` command to unpark the device.

STATUSBUSY

The device cannot execute the command because it is currently busy.

SYSTEMERROR

The device failed to execute the command due to a system failure. Use the `system errors` command to review errors.

bbbb

Status

Size: 4 bytes

On devices that control motion axes, the device status is `BUSY` when the axis is moving and `IDLE` otherwise. All motion commands, including `stop`, put the axis into the `BUSY` state while they are being executed. During streamed motion, `wait` commands are considered to be busy, not idle. If the reply message applies to the whole device, the status is `BUSY` if any axis is busy and `IDLE` if all axes are idle.

ww

Warning Flag

Size: 2 bytes

The warning flag is the highest priority warning currently active for the device or axis, or `--` when there are no warnings. For the list of warning flags, see Warning Flags.

`xxx...`

Data

Size: 1+ bytes

The contents and format of the data vary depending on the command the reply relates to. The data is `0` for commands that don't return specific information.

`\`

Line Continuation

Size: 1 byte

Indicates that the message cannot fit within the `comm.packet.size.max` character limit of an ASCII packet and is incomplete. The remainder of the message will be sent in subsequent Info packet(s). See Line Continuation for more details.

Introduced in 7.12

`cc`

Message Checksum

Size: 3 bytes

The `comm.checksum` setting determines whether a reply includes a checksum. For more information about checksums, see Checksums.

`ff`

Message Footer

Size: 2 bytes

The message footer is always composed of a CR-LF combination (`\r\n`).

Warning Flags

Each reply message includes a warning flag, indicating whether any device fault or warning is active. If more than one condition is active, it shows the one with highest priority.

There are three categories of warning flags:

Fault

Flags in this category indicate an event that has immediate consequences to the behaviour of the device. As a result, the device may have deviated from its trajectory or stopped.

Warning

Flags in this category indicate that continued operation may lead to unexpected behaviour.

Note

Flags in this category indicate that an event should be noted by the user before continued operation, but is part of normal device operation.

The warning flags are defined as follows, with the highest priority first:

FF

Fault** - **Critical System Error

The device has experienced a critical system error. This should not occur during normal operation. Please contact Bosch Rexroth Support with the response from `system errors` if this occurs.

FH

Fault** - **Hardware Emergency Stop Driver Disabled

The device has disabled the driver in response to a hardware emergency stop signal.

This warning persists until the driver returns to normal operating conditions, the hardware emergency stop is deactivated, and the user re-enables the driver with a `driver enable` command.

FV

Fault** - **Overvoltage or Undervoltage Driver Disabled

The device has disabled the driver in response to the driver voltage being outside the normal operating range. This can occur if a stall has occurred, if the motor is back driven, if there is damage to the driver hardware, or if there is a problem with the power supply.

This warning persists until the driver voltage returns to normal operating conditions, and the user re-enables the driver with a `driver enable` command.

This warning may appear briefly at power-up.

FO

Fault** - **Driver Disabled

The device has disabled the driver.

This warning may appear at power-up or after a reset, while the device starts up. Once the device is ready, the driver is automatically enabled and the warning flag is cleared.

This warning will also appear after the user has sent a `driver disable` command. In this case, it will persist until the user re-enables the driver with a `driver enable` command.

Introduced in 7.11

FC

Fault** - **Current Inrush Error

The device has disabled the driver due to an excessive inrush of current from the power supply. This may occur due to a problem with the driver or motor from its supply. It may also occur when power supplies are connected to or disconnected from a controller supporting multiple power supplies or devices in a daisy-chain.

This warning persists until the user re-enables the driver with a `driver enable` command.

This warning may also appear at power-up or after a reset, while the device starts up. Once the device is ready, the driver is automatically enabled and the warning flag is cleared.

Introduced in 7.11

FM

Fault** - **Motor Temperature Error

The device has disabled the driver due to an overheating motor.

This warning persists until the motor temperature returns to normal operating conditions and the user re-enables the driver with a `driver enable` command.

Warning: Continually driving the motor at power levels sufficient to repeatedly trigger this fault may permanently damage the motor.

Introduced in 7.11

FD

Fault** - **Driver Temperature/Current Error

The device has disabled the driver due to an overheating driver or an over-current condition.

This warning persists until the driver temperature and current return to normal operating conditions and the user re-enables the driver with a `driver enable` command.

FQ

Fault** - **Encoder Error

The encoder-measured position may be unreliable. The encoder has encountered a read error due to poor sensor alignment, vibration, dirt or other environmental conditions. Reset the encoder reference position by sending `system reset` followed by a `home` operation.
This warning persists until acknowledged and cleared by the user with the `warnings clear` command.
Please contact Bosch Rexroth Support if this error persists.

FS

Fault** - **Stalled and Stopped

The axis detected a stall and has stopped; the requested movement was not completed.
This warning persists until acknowledged and cleared by the user with the `warnings clear` command.

FB

Fault** - **Stream Bounds Error

The device could not execute a previously sent `stream` movement because it failed a precondition (e.g. motion exceeds device bounds, calls nested too deeply).
Obtain the error reason with the `stream info` command.
This warning persists until acknowledged and cleared by the user with the `warnings clear` command or until the stream is disabled with the `stream setup disable` command. Until the warning is cleared, no further streamed motions can be sent to the failed stream.

FE

Fault** - **Limit Error

Either the axis could not reach the target limit sensor or the sensor is faulty.
This warning persists until acknowledged and cleared by the user with the `warnings clear` command.
While homing or spontaneously encountering a limit sensor, this flag may appear if the offset position dictated by `limit.home.offset` is outside the range `limit.min` to `limit.max`.

WL

Warning** - **Unexpected Limit Trigger

A movement operation did not complete due to a triggered limit sensor. This flag is set if a limit sensor interrupts a movement operation and the ***No Reference***

Position (`WR`) warning flag is not present. This may be an indication that the axis has slipped or one of `limit.min` and `limit.max` is incorrect.

This warning persists until acknowledged and cleared by the user with the `warnings clear` command.

WV

Warning - ***Voltage Out of Range***

The supply voltage is outside the recommended operating range of the device. Damage to the device could occur if not remedied.

This warning persists until the condition is remedied.

WT

Warning - ***Temperature High***

The internal temperature of the controller has exceeded the recommended limit for the device.

This warning persists until the over-temperature condition is remedied.

See `driver.temperature`.

See `system.temperature`.

WS

Warning - ***Stalled with Recovery***

The axis detected a stall, but has recovered or is recovering.

This warning persists until acknowledged and cleared by the user with the `warnings clear` command.

WM

Warning - ***Displaced When Stationary***

While not in motion, the axis has been forced out of its position.

This warning persists until the axis is moved.

WP

Warning - ***Invalid Calibration Type***

There is calibration data saved which this axis configuration does not support.

This warning persists until the axis configuration is changed to support the data or until the data is deleted.

WR

Warning - ***No Reference Position***

The axis does not have a reference position.

This warning persists until the axis position is updated by homing or any command/action that sets position.

WH

Warning - ***Device Not Homed***

The axis has a position reference, but has not been homed.

This warning persists until the axis has been homed.

NC

Note - ***Manual Control***

The axis is busy due to manual control via the knob.

This warning persists until a movement command is issued.

NI

Note - ***Movement Interrupted***

The device prematurely ended a previous motion command or displacement recovery in order to execute a newly-received movement operation (either a command or manual control). This indicates that a movement command or displacement recovery did not complete. This flag will not occur if the movement that was prematurely ended was initiated using the manual knob.

This warning persists until a movement command is issued when the axis is either idle or executing a manual control movement.

ND

Note - ***Stream Discontinuity***

The axis has slowed down while following a streamed motion path because it has run out of queued motions.

This warning persists until the stream has enough motions queued that it no longer needs to decelerate for that reason, or until the stream is disabled.

NR

Note - ***Value Rounded***

One or more values in a command exceeded their specified levels of precision and were rounded.

This warning persists until acknowledged and cleared by the user with the `warnings clear` command.

NT

Note - ***Value Truncated***

The value could not fit into a single packet and was truncated. See Line Continuation for more details.

This warning persists until acknowledged and cleared by the user with the `warnings clear` command.

Introduced in 7.12

Info

Info messages contain extra information from the device. One or more info messages can follow a reply to select commands.

An example of a typical info message and its fields is:

```
#01 0 Visit https://www.boschrexroth.com/en/us/products/product-groups/linear-motion-technology/topics/linear-axes/modules/small-handling/ for instruction manuals.↵
#nn a xxxxxxxxxxxx...[\][:CC]ff
```

#

Message Type

Size: 1 byte

The message type for an info message is always `#`.

nn

Device Address

Size: 2 bytes

The device address contains the address of the device sending the info, and is always formatted as two digits.

a

Axis Number

Size: 1 byte

The axis number limits the scope of the info message, and has a value in the range of 0 to 9. An axis number between 1 and 9 indicates the info message applies specifically to that axis. An axis number of 0 indicates that the info message applies to the whole device and all axes on it.

xxx...

Data

Size: 1+ bytes

The data for the info message contains command specific content.

****

Line Continuation

Size: 1 byte

Indicates that the message cannot fit within the `comm.packet.size.max` character limit of an ASCII packet and is incomplete. The remainder of the message will be sent in subsequent Info packet(s). See Line Continuation for more details.

Introduced in 7.12

cc

Message Checksum

Size: 3 bytes

The `comm.checksum` setting determines whether an info message includes a checksum.

For more information about checksums, see Checksums.

ff

Message Footer

Size: 2 bytes

The message footer always contains a CR-LF combination (`\r\n`) for an info message.

Alerts

The device sends an alert message when an axis' status changes to `IDLE`. For example, on motion axes, `BUSY` indicates that the axis is moving, so the device will send an alert when motion completes.

Alerts are controlled by the `comm.alert` setting, which must be 1 for the device to send status alerts. If it is enabled, an alert can be sent at any time without being preceded by a command from the user.

This message type is used for informational purposes or time-sensitive operations.

An example of a typical alert message and its fields is:

```
!01 1 IDLE --↵
!nn a ssss ww x[:CC]ff
```

!

Message Type

Size: 1 byte

The message type for an alert message is always `!`.

nn

Device Address

Size: 2 bytes

The address indicates the address of the device sending the message, and is always formatted as two digits.

a

Axis Number

Size: 1 byte

The axis number limits the scope of the alert, and has a value in the range of 1 to 9. The axis number indicates that the alert applies specifically to that axis.

ssss

Device Status

Size: 4 bytes

On devices that control motion axes, the device status is `BUSY` when the axis is moving and `IDLE` otherwise. All motion commands, including `stop`, put the axis into the `BUSY` state while they are being executed. During streamed motion, `wait` commands are considered to be busy, not idle.

ww

Warning Flag

Size: 2 bytes

The warning flag is the highest priority warning currently active for the device or axis, or `--` when there are no warnings. For the list of warning flags, see Warning Flags.

xxx...

Data

Size: 0+ bytes

The contents and format of the data vary depending on the purpose of the alert. The data is omitted for alerts that do not need it.

cc

Message Checksum

Size: 3 bytes

The checksum is included if the `comm.checksum` setting is configured to 1. For more information about checksums, see Checksums.

ff

Message Footer

Size: 2 bytes

The message footer is always composed of a CR-LF combination (`\r\n`).

Command Reference

The following section details available ASCII commands.

Commands are associated with either the device or its axes, which is referred to as the command's scope. For device-scope commands, specifying an axis number other than 0 results in a DEVICEONLY error:

```
/1 tools parking park↵
@01 0 OK IDLE -- 0
/1 0 tools parking park↵
@01 0 OK IDLE -- 0
/1 1 tools parking park↵
@01 1 RJ IDLE -- DEVICEONLY
```

For axis-scope commands, specifying an axis number of 0 (or omitting the axis number) directs the command to all axes on the device. If one of the axes is unable to accept the command, a BADDATA response is returned and none of the axes execute the command. For example, moving to a position that is valid for one axis, but not another, results in an error:

```
/1 get limit.max↵
@01 0 OK IDLE -- 3038763 6062362
/1 move abs 4750000↵
@01 0 RJ IDLE -- BADDATA
```

driver

These commands allow enabling and disabling the axis driver (which controls current to the motor).

See the driver settings section for settings related to the axis driver.

driver disable

Disables the axis driver

Parameters

None

Scope

Axis

Access Level

Normal

Disables the axis driver, which prevents current from being sent to the motor. On firmware versions 7.11 and above, raises the ***Driver Disabled*** (FO) warning flag. If the driver is already disabled when this command is sent, the command is accepted and the driver remains disabled.

Warning: Exercise caution while using this command, as the motor will immediately stop operating and will not maintain the position or exert any force.

Note: Removing power will reset the driver state. The device always enables drivers after power-up.

`driver enable`

Enables the axis driver

Parameters

None

Scope

Axis

Access Level

Normal

Attempts to enable the axis driver. If successful, the ***Overvoltage or Undervoltage Driver Disabled*** (`FV`), ***Driver Disabled*** (`FO`), ***Current Inrush Error*** (`FC`), ***Motor Temperature Error*** (`FM`), and ***Driver Temperature/Current Error*** (`FD`) warning flags are cleared. However, if a fault condition is still present, the driver is not enabled; the command is rejected with the reason `DRIVERDISABLED`. If the driver is already enabled when this command is sent, the command is accepted and the driver remains enabled.

`get <setting>`

Returns the current value of a device or axis setting

Parameters

`<setting>`

The name of one of the device settings

Scope

Device and Axis

Access Level

Normal

Returns the current value of the setting `<setting>`.

See the Setting Reference for a detailed list of settings and what they do.

Example: Reading the value of a setting

Reading the `device.id` setting, which in this example is 50106 (X-LSQ150B):

```
/get device.id↵
@01 0 OK IDLE -- 50106
```

Example: Attempting to read a non-existent setting

The setting "nonexistent.setting" does not exist. Attempting to read it results in a BADCOMMAND rejection reply:

```
/get nonexistent.setting↵
@01 0 RJ IDLE -- BADCOMMAND
```

Not all settings exist on every Bosch Rexroth product: an attempt to read a setting which is valid on one Bosch Rexroth product may be rejected with BADCOMMAND on a product with different features.

home

Moves the axis towards the home sensor to set the reference position

Parameters

None

Scope

Axis

Access Level

Normal

Moves the axis towards the home position at the lesser of limit.approach.maxspeed and maxspeed until the home sensor is triggered. Once the home position is reached, the current position is reset to limit.home.preset, establishing the reference position. Additionally, limit.home.triggered is set to 1, and the ***No Reference Position*** (WR) warning flag is cleared.

This command is similar to issuing /tools gotolimit home neg 2 0. On some products, the axis may reverse direction when searching for the home sensor during homing. However, tools gotolimit will only search in the specified direction.

Note: On power-up, issue this command to obtain a reference position. Otherwise, motion commands may respond with a rejection reply or behave unexpectedly.

Note: If limit.home.action is set to 2 or 3, and limit.home.offset is non-zero, the axis will move to the offset position instead of the home position at the end of homing.

Note: When homing, the axis attempts to move towards the home sensor and ignores the value of limit.min. To initialize the axis without homing, use the tools parking park command to park the device before powering down and the tools parking unpark command to unpark it when power is restored.

Example: Homing

```
/home↵
@01 0 OK BUSY WR 0
```

io

These commands provide access to the digital or analog input or output (IO) channels on the device.

There are four types of IO channels:

ao
: Analog output

ai
: Analog input

do
: Digital output

di
: Digital input

All IO channels of a particular type are collectively called a ***port*** and each channel is identified by its position within the port. For instance, `do 1` is the first digital output. Note that channels only have one type and cannot change types, e.g., a digital output cannot function as a digital input.

Not all IO types are available on every device. Use the `io info` command to see the IO channels available on a particular device.

io get di <channel>|port

Returns the state of a digital input channel or port

Parameters

<channel>

The digital input channel number

Must be in the range 1 to the value returned by `io info di`

port

Return the states for all digital input channels

Scope

Device

Access Level

Normal

This command returns the state of the specified digital input channel. If the **port** parameter is provided, this command returns a space-separated list of values, one for each channel in the port. The command returns 0 for low or 1 for high.

In firmware versions 7.22 and above, the state of the digital input port can also be read using the `io.di.port` setting.

Introduced in 7.07

Example: Reading a digital input

```
/io get di 2↵
@01 0 OK IDLE -- 1
```

There is a high signal on digital input 2.

Example: Reading all digital inputs

```
/io info di↵
@01 0 OK IDLE -- 2
/io get di port↵
@01 0 OK IDLE -- 0 1
```

The device has two digital inputs: digital input 1 is low, and digital input 2 is high.

`io get do <channel>|port`

Returns the state of a digital output channel or port

Parameters

`<channel>`

The digital output channel number

Must be in the range 1 to the value returned by `io info do`

`port`

Return the states for all digital output channels

Scope

Device

Access Level

Normal

This command returns the state the device is currently driving the specified digital output channel at. If the **`port`** parameter is provided, this command returns a space-separated list of values, one for each channel in the port. The command returns 0 for low or 1 for high.

In firmware versions 7.22 and above, the state of the digital output port can also be read using from the `io.do.port` setting.

Introduced in 7.07

Example: Reading a digital output

```
/io get do 1↵
@01 0 OK IDLE -- 0
```

The device is driving digital output 1 low.

Example: Reading all digital outputs

```
/io info di↵
@01 0 OK IDLE -- 2
/io get di port↵
@01 0 OK IDLE -- 0 1
```

The device has two digital outputs: digital output 1 is low, and digital output 2 is high. (note: inputs and outputs are 24VDC)

io info [<type>]

Returns the number of IO channels on the device

Parameters

<type>

The IO type (optional)

Must be one of `ao` (analog output), `ai` (analog input), `do` (digital output), or `di` (digital input).

Scope

Device

Access Level

Normal

Returns the number of IO channels available on the device, grouped by type. They are returned in the order `ao`, `ai`, `do`, and then `di`.

If the optional **`type`** parameter is specified, only the number of IO channels of that type is returned.

Introduced in 7.07

Example: Reading the number of IO channels

```
/io info↵
@01 0 OK IDLE -- 0 1 4 3
```

The device has no analog outputs, one analog input, four digital outputs, and three digital inputs.

Example: Reading the number of digital inputs

```
/io info di↵
@01 0 OK IDLE -- 3
```

The device has three digital inputs.

io set do <channel>|port <action>...

Sets the state of a digital output channel

Parameters

<channel>

The channel number

Must be in the range 1 to the value returned by `io info do` for the given channel type

<action>

The action can be one of `0` (low), `1` (high), `t` (toggle), or `k` (keep).

If a channel is specified, then provide a single action to perform on the channel.

If the port parameter is provided, then provide a space-separated list of actions, one for each channel in the port.

Scope

Device

Access Level

Normal

Sets the state of digital output channels.

In firmware versions 7.22 and above, the state of the digital output port can also be set using the `io.do.port` setting.

Introduced in 7.07

Example: Setting a digital output

```
/io set do 2 1↵
@01 0 OK IDLE -- 0
```

Digital output 2 is now outputting high.

Example: Toggling a digital output

```
/io get do 1↵
@01 0 OK IDLE -- 0
/io set do 1 t↵
@01 0 OK IDLE -- 0
/io get do 1↵
@01 0 OK IDLE -- 1
/io set do 1 t↵
@01 0 OK IDLE -- 0
/io get do 1↵
@01 0 OK IDLE -- 0
```

Digital output 1 was toggled from `0` (low) to `1` (high) and back again.

Example: Attempting to set a non-existent channel

```
/io info do↵
@01 0 OK IDLE -- 2
/io set do 5 1↵
@01 0 RJ IDLE -- BADDATA
```

The device only has 2 digital output channels so attempting to set the non-existent digital output 5 is rejected.

Example: Setting multiple output channels

```
/io get do port↵
@01 0 OK IDLE -- 0 1 1 1
/io set do port 1 0 k t↵
@01 0 OK IDLE -- 0
/io get do port↵
@01 0 OK IDLE -- 1 0 1 0
```

Digital output 1 was set high, digital output 2 was set low, digital output 3 was kept high, and digital output 4 was toggled from high to low.

move

These commands provide the basic methods for moving an axis.
The `move abs` and `move rel` commands move the axis to a position, while the `move vel` command moves the axis at a specified velocity. The `move min|max`, `move index next|prev`, and `move index` commands move the axis to predetermined positions along its travel. The `move sin` command moves the axis along a sinusoidal trajectory.
These commands, except `move sin`, will pre-empt any prior movements that the axis was executing.

Example: Moving beyond the range of the axis

If a specified position is beyond the range of the axis, a `move` command will be rejected with the reason `BADDATA`:

```
/get limit.max↵
@01 0 OK IDLE -- 305381
/move abs 305888↵
@01 0 RJ IDLE -- BADDATA
```

move abs *<position>* [*<maxspeed>* [*<accel>*]]

Moves to the specified absolute *<position>*

Parameters

<position>

The absolute position to move to

Must be in the range limit.min to limit.max

<maxspeed>

The motion's maximum speed (optional)

Must be in the valid data range for maxspeed

Introduced in 7.25

<accel>

The motion's acceleration (optional)

Must be in the valid data range for accel

Introduced in 7.25

Scope

Axis

Access Level

Normal

Moves the axis to the specified absolute *<position>*.
To move the axis a relative distance, use the move rel command.
If the *<maxspeed>* and *<accel>* parameters are specified, they will be used instead of the global maxspeed and accel values. The global maxspeed and accel settings are not changed.
See the Physical Units section for information about unit conversions.

Example: Moving all axes

Move all axes on a device to the absolute position 200,000:

```
/move abs 200000↵
@01 0 OK BUSY -- 0
```

Example: Moving all axes with custom maximum speed and acceleration

Move all axes on a device to the absolute position 200,000 with maximum speed 10,000 and acceleration 200:

```
/move abs 200000 10000 200↵
@01 0 OK BUSY -- 0
```

move index *<number>* [*<maxspeed>* [*<accel>*]]

Moves the axis to an index position

Parameters

<number>
The index number, which must be positive

<maxspeed>

The motion's maximum speed (optional)

Must be in the valid data range for `maxspeed`

Introduced in 7.25

<accel>

The motion's acceleration (optional)

Must be in the valid data range for `accel`

Introduced in 7.25

Scope

Axis

Access Level

Normal

Moves the axis to the index position specified by *<number>*. Index positions are positions separated by `motion.index.dist`. For a provided *<number>*, the axis moves to the absolute position (*<number>* - 1) × `motion.index.dist`.
If the *<maxspeed>* and *<accel>* parameters are specified, they will be used instead of the global `maxspeed` and `accel` values. The global `maxspeed` and `accel` settings are not changed.
For linear devices, the target position must be within the valid travel of the device, i.e., in the range `limit.min`, `limit.max`.

Example: Moving to an index

Move the axis to index position 5:

```
/move index 5↵
@01 0 OK BUSY -- 0
```

Example: Moving to an index with custom maximum speed and acceleration

Move the axis to index position 5 with maximum speed 10,000 and acceleration 200:

```
/move index 5 10000 200↵
@01 0 OK BUSY -- 0
```

move index next|prev [*<maxspeed>* [*<accel>*]]

Moves the axis to the next or previous index position

Parameters

next
Move the axis to the next index position

prev
Move the axis to the previous index position

<maxspeed>

The motion's maximum speed (optional)

Must be in the valid data range for `maxspeed`

Introduced in 7.25

<accel>

The motion's acceleration (optional)

Must be in the valid data range for `accel`

Introduced in 7.25

Scope

Axis

Access Level

Normal

Moves the axis to the next or previous index position. See also `move index`.
If the move would place the axis outside of the range `limit.min` to `limit.max`, the command will be rejected with the reason `BADDATA`.
If the *<maxspeed>* and *<accel>* parameters are specified, they will be used instead of the global `maxspeed` and `accel` values. The global `maxspeed` and `accel` settings are not changed.

Introduced in 7.14

Example: Moving to the next index position

Move all axes to their next index positions:

```
/move index next↵
@01 0 OK BUSY -- 0
```

Example: Moving to the next index position with custom maximum speed and acceleration

Move all axes to their next index positions with maximum speed 10,000 and acceleration 200:

```
/move index next 10000 200↵
@01 0 OK BUSY -- 0
```

`move min|max [<maxspeed> [<accel>]]`

Moves the axis to the minimum or maximum position

Parameters

`min`
Move the axis to the minimum position

`max`
Move the axis to the maximum position

`<maxspeed>`

The motion's maximum speed (optional)

Must be in the valid data range for `maxspeed`

Introduced in 7.25

`<accel>`

The motion's acceleration (optional)

Must be in the valid data range for `accel`

Introduced in 7.25

Scope

Axis

Access Level

Normal

Moves the axis to the minimum position, as specified by `limit.min`, or the maximum position, as specified by `limit.max`.

If the *<maxspeed>* and *<accel>* parameters are specified, they will be used instead of the global `maxspeed` and `accel` values. The global `maxspeed` and `accel` settings are not changed.

Example: Moving to the maximum position

Move all axes to their maximum positions:

```
/move max↵
@01 0 OK BUSY -- 0
```

Example: Moving to the maximum position with custom maximum speed and acceleration

Move all axes to their maximum positions with maximum speed 10,000 and acceleration 200:

```
/move max 10000 200↵
@01 0 OK BUSY -- 0
```

`move rel <distance> [<maxspeed> [<accel>]]`

Moves the axis a distance specified by *<distance>*

Parameters

<distance>

The relative distance to move by

Must be in the range `limit.min` - `pos` to `limit.max` - `pos`

<maxspeed>

The motion's maximum speed (optional)

Must be in the valid data range for `maxspeed`

Introduced in 7.25

<accel>

The motion's acceleration (optional)

Must be in the valid data range for `accel`

Introduced in 7.25

Scope

Axis

Access Level

Normal

Moves the axis the specified *<distance>* from the current position. The distance may be positive or negative.

To move the axis to an absolute position, use the `move abs` command.

If the *<maxspeed>* and *<accel>* parameters are specified, they will be used instead of the global `maxspeed` and `accel` values. The global `maxspeed` and `accel` settings are not changed.

See the Physical Units section for information about unit conversions.

Example: Moving all axes

Move all axes on a device forward a distance of 200,000, relative to the current position:

```
/move rel 200000↵
@01 0 OK BUSY -- 0
```

Example: Moving all axes with custom maximum speed and acceleration

Move all axes on a device to the relative position 200,000 with maximum speed 10,000 and acceleration 200:

```
/move rel 200000 10000 200↵
```

```
@01 0 OK BUSY -- 0
```

move sin <amplitude> <period> [<count>]

Moves the axis in a sinusoidal trajectory

Parameters

<amplitude>

Amplitude of the sinusoidal motion

<period>

Period of the sinusoidal motion in milliseconds

Must be in the range 0.2 ms to 429,496,729.5 ms

<count>

Number of repeated cycles (optional)

Must be in the range 1 to 4,294,967,294

Scope

Axis

Access Level

Normal

Starts moving the axis along a sinusoidal path.

The amplitude of the sinusoidal motion is specified by *<amplitude>*. The amplitude is half of the motion's peak-to-peak amplitude. A positive number implies that the sinusoidal motion starts at the minimum position and moves in a positive direction from the starting position. A negative number implies that the sinusoidal motion starts at the maximum position and moves in a negative direction from the starting position. The minimum and maximum positions of the sinusoidal motion must both be within the range `limit.min` to `limit.max`.

The period of the sinusoidal motion is specified by *<period>* in milliseconds, with resolution down to 0.1 ms. A minimum value of 1 ms is recommended to achieve a reasonable resolution.

The optional parameter *<count>* specifies the number of repeated cycles.

If *<count>* is not provided, the motion will continue until another motion or `stop` command pre-empts it. To stop the sinusoidal motion after the current cycle completes, use the `move sin stop` command.

Since the sinusoidal motion starts at the minimum or maximum of the sinusoid, the initial velocity of the motion is zero. As a result, the motion can only be started while the axis is `IDLE`.

The peak velocity and peak acceleration data values during the sinusoidal motion can be determined from the *<amplitude>* and *<period>* parameters, as defined in the following equations. The device does not validate that the peak velocity and peak acceleration resulting from the *<amplitude>* and *<period>* are reasonable; it is up to the user to ensure they are low enough to prevent the device from stalling. If the peak velocity would exceed the valid data range of `maxspeed`, then the command will be rejected.

The peak velocity is:
v_{peak} = Amplitude × 2π / Period

The peak acceleration is:
a_{peak} = Amplitude × [2π / Period]^2

Sometimes it is useful to calculate these quantities in Bosch Rexroth distance units, so they can be compared with other setting values. See the Physical Units section for information about unit conversions. The peak velocity is:
v_{peak} = |**amplitude**| × (2π × 1000) / **period** × 1.6384

The peak acceleration (in Bosch Rexroth acceleration data units) is:
a_{peak} = |**amplitude**| × [(2π × 1000) / **period**]^2 × (1.6384 / 10000)

Note: A sinusoidal motion has limited ability to recover from a stall.

Example: Starting a sinusoidal motion that runs for two cycles

Start sinusoidal motion with an amplitude of 200 `pos` units and a period of 1 second (1000 ms) that will stop after two cycles:

```
/move sin 200 1000 2↵
@01 0 OK BUSY -- 0
```

Example: Attempting to start sinusoidal motion on a busy axis

Start a velocity move and then request sinusoidal motion:

```
/move vel 20↵
@01 0 OK BUSY -- 0
/move sin 200 1000 1↵
@01 0 RJ BUSY -- STATUSBUSY
```

The `move sin` command is rejected because the axis is busy.

move sin stop

Stops sinusoidal motion at the end of a cycle

Parameters

None

Scope

Axis

Access Level

Normal

Stops motion initiated using the `move sin` command when the current cycle completes. To stop the motion before the current cycle completes, use the `stop` command. If the axis is not performing sinusoidal motion, the command is rejected with the reason `NOTSIN`.

Example: Stopping a sinusoidal motion

Start sinusoidal motion with amplitude 200, period 1000, and an indefinite number of cycles, then stop it at the end of a complete cycle:

```
/move sin 200 1000↵
@01 0 OK BUSY -- 0
/move sin stop↵
@01 0 OK BUSY -- 0
```

move stored *<number>* [*<maxspeed>* [*<accel>*]]

Moves to a stored position

Parameters

<number>

The number of the stored position to move to

Must be in the range 1 to 16

<maxspeed>

The motion's maximum speed (optional)

Must be in the valid data range for `maxspeed`

Introduced in 7.25

<accel>

The motion's acceleration (optional)

Must be in the valid data range for `accel`

Introduced in 7.25

Scope

Axis

Access Level

Normal

Moves the axis to the stored position specified by *<number>*. Each stored position defaults to position 0 until another position is saved to it, so moving to a stored position that has not been set will move the axis to position 0. To store a position, use `tools storepos current` or `tools storepos position`. To read the value of a stored position, use `tools storepos`.
If the *<maxspeed>* and *<accel>* parameters are specified, they will be used instead of the global `maxspeed` and `accel` values. The global `maxspeed` and `accel` settings are not changed.

Example: Moving to a stored position

Move to the position in stored position number 3:

```
/move stored 3↵
@01 0 OK BUSY -- 0
```

Example: Moving to a stored position with custom maximum speed and acceleration

Move to the position in stored position number 3 with maximum speed 10,000 and acceleration 200:

```
/move stored 3 10000 200↵
@01 0 OK BUSY -- 0
```

move vel *<velocity>* [*<accel>*]

Moves the axis at the specified *<velocity>*

Parameters

<velocity>

The velocity to move at

For stepper motor products, must be in the range −`resolution` × 16,384 to `resolution` × 16,384

<accel>

The motion's acceleration (optional)

Must be in the valid data range for `accel`

Introduced in 7.25

Scope

Axis

Access Level

Normal

Moves the axis at the specified *<velocity>* until limit.min or limit.max is reached, a limit sensor is triggered, or the axis is pre-empted by another movement command such as stop.
If the *<accel>* parameter is specified, it will be used instead of the global accel value. The global accel setting is not changed.
See the Physical Units section for information about unit conversions.

Example: Moving all axes at a specific velocity

Move all axes on a device at velocity 20:

```
/move vel 20↵
@01 0 OK BUSY -- 0
```

Example: Moving all axes with custom acceleration

Move all axes on a device at velocity 20 with acceleration 200:

```
/move vel 20 200↵
@01 0 OK BUSY -- 0
```

renumber [*<value>*]

Renumbers the address of one or more devices

Parameters

<value>
1 to 99 (optional)

Scope

Device

Access Level

Normal

When this command is sent to all devices in a daisy-chain, it sequentially sets comm.address on each device. The device closest to the computer is given an address of *<value>*, if provided. Otherwise, it is set to 1. The remaining devices in the daisy-chain are numbered consecutively.
When renumber is sent to a specific device address, it sets comm.address to *<value>* on the specified device only. In this case, providing *<value>* is required.
Note: When a device receives a renumber command, it responds using the new device address, not the address the command was sent to.

Note: The Line Continuation feature cannot be used with the renumber command. The device rejects a renumber command sent using Line Continuation with the reason BADSPLIT.

Example: Renumbering all devices in a chain

```
/renumber↵
@01 0 OK IDLE -- 0
@02 0 OK IDLE -- 0
```

The devices are renumbered, with the device closest to the computer having address 1, and the next closest having address 2.

Example: Renumbering a specific device

Renumber device 2 to device 4:

```
/2 renumber 4↵
@04 0 OK IDLE -- 0
```

This is equivalent to `/2 set comm.address 4↵`.

Example: Renumbering to an invalid value

```
/renumber 999↵
@01 0 RJ IDLE -- BADDATA
```

The requested device address was outside the allowable range.

scope

These commands provide tools for capturing real-time data for analysis.

The scope commands, along with the settings `scope.timebase` and `scope.delay`, provide a mechanism to capture any numerical `setting` (or group of settings) at a regular interval down to 0.1 ms (10,000 samples per second). The device has six channels, allowing up to six settings to be captured at once. The device begins capturing setting values after receiving the `scope start` command and the delay specified by `scope.delay` has elapsed. Setting values are captured until the scope buffer is full or the `scope stop` command is received. The captured data points are stored until the next `scope start` command is received or the device is reset.

Example: Sampling multiple settings during a move

In this example the planned trajectory position (`pos`) and the encoder-measured position (`encoder.pos`) are captured during a constant-velocity move. The `sampling timebase` is set to 0.1 ms. The `scope.delay` setting of 0 causes sampling to start as soon as the `scope start` command is received.

Only the first five samples for each captured channel are shown here (the omitted samples are indicated with `...`). See `scope print` for details on the data format.

```
/set scope.timebase 0.1↲
@01 0 OK IDLE -- 0
/set scope.delay 0↲
@01 0 OK IDLE -- 0
/scope add pos↲
@01 0 OK IDLE -- 0
/scope add encoder.pos↲
@01 0 OK IDLE -- 0
/move vel 10000↲
@01 0 OK BUSY -- 0
/scope start↲
@01 0 OK BUSY -- 0
/scope print↲
@01 0 OK BUSY -- 0
#01 0 count 1024 chan 2
#01 0 chan 1 pos axis 1
#01 0 data 14058
#01 0 data 14059
#01 0 data 14059
#01 0 data 14060
#01 0 data 14061
...
#01 0 chan 2 encoder.pos axis 1
#01 0 data 14117
#01 0 data 14122
#01 0 data 14125
#01 0 data 14122
#01 0 data 14125
...
```

scope add *<setting>*

Adds a scope channel to capture a setting

Parameters

<setting>
The name of the setting to capture

Scope

Axis

Access Level

Normal

When sent to an individual axis, this command adds a channel to capture the *<setting>* for that axis. If the command is sent to all axes then a channel capturing the *<setting>* is added for each axis on the device.

The device has six channels, allowing up to six settings to be captured at once. If the requested number of channels is more than this, the command is rejected with `FULL`.

To clear the channel list use the `scope clear` command.

See the Setting Reference for a list of settings.

If a scope capture is in progress, this command is rejected with STATUSBUSY.
Most settings can be captured using the scope feature. If the setting cannot be captured, the command is rejected with BADDATA.
If an invalid setting name is provided, the command is rejected with BADCOMMAND.

Example: Capturing multiple settings on one axis

Clear any existing channels and then add two channels to capture pos and limit.home.triggered for axis 1 on device 2:

```
/2 scope clear↵
@02 0 OK IDLE -- 0
/2 1 scope add pos↵
@02 1 OK IDLE -- 0
/2 1 scope add limit.home.triggered
@02 1 OK IDLE -- 0
```

scope clear

Clears the scope channel list and captured data

Parameters

None

Scope

Device

Access Level

Normal

Clears all scope channels of assigned settings and clears any previously captured data.

Add a channel to capture a setting using the scope add command.

If a scope capture is in progress, this command is rejected with STATUSBUSY. To stop the capture, send scope stop, then send scope clear to clear the channel list.

scope print

Prints the captured data

Parameters

None

Scope

Device

Access Level

Normal

Prints the results of a scope capture.

Data is printed in the format described below. If a capture has not started or has completed (either manually with the scope stop command or automatically when the scope buffer becomes full), the command will print whatever data has been captured (if

any). If a capture has started, but is not yet complete, this command is rejected with `STATUSBUSY`.

Scope Output Format

After the standard reply, the `scope print` command prints the content of the scope capture as a series of Info messages. The first message describes the number of data points per channel and the number of channels in the capture:

```
#01 0 count 1024 chan 2
```

In this case, 2 channels were captured with 1024 data points each.
The captured channels are then printed one at a time, in the order that they were added. The first Info message in each channel's data block contains the channel number, the setting name, and the axis that it is captured on:

```
#01 0 chan 1 pos axis 1
```

In this case, channel 1 contains the `pos` setting values captured on axis 1.
Following the channel description, the captured data is sent with one data point per Info message, each preceded by "data":

```
#01 0 data 14058
#01 0 data 14059
#01 0 data 14059
#01 0 data 14060
#01 0 data 14061
...
```

In this case, the first data point was 14058, followed by 14059, 14059, 14060, and 14061. The remaining data is sent in the same manner.
The data block for channel 2 is printed immediately following the data for the first channel, starting with its description:

```
#01 0 chan 2 system.voltage
#01 0 data 48.143
#01 0 data 48.071
...
```

In this case, channel 2 contains `system.voltage` and, as a device-scope setting, axis information is omitted from the channel description line. Any remaining channels are printed in the same manner.

scope start

Starts a scope capture

Parameters

None

Scope

Device

Access Level

Normal

Starts a scope capture. Capturing begins as soon as the command is received and the delay specified by `scope.delay` has elapsed. After the delay, data is captured at a regular rate as specified by `scope.timebase`.
If the `scope start` command is sent and there are no settings assigned to any scope channels, the command is rejected with `BADDATA`.
If a scope capture is already in progress, this command is rejected with `STATUSBUSY`.

`scope stop`

Stops a scope capture

Parameters

None

Scope

Device

Access Level

Normal

Stops an on-going scope capture immediately. This can be used to abort a time-consuming scope capture due to either a long delay (`scope.delay`) or slow timebase (`scope.timebase`). Use the `scope print` command to print the data captured up to the time the stop command was received.

`set <setting> <value>`

Sets the value of a device or axis setting

Parameters

`<setting>`

The name of the setting to modify

`<value>`

The desired new value of the setting

Scope

Device and Axis

Access Level

Normal

Sets *`<setting>`* to *`<value>`*.

See the Setting Reference for a detailed list of settings and what they do.

Example: Writing to a setting

```
/set knob.enable 1↵
@01 0 OK IDLE -- 0
```

The setting was successfully configured.

Example: Writing an invalid value

```
/set knob.enable 7↵
@01 0 RJ IDLE -- BADDATA
```

Example: Writing to a read-only setting

Read-only settings accept `get` commands but reject `set` commands.

```
/get system.voltage↵
@01 0 OK IDLE -- 0
/set system.voltage 48.412↵
@01 0 RJ IDLE -- BADCOMMAND
```

stop

Decelerates an axis and brings it to a halt

Parameters

None

Scope

Axis

Access Level

Normal

Decelerates the axis to a halt. The axis decelerates at the rate defined by `motion.decelonly` unless the axis is already stopping; if the axis is already stopping, the axis attempts to stop instantly.

Warning: Stopping instantly may result in damage to the product and reduced lifespan. Use the instant stop feature with extreme caution if the axis is under heavy load. Under heavy load, attempting to stop instantly may cause the driver to shut off and the stage to continue moving in an uncontrolled manner.

Warning: The instant stop feature should not be relied on in applications where stopping the device is time-critical or safety-critical.

Example: Stopping a device

```
/stop↵
@01 0 OK BUSY -- 0
```

storage

These commands provide an interface to store, read, and manage custom data.

Data is stored using a key-value mechanism, which assigns data to a unique key that can be used later to retrieve the data. In many programming languages, data structures that provide this kind of behaviour may be referred to as dictionaries or hash tables. Use

the `storage set` and `storage append` commands to store data, and the `storage get` and `storage print` commands to retrieve data.

```
/storage set name gantry A↵
@01 0 OK IDLE -- 0
/storage get name↵
@01 0 OK IDLE -- gantry A
```

Like settings, keys are associated with either the device or its axes. By default, storage commands operate on device-scope keys. The **`axis`** and **`all`** keywords are used (where applicable) to operate on axis-scope keys and all keys, respectively.

```
/1 1 storage axis set name x-axis↵
@01 1 OK IDLE -- 0
/1 2 storage axis set name y-axis↵
@01 2 OK IDLE -- 0
/storage all print↵
@01 0 OK IDLE -- 0
#01 0 set name gantry A
#01 1 set name x-axis
#01 2 set name y-axis
```

These keywords are distinct from and supplementary to the command's axis number. This allows for operating on either a device-scope key, or an axis-scope key on all axes, which are both done with an axis number of 0.
Data stored with this feature is persistent in the face of power loss, `system reset`, and `system restore`. To manually clear keys use the `storage erase` command.
The device does not reference the stored data and the data's meaning is entirely user-defined. Furthermore, stored data cannot be accessed outside of the `storage` commands. To store custom integer-based data as a setting, which can be accessed via other commands (e.g. `triggers`), see `user` settings.

Key Etiquette

A key can be any value that does not contain reserved characters and is at most `comm.word.size.max` characters. However, in order to avoid naming conflicts with keys that other people or software may create, it is recommended to add a unique prefix to your keys. For instance, keys used by Bosch Rexroth's software have a "Bosch Rexroth." prefix.

Avoid modifying or erasing keys that are created by other people or software. Doing so can cause software that depends on those keys to misbehave. Nevertheless, authors of software that use this feature should not completely depend upon others to not modify or erase their keys. Software should still handle corrupt or missing keys.

`storage [axis] append <key> <value>...`

Appends data to a key

Parameters

`axis`

The key is axis-scope (optional)

By default the key is device-scope

`<key>`

The key to append data to

Must be at most `comm.word.size.max` characters

`<value>`

One or more space-delimited words

Each word must be at most `comm.word.size.max` characters

Scope

Device or Axis

Access Level

Normal

Appends *`<value>`* to the current value of *`<key>`*. If the key does not exist, it is created.

By default the *`<key>`* is device-scope, unless the **`axis`** parameter is specified.

To create or overwrite a key, use the `storage set` command.

As with all ASCII messages:

- The packet must not exceed `comm.packet.size.max` characters.
- Each word must not exceed `comm.word.size.max` characters.
- Each word cannot contain any reserved characters.
- Leading and trailing whitespace is discarded and adjacent whitespace is merged.

To store more values than can fit in a single packet either use Line Continuations or split the values across multiple `storage append` commands. To preserve whitespace or use reserved characters, encode the relevant words using URL-safe base64 encoding before sending them to the device.

Introduced in 7.26

Example: Appending data to a key

Append two dates to the preexisting device-scope key `myorg.maintenance.history`:

```
/storage append myorg.maintenance.history 2022-01-15 2022-01-30↵
@01 0 OK IDLE -- 0
```

`storage [axis|all] erase [<key>]`

Erases keys

Parameters

axis

The key is axis-scope (optional)

By default the key is device-scope

all

The key is both device- and axis-scope (optional)

By default the key is device-scope

<key>

The key to erase (optional)

Must be at most comm.word.size.max characters

Scope

Device and/or Axis

Access Level

Normal

Erases all matching keys in the specified scope. If *<key>* is not specified, the command erases all keys in the specified scope. If *<key>* is specified but does not exist anywhere in the specified scope, the command is rejected with the reason BADDATA.

The **axis** and **all** parameters define the scope of the keys to erase:

- If neither is specified, matching device-scope keys are erased.
- If the **axis** parameter is specified, matching axis-scope keys on the specified axes are erased.
- If the **all** parameter is specified, matching device- and axis-scope keys are erased. In this case the axis number must be 0 or omitted, otherwise the command is rejected with the reason DEVICEONLY.

Introduced in 7.26

Example: Erasing an axis-scope key

Erase the key myorg.axis.name on axis 1:

```
/01 1 storage axis erase myorg.axis.name↵
@01 1 OK IDLE -- 0
```

This example assumes the key myorg.axis.name was previously defined using storage set or storage append.

storage [axis] get <key>

Returns the current value of a key

Parameters

axis

The key is axis-scope (optional)

By default the key is device-scope

<key>

The name of a key

Must be at most `comm.word.size.max` characters

Scope

Device or Axis

Access Level

Normal

Returns the current value of *<key>* in the specified scope. If the key does not exist in the specified scope, the command is rejected with the reason `BADDATA`.
To print all keys in a given scope, use `storage print`.
The key must be at most `comm.word.size.max` characters and cannot contain reserved characters.
If the **`axis`** parameter is not specified, the axis number must be omitted or 0, otherwise the command will be rejected with the reason `DEVICEONLY`.

Introduced in 7.26

Example: Reading the value of a key

Read the value of the key `myorg.device.name`, which is `my device` in this example:

```
/1 0 storage get myorg.device.name↵
@01 0 OK IDLE -- my device
```

storage [axis|all] print

Prints keys and their current values

Parameters

`axis`

The key is axis-scope (optional)

By default the key is device-scope

`all`

The key is both device- and axis-scope (optional)

By default the key is device-scope

Scope

Device and/or Axis

Access Level

Normal

Prints all keys and their associated values in the specified scope. Each key and its associated values are returned, line by line, as separate Info messages. Each message is formatted as a `storage set` command that can be used to recreate the key, but with the `storage` and `axis` words omitted.

To read the value of a single key, use the `storage get` command.

The **`axis`** and **`all`** parameters define the scope of the keys to print:

- If neither is specified, all device-scope keys are printed.
- If the **`axis`** parameter is specified, all axis-scope keys on the specified axes are printed.
- If the **`all`** parameter is specified, all device- and axis-scope keys are printed. In this case the axis number must be 0 or omitted, otherwise the command is rejected with the reason `DEVICEONLY`.

Note: The order that keys are printed is arbitrary.

Introduced in 7.26

Example: Printing all keys

Print all keys on device 1:

```
/1 storage all print
@01 0 OK IDLE -- 0
#01 0 set myorg.device.name my device
#01 1 set myorg.axis.name x axis
```

The device has two keys:

- the device-scope key `myorg.device.name` with the value `my device`
- the axis-scope key `myorg.axis.name` with the value `x axis`

`storage [axis] set <key> <value>...`

Sets the value of a key

Parameters

`axis`

The key is axis-scope (optional)

By default the key is device-scope

`<key>`

The key to create or overwrite

Must be at most `comm.word.size.max` characters

`<value>`

One or more space-delimited words

Each word must be at most `comm.word.size.max` characters

Scope

Device and Axis

Access Level

Normal

Sets *<key>* to *<value>* in the specified scope. If the key already exists, the value is overwritten. By default the *<key>* is device-scope, unless the **axis** parameter is specified.

To append data to an existing key, use the `storage append` command.

As with all ASCII messages:

- The packet must not exceed `comm.packet.size.max` characters.
- Each word must not exceed `comm.word.size.max` characters.
- Each word cannot contain any reserved characters.
- Leading and trailing whitespace is discarded and adjacent whitespace is merged.

To store more values than can fit in a single packet either use Line Continuations or split the values across multiple `storage append` commands. To preserve whitespace or use reserved characters, encode the relevant words using URL-safe base64 encoding before sending them to the device.

Introduced in 7.26

Example: Setting a device-scope key

Set the device-scope key `myorg.device.name` to `my device`:

```
/01 0 storage set myorg.device.name my-device
@01 0 OK IDLE -- 0
```

stream

These commands create queued sequential actions for use in geometric path following

A stream is a series of queued actions that will execute sequentially. Motion actions in streams use geometric shapes, such as line segments, which combine in sequence to create a path for the device to follow. Unlike normal `move` commands, `stream` commands append to, rather than preempt, existing motion. When executing a stream, the axis does not slow down between segments (unless otherwise specified) and will transition smoothly between each path segment (within acceleration constraints).

To start using a stream, initiate it using either `stream setup live` or `stream setup store`. A stream initiated with `stream setup live` is in ***live mode*** and will cause the specified axes to perform the subsequent stream actions sent to it. A live stream can only be initiated if the specified axes have a position reference, and if the device status of all specified axes are `IDLE`. A stream initiated with `stream setup store` is in ***store mode*** and stores the subsequent stream actions to the specified

memory location (called a buffer), which can later be played back on a live stream using `stream call`.
To check the status of a stream, use `stream info`. To check what actions are stored in a particular buffer, use `stream buffer print`. To clear all actions from a particular buffer, use `stream buffer erase`. End a stream using `stream setup disable`, or by sending a command that would otherwise pre-empt the stream (such as `stop`). When ending a live stream, make sure motion is complete and the device status is `IDLE` before ending the stream, or else motion will end prematurely.
Stream actions that can be queued include:

- moving along a line (`stream line`)
- waiting for a length of time (`stream wait`)
- changing speed or acceleration (`stream set`)
- waiting for an IO to meet a condition (`stream wait io`)
- changing a digital output (`stream io set do`)
- calling a stored buffer (`stream call`)

In a live stream, it is important to manage the queue of actions. If the queue becomes empty when the device is moving, the movement will complete but decelerate to a stop at that point; the transition to a subsequent movement may not be smooth. In this case, the ***Stream Discontinuity*** (`ND`) warning flag will be raised to indicate a discontinuity in the motion. To help avoid this condition at the beginning of a stream, `stream fifo` can be used to fill the queue before actions start executing. If the queue becomes full, any subsequent commands to add actions will be rejected with the reason `AGAIN`. In this case, the command should be re-sent until the queue has space and the command is no longer rejected.

Example: Moving between two points

Starting from position zero, move to 10,000 and then back:

```
/stream 1 setup live 1↵
@01 0 OK IDLE -- 0
/stream 1 line abs 10000↵
@01 0 OK BUSY -- 0
/stream 1 line abs 0↵
@01 0 OK BUSY -- 0
```

stream buffer <buf> erase

Erases the stream actions stored in a buffer

Parameters

<buf>

The number of the buffer to erase

Must be in the range 1 to `stream.numbufs`

Scope

Device

Access Level

Normal

Erases the contents of the buffer specified by *<buf>*. After erasure, the buffer can no longer be called until it is filled with new actions using `stream setup store`.
Note: This command can be used even if *<buf>* has been called and is not yet finished playing back; in this case, playback completes normally. This command cannot be used if a stream is currently storing to *<buf>*; in this case, send `stream setup disable` first, then erase the buffer.

Introduced in 7.05

Example: Erasing a buffer

Erase buffer 1:

```
/stream buffer 1 erase↵
@01 0 OK IDLE -- 0
```

stream buffer <buf> print

Returns the stored stream actions

Parameters

<buf>

The number of the buffer to print

Must be in the range 1 to `stream.numbufs`

Scope

Device

Access Level

Normal

Prints the stream actions stored in the buffer specified by *<buf>*. The commands used to create each action are returned, line by line, as separate Info messages. Output begins with the `stream setup store` command and ends with the `stream setup disable` command. The `stream` and the stream number prefixes from the original commands are omitted.
Note: This command can be used while the stream is being written to print out all the stream actions stored so far. It can also be used after writing is finished, including while the buffer is being called, to print out the whole buffer. Trying to print out a buffer that does not exist will fail.

Note: Some commands may not be displayed in exactly the same format they were originally entered, however they are displayed in a functionally equivalent form.

Introduced in 7.05

Example: Printing a buffer

Print buffer 1, which contains three actions:

```
/stream buffer 1 print↵
@01 0 OK IDLE -- 0
#01 0 setup store 1 1
#01 0 line abs 10000
#01 0 wait 1000
#01 0 line abs 20000
#01 0 setup disable
```

stream <stream> call <buffer>

Queues the stream actions stored in a buffer

Parameters

<stream>

The number of the stream to add the actions to

Must be in the range 1 to `stream.numstreams`

<buffer>

The number of the buffer to call

Must be in the range 1 to `stream.numbufs`

Scope

Device

Access Level

Normal

Adds all of the stream actions stored in the buffer specified by *<buffer>* to the stream specified by *<stream>*.

Calling a buffer from a stream in live mode (i.e. created with `stream setup live`) behaves the same as sending all the individual stream actions in the buffer, one by one. For example, if a line segment in the buffer uses absolute coordinates, that segment will end at the absolute coordinates recorded in the buffer. However, if a line segment in the buffer uses relative coordinates, the relative coordinates recorded in the buffer are interpreted relative to the position before the stream actions were added. The buffer must have already been created using `stream setup store`, and must have been created with the same number of axes as are in the stream specified by *<stream>*.

A buffer can also be called from a stream in store mode (i.e. created with `stream setup store`). However, in that case, the stored `stream call` action only acts as a reference to the buffer; its contents are not stored. When the buffer containing the `stream call` command is later called from a stream in live mode, the contents of the referenced *<buffer>* at that time will be queued. While nesting calls to buffers like this is a supported usage, nesting too many `stream call` actions can overflow the live stream's action queue. This will cause an error; the ***Stream Discontinuity*** warning flag will become active, and the `stream info` command will return `stackdepth` as the error reason. A buffer that calls itself is a special case, and will loop indefinitely without generating an error.

Introduced in 7.05

Example: Running a recorded buffer at three different locations

Move the axis to three different locations (0, 10000, and 20000) and play back the commands recorded in buffer 1 at each location:

```
/stream 1 line abs 0↵
@01 0 OK BUSY -- 0
/stream 1 call 1↵
@01 0 OK BUSY -- 0
/stream 1 line abs 10000↵
@01 0 OK BUSY -- 0
/stream 1 call 1↵
@01 0 OK BUSY -- 0
/stream 1 line abs 20000↵
@01 0 OK BUSY -- 0
/stream 1 call 1↵
@01 0 OK BUSY -- 0
```

stream *<stream>* fifo cork|uncork

Controls execution of queued actions at the beginning of a stream

Parameters

<stream>

The number of the stream to manipulate

Must be in the range 1 to `stream.numstreams`

cork

Queues stream actions without executing them

uncork

Begins execution of queued stream actions

Scope

Device

Access Level

Normal

Allows a stream in live mode to build up a queue of actions by preventing them from executing. The `stream fifo cork` command will prevent subsequent actions from executing. They will begin executing once the queue has become full, or the `stream fifo uncork` command is sent.

Normally, a stream in live mode will execute actions as soon as the previous action has completed; if the queue is empty, the next stream action will execute immediately. If the stream begins with a number of very short path segments, the commands to add subsequent actions may not be sent fast enough to keep up with the device's motion; in this case, the queue empties and the device slows down as it approaches the end of the path it has seen so far. In some cases, this deceleration may be undesirable; for example, some applications may require that the entire path be traversed at constant speed. This is the situation where `stream fifo cork` is useful. It allows a larger number of actions to be queued before they start executing, reducing the likelihood of running out of path while moving.

The `stream fifo uncork` command can be sent at any time during a stream in live mode. If the device has already started moving because it has received enough actions, the command will do nothing. The `stream fifo cork` command is only accepted for a stream in live mode when the stream's axes are `IDLE`.

Introduced in 7.05

Example: Moving to two points quickly

Move to 10,000, then back to 0, being careful not to delay at 10,000:

```
/stream 1 setup live 1↵
@01 0 OK IDLE -- 0
/stream 1 fifo cork↵
@01 0 OK IDLE -- 0
/stream 1 line abs 10000↵
@01 0 OK IDLE -- 0
/stream 1 line abs 0↵
@01 0 OK IDLE -- 0
/stream 1 fifo uncork↵
@01 0 OK BUSY -- 0
```

Even if it takes a long time to send the second `stream line` command, motion will not start until the `stream fifo uncork` command is sent. This guarantees that the device will not stop for a long time at 10,000.

Example: Moving with a digital output

Move from 0 to 10,000, with digital output port 1 high only while moving:

```
/stream 1 setup live 1↵
@01 0 OK IDLE -- 0
/stream 1 fifo cork↵
@01 0 OK IDLE -- 0
/stream 1 io set do 1 1↵
@01 0 OK IDLE -- 0
/stream 1 line abs 10000↵
@01 0 OK IDLE -- 0
/stream 1 io set do 1 0↵
@01 0 OK IDLE -- 0
/stream 1 fifo uncork↵
@01 0 OK BUSY -- 0
```

Without the `stream fifo cork` command, digital output port 1 would go high as soon as the first `stream io set do` command was sent. With the `stream fifo cork` command, digital output port 1 will be high only while the device is moving.

stream <stream> info

Returns information about a stream

Parameters

<stream>

The number of the stream to examine

Must be in the range 1 to `stream.numstreams`

Scope

Device

Access Level

Normal

Returns information about the stream specified by *<stream>*. The reply contains a list of the following values, separated by spaces:

1. the stream mode, one of (`disabled`, `live`, or `store`)
2. the number of axes, or - if the stream mode is `disabled`
3. the maximum centripetal acceleration, or - if the stream mode is not `live`
4. the maximum tangential acceleration, or - if the stream mode is not `live`
5. the maximum speed, or - if the stream mode is not `live`
6. an error reason indicating why ***Stream Bounds Error*** (`FB`) was set, or - if ***Stream Bounds Error*** (`FB`) was not set on this stream

The possible error reasons are:

`axislimit`

A path segment tried to move an axis to a position less than `limit.min` or greater than `limit.max`.

`setting`

A `stream set` action had an out-of-range value.

stackdepth
: The stream's action queue overflowed from too many nested `stream call` actions (i.e. a stream buffer contained a `stream call` action, which called another buffer containing a `stream call` action, etc.)

bufempty
: A `stream call` action called a buffer which is empty.

bufaxes
: A `stream call` action called a buffer which was recorded for a different number of axes than are driven by the stream.

Introduced in 7.05

Example: Reading information about a disabled stream

Show information about stream 1, which is disabled:

```
/stream 1 info↵
@01 0 OK IDLE -- disabled - - - - -
```

Example: Reading information about a stream in live mode

Show information about stream 1, which is in live mode with one axis and recently tried to call an empty buffer:

```
/stream 1 info↵
@01 0 OK IDLE -- live 1 205 205 153600 bufempty
```

Example: Reading information about a stream in store mode

Show information about stream 1, which is in store mode writing a one-axis buffer:

```
/stream 1 info↵
@01 0 OK IDLE -- store 1 - - - -
```

stream *<stream>* io set do *<channel>*|port *<action>*...

Sets the state of digital output channels in a stream

Parameters

<stream>

The number of the stream to add the action to

Must be in the range 1 to `stream.numstreams`

<channel>

The number of the output channel to modify

Must be in the range 1 to the value of `io info` for the `do` channel type

<action>

The action can be one of `0` (low), `1` (high), `t` (toggle), or `k` (keep).

If a channel is specified, then provide a single action to perform on the channel.

If the port parameter is provided, then provide a space-separated list of actions, one for each channel in the port.

Scope

Device

Access Level

Normal

Sets the state of digital output channels during a stream. A `stream io set do` action will not interrupt motion.

Introduced in 7.07

Example: Setting an output channel at an exact position

Move the device from its initial position of 0 to position 20,000 with digital output channel 1 becoming high as the device passes position 10,000 without slowing down:

```
/stream 1 line abs 10000↵
@01 0 OK BUSY -- 0
/stream 1 io set do 1 1↵
@01 0 OK BUSY -- 0
/stream 1 line abs 20000↵
@01 0 OK BUSY -- 0
```

Example: Setting an output port at an exact position

Move the device from its initial position of 0 to position 20,000 with digital output channel 1 becoming high, 2 becoming low, 3 toggling, and 4 retaining its state as the device passes position 10,000 without slowing down:

```
/io get do↵
/@01 0 OK BUSY -- 1 1 1 1
/stream 1 line abs 10000↵
@01 0 OK BUSY -- 0
/stream 1 io set do port 1 0 t k↵
@01 0 OK BUSY -- 0
/stream 1 line abs 20000↵
@01 0 OK BUSY -- 0
/io get do↵
/@01 0 OK BUSY -- 1 0 0 1
```

stream *<stream>* [on *<axis>*] line abs|rel *<value>* [join maxspeed *<speed>*]

Creates a straight line path segment

Parameters

<stream>

The number of the stream to add the action to

Must be in the range 1 to `stream.numstreams`

<axis>

The stream axis (a) on which to execute the line segment (optional)

The first stream axis is labelled "a", the second "b", and so on

abs

Indicates that *<value>* is an absolute position

rel

Indicates that *<value>* is a position relative to the end point of the previous path segment

<value>

The position to move to

The resulting coordinate must be in the range `limit.min` to `limit.max`

<speed>

The speed limit for the transition between the previous segment and this one (optional)

Must be in the range 0 to `resolution` × 16,384 for stepper motor products

Scope

Device

Access Level

Normal

Appends a straight line segment to the stream specified by *<stream>*. The start of the line segment will be the end of the previous segment (or the location where the stream was initiated for the first segment), and the end point of the line segment is specified by *<value>*. The device will move along the line segment from the start point to the end point.

For firmware versions 7.26 and above, the `on <axis>` syntax is supported on single-axis streams to make the command structure consistent with multi-axis streams, where line segments could be executed on a subset of the axes that the stream was set up with. It does not change the behaviour of the line segment and is simply a convenience.

For firmware versions 7.28 and above, the `join maxspeed <speed>` syntax allows a user to specify a momentary speed constraint at the start of the segment it is included on. Other speed constraints imposed on each of the joined segments also apply.

When `join maxspeed` is not specified, a default transition limit is calculated automatically. Specifying `join maxspeed` can either be used to cause a slower than default transition (down to using a value of 0 to cause a momentary dwell at the transition), or to cause a faster than default transition (possibly at the risk of requiring a higher acceleration than the device is capable of).

Introduced in 7.05

Example: Appending a one-axis line

Add a one-axis line segment to the path. The line ends at 1234:

```
/stream 1 line abs 1234↵
@01 0 OK BUSY -- 0
```

This example assumes the stream was previously set up on an axis using `stream setup live` or `stream setup store`.

Example: Appending a one-axis line to a stream on a specific axis

Add a one-axis line segment to a stream. The line ends at 1234.:

```
/stream 1 on a line abs 1234↵
@01 0 OK BUSY -- 0
```

This example assumes the stream was previously set up with at least one axis using `stream setup live` or `stream setup store`.

Example: Specifying the speed limit at the transition between two line segments

Add a one-axis line segment to the path. The line ends at 1234:

```
/stream 1 line abs 1234↵
@01 0 OK BUSY -- 0
```

Add another one-axis line segment to the path, specifying the maximum join speed to 0. The line ends at 2468:

```
/stream 1 line abs 2468 join maxspeed 0↵
@01 0 OK BUSY -- 0
```

This example assumes the stream was previously set up on an axis using `stream setup live` or `stream setup store`.

`stream <stream> set <setting> <value>`

Sets a speed or acceleration limit

Parameters

<stream>

The number of the stream to add the action to

Must be in the range 1 to `stream.numstreams`

<setting>

The stream setting to set

Must be one of `maxspeed`, `tanaccel`

<value>

The value to change the specified setting to

Must be in the range 1 to `resolution` × 16,384 for the `maxspeed` setting for stepper motor products

Must be in the range 0 to 2,147,483,647 for the `tanaccel` setting

Scope

Device

Access Level

Normal

Specifies a speed or acceleration limit that subsequent path segments will follow. This is a stream action and will be applied in order with other stream actions; it will not have an immediate affect on the active motion.

Note: These settings only relate to the stream, and are separate from the `maxspeed` and `accel` settings. The settings described in this section are volatile, persist only for as long as the stream is enabled, and apply only to motions within the stream.

The `maxspeed` setting controls the maximum limit that the speed can reach. When `maxspeed` is increased, the device can accelerate after passing the point where it was increased, until the speed reaches the new limit. When `maxspeed` is decreased, the device will slow down before reaching the point where it decreases, in order to obey the new limit. The default `maxspeed` when a live stream is initiated is the device's `maxspeed` setting.

The `tanaccel` setting controls the maximum acceleration and deceleration used for linear segments. The default `tanaccel` when a live stream is initiated is the device's `accel` setting.

Introduced in 7.05

Example: Setting the maximum speed

All path segments sent after this command will be limited to a speed of 1,000:

```
/stream 1 set maxspeed 1000↵
```

```
@01 0 OK BUSY -- 0
```

stream <stream> setup disable

Disables a stream

Parameters

<stream>

The number of the stream to disable

Must be in the range 1 to stream.numstreams

Scope

Device

Access Level

Normal

Disables the stream specified by *<stream>*. If the stream was not enabled, this command does nothing. Once disabled, the stream will no longer accept stream commands until it is re-enabled with stream setup live or stream setup store. If the ***Stream Bounds Error*** (FB) flag is present on the axes in the stream, it is cleared.

Introduced in 7.05

stream <stream> setup live [] <axis>

Enables a stream in live mode

Parameters

<stream>

The number of the stream to enable

Must be in the range 1 to stream.numstreams

<axis>

The axis number to control

Must be in the range 1 to system.axiscount

Scope

Device

Access Level

Normal

Enables a stream, identified by *<stream>*, in live mode. A stream in live mode allows a series of stream actions to be sent to the device which will be queued and executed in sequence, with the next action starting after the previous one has completed.
The *<axis>* parameter specifies the axis that the stream actions will act on.
An *<axis>* must have a device status of IDLE, must not already be part of another

stream, and must have a reference position (i.e. must not have the ***No Reference Position*** (WR) warning flag).

Introduced in 7.05

Example: Setting up a stream in live mode

Set up stream number 1 to drive axis 1:

```
/stream 1 setup live 1↵
@01 0 OK IDLE -- 0
```

stream <stream> setup store <buffer> <axes>

Enables a stream in store mode

Parameters

<stream>

The number of the stream to enable

Must be in the range 1 to stream.numstreams

<buffer>

The number of the buffer to write to

Must be in the range 1 to stream.numbufs

<axes>

The number of axes in the path

Must be in the range 1 to system.axiscount

Scope

Device

Access Level

Normal

Enables a stream, identified by *<stream>*, in store mode. A stream in store mode allows a series of stream actions to be stored on the device, and then later executed using the stream call command in a stream in live mode (i.e. enabled using stream setup live). The stream actions are stored in a numbered location, specified by *<buffer>*. The specified *<buffer>* must not already contain any data, or else it must be emptied using stream buffer erase. The *<axes>* parameter defines the number of axes that the stored stream actions will reference; a stream in live mode can only call buffers that have the matching number of axes.

Introduced in 7.05

Example: Setting up a stream in store mode

Set up stream number 1 to write to buffer number 3 with one axis:

```
/stream 1 setup store 3 1↵
@01 0 OK IDLE -- 0
```

stream <stream> wait <duration>

Maintains a position for a specified period of time

Parameters

<stream>

The number of the stream to add the action to

Must be in the range 1 to `stream.numstreams`

<duration>

The length of the delay, in milliseconds

Must be in the range 1 to 4,294,967,295

Scope

Device

Access Level

Normal

Slows the device to a stop, then holds the position for the time specified by *<duration>* before proceeding with the next action.

Introduced in 7.05

Example: Delaying motion for a specific time

Move the device to position 10,000, wait for one second, and then move to position 20,000:

```
/stream 1 setup live 1↵
@01 0 OK IDLE -- 0
/stream 1 line abs 10000↵
@01 0 OK BUSY -- 0
/stream 1 wait 1000↵
@01 0 OK BUSY -- 0
/stream 1 line abs 20000↵
@01 0 OK BUSY -- 0
```

stream <stream> wait io <type> <channel> <operator> <value>

Maintains a position until an IO meets a condition

Parameters

<stream>

The number of the stream to add the action to

Must be in the range 1 to `stream.numstreams`

<type>

The type of IO channel

Must be one of `ao` (analog output), `ai` (analog input), `do` (digital output), or `di` (digital input).

<channel>

The IO channel number

Must be in the range 1 to the value returned by `io info` for the given channel type

<operator>

The comparison to perform

Must be one of the operators listed below

<value>

The value to compare to

For digital ports, one of `0` or `1`

For analog ports, the voltage, in volts, in the range specified for the device

Scope

Device

Access Level

Normal

Slows the device to a stop, then holds the position until the specified condition is met on an IO channel before proceeding with the next action. The condition is not evaluated until the device has finished stopping, so even if the condition would be met before stopping, the device will always fully stop. For the `ao` channel type, the condition is evaluated with the value the pin is being driven at, which may differ slightly from the actual output.

<operator> must be one of the following operators:

==

equal to

<>

not equal to

<

less than

> greater than

<=

less than or equal to

>=

greater than or equal to

Entering `!=` instead of `<>` for not equal to will fail silently because ! is a reserved character in the ASCII protocol.

Introduced in 7.05

Example: Waiting on a digital input

Move the device to position 10,000, wait until digital input port 1 is high, and then move to position 20,000:

```
/stream 1 setup live 1↵
@01 0 OK IDLE -- 0
/stream 1 line abs 10000↵
@01 0 OK BUSY -- 0
/stream 1 wait io di 1 == 1↵
@01 0 OK BUSY -- 0
/stream 1 line abs 20000↵
@01 0 OK BUSY -- 0
```

system

These commands return the device to a certain state or provide system-wide information.

system errors [clear]

Prints the recorded system errors

Parameters

clear

Clear system errors (optional)

Scope

Device

Access Level

Normal

This command prints any system errors that have occurred as a series of Info messages. If there are no system errors, no Info messages are sent. There should be no system errors during normal operation. If any errors are present, please contact Bosch Rexroth Support.

If the optional parameter **clear** is provided, all errors are cleared after printing them.

system reset

Resets the device to the power-up state

Parameters

None

Scope

Device

Access Level

Normal

This command resets the device to the power-up state. After receiving this command and replying, the device waits until no other chained devices have communicated for 200 ms before performing the reset. Any commands sent to the device during this 200 ms period are discarded with no reply sent. Once the device begins performing the reset, it will be unresponsive for a few seconds as it powers up.

Example: Resetting a device

```
/1 system reset↵
@01 0 OK IDLE -- 0
```

system restore

Restores the device to a default state

Parameters

None

Scope

Device

Access Level

Normal

Restores the system to a default state, which includes:

- restoring most axis-scope settings to their factory values
- restoring most device-scope settings to their factory values
- disabling and clearing all `triggers`

Communication settings (see `comm`) are not modified.

Example: Restoring settings on a device

```
/1 system restore↵
@01 0 OK IDLE -- 0
```

tools

These commands perform miscellaneous functions that are not expected to be commonly used, but are useful for specific cases.

tools echo [*<message>*]

Echoes the provided message back to the user

Parameters

<message>

Data to echo (optional)

Scope

Device

Access Level

Normal

The device replies with a data value of *<message>*, with the exception that consecutive spaces are treated as a single space. If any reserved characters are included in *<message>*, the device may not reply, or, in the case of CR and LF, reply with a truncated *<message>*.

Example: Echoing a message

```
/tools echo hello↵
@01 0 OK IDLE -- hello
```

tools findrange

Sets the valid range of the axis using the home and away sensors

Parameters

None

Scope

Axis

Access Level

Normal

This command can be used instead of the home command on an axis equipped with both home and away limit sensors. It first homes the axis and sets the current position to limit.home.preset. The axis then moves to the away sensor and, once it is triggered, sets limit.max to the current position.

This command is equivalent to the following set of commands issued in order:

```
home
tools gotolimit away pos 1 1
```

Note: On power-up, issue this command to obtain a reference position and valid travel range. Otherwise, motion commands may respond with a rejection reply or behave unexpectedly.

Introduced in 7.07

tools gotolimit *<sensor> <direction> <action> <update>*

Moves the axis to a limit sensor and performs the specified action

Parameters

<sensor>

The sensor to move to

One of:

- `home`
- `away`

<direction>

The direction to move towards the sensor

`pos` for the positive direction or `neg` for the negative direction

<action>

The action to perform when the sensor is triggered

Any valid value for the `limit.<sensor>.action` setting
(e.g. `limit.home.action`), except 0, 3, or 5

<update>

How to update the sensor position setting

Any valid value for the `limit.<sensor>.posupdate` setting
(e.g. `limit.home.posupdate`)

Scope

Axis

Access Level

Normal

This command moves the axis in the specified *<direction>* and waits for the sensor to trigger. The axis then aligns itself to the edge of the sensor and performs the limit switch action and position update specified by *<action>* and *<update>*.
The `limit.<sensor>.triggered` setting (e.g. `limit.home.triggered`) is set to 1.
If *<action>* is 2 or 4, the ***No Reference Position*** (`WR`) warning flag is cleared. If *<action>* is 2, the axis will move the to the specified offset position.

Introduced in 7.07

Example: Going home and resetting the current position

The command below is very similar to the `home` command: the axis travels in the negative direction towards the home sensor and updates `pos` when it arrives. It differs in that, if the axis is already on the negative side of the home sensor, the command will fail. This is not the case with homing on some products. The axis may try moving in both directions when searching for the sensor during homing.

```
/tools gotolimit home neg 2 0↵
@01 0 OK BUSY -- 0
```

Example: Adjusting the effective travel range on products with away sensors

The command below moves the axis in the positive direction towards the away sensor, aligns with its edge, and then updates `limit.max` to `pos`. It requires that a reference position is already established and will not update `pos` when the away sensor is triggered.

```
/tools gotolimit away pos 1 1↵
@01 0 OK BUSY -- 0
```

tools parking park|unpark

Parks or unparks the axis

Parameters

park
Park the axis

unpark
Unpark the axis

Scope

Axis

Access Level

Normal

Parking allows the device to be turned off and then used at a later time without first having to `home` the axes. A parked axis rejects any movement command with a `PARKED` response, except for `home`, which homes the axis and clears the parked state. For firmware versions 7.15 and above the blue LED fades in and out while the axis is parked. Motor drivers remain enabled and `hold current` continues to be applied until the device is powered off. To restore a parked device, power it on and issue the `tools parking unpark` command. To determine whether an axis is parked or not, read the `parking.state` setting.

Note: Do not remove power when there is a load on the device that could cause it to slip when the motor hold current is turned off. Unparking an axis that moved while it was parked, including while the device was turned off, will result in an invalid axis state. If this occurs, home the axis before continuing to use the device.

Note: A busy axis will reject the **park** command.

Example: Parking all axes

Park all axes on a device:

```
/tools parking park↵
@01 0 OK IDLE -- 0
```

Example: Unparking all axes

Unpark all axes on a device:

```
/tools parking unpark↵
@01 0 OK IDLE -- 0
```

tools storepos <number>

Returns a stored position

Parameters

<number>

The number of the stored position to return

Must be in the range 1 to 16

Scope

Axis

Access Level

Normal

Returns a position previously saved by a `tools storepos current` or `tools storepos position` command. The *<number>* parameter specifies which stored position to return.

Note: Each stored position defaults to position 0 until it is explicitly set.

Example: Reading a stored position

Read the value held in stored position number 3 of axis 1 on device 2:

```
/2 1 tools storepos 3↵
@02 1 OK IDLE -- 5678
```

tools storepos <number> current

Stores the current position

Parameters

<number>

The number of the stored position to update

Must be in the range 1 to 16

Scope

Axis

Access Level

Normal

Saves the current position (pos) into the stored position specified by *<number>*. The stored position persists across a reset or power-up.
To store a specific position, other than the current position, use tools storepos position.
To move to a stored position use move stored.

Example: Storing the current position

On each axis, store the current position into stored position number 3:

```
/tools storepos 3 current↵
@01 0 OK IDLE -- 5678
```

tools storepos *<number>* *<position>*

Stores a specific position

Parameters

<number>

The number of the stored position to update

Must be in the range 1 to 16

<position>

The position to store

Must be in the range -1,000,000,000 to 1,000,000,000

Scope

Axis

Access Level

Normal

Saves the specified *<position>* into a stored position specified by *<number>*. The stored position persists after a power-up.
To store the current position (pos) rather than a specific position, use tools storepos current.
To move to a stored position use move stored.
See the Physical Units section for information about unit conversions.

Example: Storing a specific position

Set stored position number 3 to the position 1234:

```
/tools storepos 3 1234↵
@01 0 OK IDLE -- 1234
```

trigger

These commands configure, and return information about, triggers.

A trigger consists of a single condition and two optional actions, labelled `a` and `b`. The actions occur when the condition becomes true, which is referred to as the trigger 'firing'.

Enabling and Disabling Triggers

Each device has multiple triggers available. These are indexed by trigger number starting from 1. Consult the `trigger.numtriggers` setting to find out how many triggers are available.

Trigger conditions are not evaluated while a trigger is disabled; they must be enabled to function. To enable a specific trigger and optionally set how many times it will fire, use the `trigger enable` command. To disable an trigger, use the `trigger disable` command. To find out whether or not a trigger is currently enabled (and how many more times it will fire), use the `trigger show` command.

Trigger Conditions

A trigger's condition can be based on any of the following:

- the value of a setting (see `trigger when setting`)
- a distance interval (see `trigger when dist` and `trigger when encoder dist`)
- a time period (see `trigger when time`)
- the value of an IO channel (see `trigger when io`, available only on devices with input or output channels)

Once a trigger is enabled, it will fire whenever its condition transitions from evaluating as false to evaluating as true. This includes the case where the transition is due to the condition being changed. In addition, if a trigger condition evaluates as true at the time a disabled trigger is enabled, the trigger will fire immediately.

Trigger Actions

Use `trigger action command` to create an action that executes a command, or `trigger action setting` to create an action that changes a setting. To clear an action, use the `trigger action none` command.

A trigger's actions run once each time the trigger fires. Action `a` will begin before action `b`, but action `b` does not wait for action `a` to complete. When a trigger's condition is met, there may be a delay before its actions occur. When multiple triggers have pending actions, there is no guarantee of which trigger's actions will happen first. The delay may increase if many triggers are firing, or if the user is continuously making motion requests

with commands or a manual knob. If a trigger's actions are changed or it is disabled, any pending actions are cancelled.
Triggers (including conditions, actions, and the total number of times the trigger was configured to fire) are non-volatile and will persist after a `system reset` or power-up. An enabled trigger will continue to be enabled on the next power-up, but will not fire immediately if the condition evaluates as true on power-up. Triggers are cleared when the `system restore` command is sent. To read the full trigger configuration, use the `trigger print` command.

Introduced in 7.06

Example: Moving between two positions

Use two triggers to continuously move axis 1 between two positions, 30,000 and 50,000:

First configure the conditions that the triggers will fire under:

```
/trigger 1 when 1 pos >= 50000↵
@01 0 OK IDLE -- 0
/trigger 2 when 1 pos <= 30000↵
@01 0 OK IDLE -- 0
```

Then configure the actions that will occur when the triggers fire:

```
/trigger 1 action a 1 move abs 30000↵
@01 0 OK IDLE -- 0
/trigger 2 action a 1 move abs 50000↵
@01 0 OK IDLE -- 0
```

Finally, enable the triggers:

```
/trigger 1 enable↵
@01 0 OK IDLE -- 0
/trigger 2 enable↵
@01 0 OK IDLE -- 0
```

If the starting position is lower than 30,000 or higher than 50,000, the triggers will start firing as soon as the trigger is enabled. If the position is between 30,000 and 50,000 when the trigger is enabled, nothing will occur; an initial motion command will be required to fire the triggers and start off the sequence of motions.

`trigger <number> action <act> [<axis>] <command>`

Sets a command to be a trigger action

Parameters

<number>

The number of the trigger to set the action for

<act>

The trigger action to set (either `a` or `b`)

<axis>

The axis on which to run the command (optional)

If *<command>* is axis-scope and no *<axis>* is provided, the command will be run on all axes. If *<command>* is device-scope, *<axis>* must be omitted or set to 0.

<command>

The command to execute as the trigger action

Scope

Device

Access Level

Normal

Specifies a command to run as action *<act>* of trigger number *<number>*.
Note: The *<command>* parameter is not validated when `trigger action command` is sent; it will only be validated when the action runs, and the user will not be notified if validation fails at that point.

The *<command>* parameter supports the following:

- `io set do`
- `move abs`
- `move index`
- `move index next|prev`
- `move min|max`
- `move rel`
- `move sin`
- `move sin stop`
- `move stored`
- `move vel`
- `scope start`
- `stop`
- `stream call`

Note: Triggers do not wait for any actions to complete before beginning the next trigger action. To set up a sequence of motions so that the first one finishes before the next one starts, see the `stream` command.

Introduced in 7.06

Example: Moving when a trigger fires

Configure axis 1 of the device to move forward a distance of 10,000, relative to the current position, when trigger 3 fires:

```
/1 trigger 3 action a 1 move rel 10000↵
@01 0 OK IDLE -- 0
```

trigger <number> action [<act>] none

Sets a trigger action to do nothing

Parameters

<number>

The number of the trigger to set the action for

<act>

The trigger action to set (either `a` or `b`) (optional)

Scope

Device

Access Level

Normal

Clears the trigger action on action *<act>* of trigger *<number>*. If no *<act>* parameter is specified, then this command clears all actions on trigger *<number>*. If the trigger fires and the specified action is set to none, nothing will happen.

When configuring a trigger, if only one action is needed it is good practice to set the other action to none, so that a previously-configured action isn't executed accidentally.

Introduced in 7.06

trigger <number> action <act> [<axis>] <setting> <operator> <value>

Sets a trigger action to update a setting

Parameters

<number>

The number of the trigger to set the action for

<act>

The trigger action to set (either `a` or `b`)

<axis>

The axis on which to change the setting (optional)

If *<setting>* is axis-scope and no *<axis>* is provided, the setting change will occur on all axes. If *<setting>* is device-scope, *<axis>* must be omitted or set to 0.

<setting>

The setting to change as the trigger action

<operator>

The operation to perform on the setting

Must be one of

=

to set the setting to *<value>*

+=

to increment the setting by *<value>*

-=

to decrement the setting by *<value>*

<value>

The value to either change the setting to, or to increment/decrement it by

Scope

Device

Access Level

Normal

Specifies a setting to change as action *<act>* of trigger number *<number>*. Any writeable setting listed in the Setting Reference can be configured as a trigger action.
Note: The *<value>* and *<operator>* parameters are not checked to ensure they produce a valid value of *<setting>* when the `trigger action setting` command is sent. When the trigger fires and the trigger action runs, the setting change will not occur if the resulting setting value is outside the valid range of *<setting>*, and the user will not be notified.

On integrated products, the value of `system.access` must be greater or equal to the "Write Access Level" of the specified setting for this command to succeed.
Lowering `system.access` after the trigger action has been configured will still allow the trigger action to run when the trigger fires.
To set up a trigger to update a setting with the value of another setting, rather than a specific value, use the command.

Introduced in 7.06

Example: Increasing the speed when a trigger fires

Configure axis 1 of the device to increase the `maxspeed` setting by 10,000 as the second action when trigger 3 fires.

```
/1 trigger 3 action b 1 maxspeed += 10000↵
@01 0 OK IDLE -- 0
```

Example: Attempting to set a setting to an invalid value when a trigger fires

First, set the maxspeed to 30,000. Configure axis 1 to decrease the maxspeed setting by 50,000 when the axis passes position 10,000. Enable the trigger to fire 5 times:

```
/1 1 set maxspeed 30000↵
@01 0 OK IDLE -- 0
/1 trigger 1 when pos >= 10000↵
@01 0 OK IDLE -- 0
/1 trigger 1 action a 1 maxspeed -= 50000↵
@01 0 OK IDLE -- 0
/1 trigger 1 enable 5↵
@01 0 OK IDLE -- 0
```

Move to position 10,000 to fire the trigger. The trigger will fire, but the action will not be executed, since maxspeed cannot be set to a negative number:

```
/1 move abs 10000
@01 0 OK BUSY -- 0
/1 get maxspeed↵
@01 0 OK IDLE -- 30000
/1 trigger show↵
@01 0 OK IDLE -- 4 d d d d d
```

The enable count has decreased from 5 to 4, indicating that the trigger fired, but maxspeed was not changed.

trigger <number> disable

Disables the specified trigger

Parameters

<number>

The number of the trigger to disable

Scope

Device

Access Level

Normal

Disables the trigger specified by *<number>*. Once disabled, the trigger will not fire and trigger actions will not run, even if the trigger conditions are met. The trigger can be re-enabled with trigger enable.

Introduced in 7.06

trigger <number> enable [<count>]

Enables the specified trigger

Parameters

<number>

The number of the trigger to enable

<count>

The number of times the trigger fires before disabling itself (optional)

Must be in the range 0 to 4,294,967,294

Scope

Device

Access Level

Normal

Enables the trigger specified by *<number>*. Once a trigger is enabled, it will fire whenever its condition transitions from evaluating as false to evaluating as true. If a trigger condition is true when a disabled trigger is enabled, the trigger will fire immediately.

If the count parameter is specified, the trigger will disable itself after it fires *<count>* times. The count decreases each time the trigger fires, even if the trigger action fails validation and cannot run. Use the `trigger show` command to see how many more times the trigger will fire before disabling.

Note: A `system reset` or power-up restores the enable status and count value to the configuration specified in the most recent `trigger enable` or `trigger disable` command. Use the `trigger print` command to print the power-up configuration of the trigger. Use the `trigger show` command to view the current trigger state.

The trigger can be disabled with `trigger disable`.

Introduced in 7.06

`trigger [<number>] print`

Prints the trigger configuration

Parameters

<number>

The number of the trigger to print (optional)

Scope

Device

Access Level

Normal

Prints the configuration for the trigger specified by *<number>* as a series of Info messages. If *<number>* is not provided, then the configuration of every trigger on the device will be printed. To replicate a trigger's configuration, for each line in the info messages that this command returns, send *<trigger>* *<number>* followed by the text that the info message contains.

This command returns information about the original trigger configuration, not the current status of each trigger. To read the current status, use the `trigger show` command.

Introduced in 7.06

Example: Printing the contents of a trigger

Using trigger 6, configure axis 1 of the device to move a distance of -3,000 when the axis passes position 10,000. Enable the trigger to only fire 15 times. Print the saved configuration.

```
/1 trigger 6 when 1 pos >= 10000↵
@01 0 OK IDLE -- 0
/1 trigger 6 action a 1 move rel -3000↵
@01 0 OK IDLE -- 0
/1 trigger 6 enable 15↵
@01 0 OK IDLE -- 0
/1 trigger 6 print↵
@01 0 OK IDLE -- 0
#01 0 when 1 pos >= 10000
#01 0 action a 1 move rel -3000
#01 0 action b none
#01 0 enable 15
```

Example: Printing the contents of all triggers

Display the configuration of all triggers. Trigger 6 has been configured as in the previous example.

```
/1 trigger print↵
@01 0 OK IDLE -- 0
#01 0 trigger 1
#01 0 when none
#01 0 action a none
#01 0 action b none
#01 0 disable
#01 0 trigger 2
#01 0 when none
#01 0 action a none
#01 0 action b none
#01 0 disable
...
#01 0 trigger 6
#01 0 when 1 pos >= 10000
#01 0 action a 1 move rel -3000
#01 0 action b none
#01 0 enable 15
```

trigger show

Returns information about the state of the triggers

Parameters

None

Scope

Device

Access Level

Normal

Returns information about the status of each trigger. This command returns a space-separated list of values, one for each trigger. The possible values are:

e

The trigger is enabled.

d

The trigger is disabled.

count

The trigger will fire *count* more times.

This command returns information about the current status of each trigger, not the original trigger configuration. To read the original trigger configuration, use the `trigger print` command.

Introduced in 7.06

Example: Reading the status of each trigger

```
/1 trigger show↵
@01 0 OK IDLE -- e d 500 d d d
```

Trigger 1 is enabled. Trigger 3 is currently enabled, and after being fired 500 more times, will be disabled. Triggers 2, 4, 5, and 6 are disabled.

trigger <number> when [<axis>] encoder dist <interval>

Sets a trigger condition to be based on a measured distance interval

Parameters

<number>

The number of the trigger to set the condition for

<axis>

The axis to monitor (optional)

<interval>

The measured distance between trigger fires, in position units

Must be in the range 1 to 2,000,000,000

Scope

Device

Access Level

Normal

This command sets the condition for the trigger specified by *<number>* such that it evaluates as true when the given *<axis>* has travelled the distance specified by *<interval>*, as indicated by the measured position encoder.pos.

When enabled on axes with an encoder, the trigger will fire whenever the axis' measured position (encoder.pos) has changed by *<interval>* position units from where it last fired or from its starting position if it has not yet fired. Enabling the trigger or reconfiguring the distance interval or axis (by sending trigger when encoder dist) resets the last fired position to the current encoder position. The last fired position is also reset anytime the ***No Reference Position*** (WR) or ***Device Not Homed*** (WH) warning flag is set or cleared, during a home command, and after modifying the pos setting.

Triggers can fire at frequencies up to 10 kHz. If a motion causes a distance trigger to fire faster than 10 kHz, it fires the trigger at the maximum rate of 10 kHz.

The encoder resolution should be taken into account when setting *<interval>*. Setting *<interval>* to a value smaller than the encoder resolution can cause the trigger to fire multiple times when it transitions between encoder counts. Setting *<interval>* close to the encoder resolution may cause the trigger to fire unexpectadely due to transitions on the edges of encoder counts.

This command only applies to integrated products with encoders and to controllers.

Introduced in 7.12

Example: Firing a trigger every time the axis has travelled 100 position units

Configure trigger 3 to fire every time axis 1's measured position (encoder.pos) changes by 100 position units.

```
/1 trigger 3 when 1 encoder dist 100↵
@01 0 OK IDLE -- 0
```

trigger *<number>* when [*<axis>*] dist *<interval>*

Sets a trigger condition to be based on a trajectory distance interval

Parameters

<number>

The number of the trigger to set the condition for

<axis>

The axis to monitor (optional)

<interval>

The trajectory distance between trigger fires, in position units

Must be in the range 1 to 2,000,000,000

Scope

Device

Access Level

Normal

This command sets the condition for the trigger specified by *<number>* such that it evaluates as true when the given *<axis>* has travelled the distance specified by *<interval>*, as indicated by the trajectory position pos.
When enabled, the trigger will fire whenever the axis' pos value has changed by *<interval>* position units from where it last fired or from its starting position if it has not yet fired. Enabling the trigger or reconfiguring the distance interval or axis (by sending trigger when dist) resets the last fired position to the current position. The last fired position is also reset anytime the ***No Reference Position*** (WR) or ***Device Not Homed*** (WH) warning flag is set or cleared, during a home command, and after modifying the pos setting.

Triggers can fire at frequencies up to 10 kHz. If a motion causes a distance trigger to fire faster than 10 kHz, it fires the trigger at the maximum rate of 10 kHz.

Introduced in 7.07

Example: Firing a trigger every time the axis has travelled 100 position units

Configure trigger 3 to fire every time axis 1 has travelled 100 position units, as measured by pos.

```
/1 trigger 3 when 1 dist 100↵
@01 0 OK IDLE -- 0
```

trigger *<number>* when io *<type>* *<channel>* *<operator>* *<value>*

Sets the trigger condition to be based on an IO channel value

Parameters

<number>

The number of the trigger to set the condition for

<type>

The type of IO channel to monitor

Must be one of `ao` (analog output), `ai` (analog input), `do` (digital output), or `di` (digital input).
See the `io` commands section for a description of these types.

<channel>
The IO channel to monitor. See the `io` commands section for a description of these channels.

<operator>
The operator to use to compare the IO channel to the value

<value>

The value to compare the IO channel to

For digital ports, one of `0` or `1`

For analog ports, the voltage, in volts, in the range specified for the device

Scope

Device

Access Level

Normal

This command sets the condition for the trigger specified by *<number>* such that it evaluates as true when a comparison between the provided IO channel (specified by *<type>* and *<channel>*) and the provided *<value>* is true. For the `ao` channel type, the comparison is evaluated with the value the pin is being driven at, which may differ slightly from the actual output.

<operator> must be one of the following operators:

==
equal to

<>
not equal to

<
less than

>
greater than

<=
less than or equal to

>=
greater than or equal to

Entering `!=` instead of `<>` for not equal to will fail silently because ! is a reserved character in the ASCII protocol.

Introduced in 7.07

Example: Firing a trigger when an analog input exceeds a particular voltage

Configure trigger 3 to fire when analog input 2 exceeds 7.5 V.

```
/1 trigger 3 when io ai 2 > 7.5↵
@01 0 OK IDLE -- 0
```

trigger <number> when [<axis>] <setting> <operator> <value>

Sets a trigger condition to be based on a setting value

Parameters

<number>
The number of the trigger to set the condition for

<axis>

The axis on which to monitor the setting (optional)

If *<setting>* is device-scope, *<axis>* must be omitted or set to 0.

<setting>
The setting to monitor

<operator>
The operator to use to compare the setting to the value

<value>
The value to compare the setting to

Scope

Device

Access Level

Normal

This command sets the condition for the trigger specified by *<number>* such that it evaluates as true when a comparison between the provided *<setting>* and the provided *<value>* is true. Any setting listed in the Setting Reference is valid to use as the *<setting>* parameter.

<operator> must be one of the following operators:

==
equal to

<>
not equal to

<
less than

>
greater than

<=
less than or equal to

>=

greater than or equal to

Entering `!=` instead of `<>` for not equal to will fail silently because ! is a reserved character in the ASCII protocol.

Introduced in 7.06

Example: Firing a trigger when the axis moves past a particular position

Configure trigger 3 to fire when the position of axis 1 reaches or exceeds 750,000.

```
/1 trigger 3 when 1 pos >= 750000↵
@01 0 OK IDLE -- 0
```

trigger <number> when time <period>

Sets a trigger condition to be based on a time interval

Parameters

<number>

The number of the trigger to set the condition for

<period>

The time interval between trigger fires, in milliseconds

Must be in the range 0.1 to 429,496,729.5, with resolution of 0.1 ms

Scope

Device

Access Level

Normal

This command sets the condition for the trigger specified by *<number>* such that it evaluates as true each time the time interval specified by *<period>* has elapsed.

The trigger will wait the specified time interval after being enabled to fire the first time, and then fires every time interval after that.

Introduced in 7.07

Example: Firing a trigger every 1000 ms

Configure trigger 3 to fire every 1000 ms.

```
/1 trigger 3 when time 1000↵
@01 0 OK IDLE -- 0
```

warnings [clear]

Returns the active warnings

Parameters

clear

Clear clearable warnings (optional)

Scope

Device and Axis

Access Level

Normal

This command prints out the number of active warnings (formatted as two digits), followed by a space-separated list of all active warning flags, as described in Warning Flags.

If the optional parameter **`clear`** is provided, all clearable warnings are reset. The reply's data field reports all warnings at the time the command was sent, and the warning flag field reports the highest priority warning flag that persists after the warning reset.

Example: Reading all active warnings

Print all active warnings on device 1:

```
/1 warnings↵
@01 0 OK IDLE -- 00
```

No warnings are active on the device.

Example: Reading and clearing all active warnings on an axis

Print and clear all active warnings on axis 1:

```
/1 1 warnings clear↵
@01 0 OK IDLE WR 02 FE WR
```

There are 2 warnings active on axis 1: ***Limit Error*** (`FE`) and ***No Reference Position*** (`WR`).

Limit Error (`FE`) is clearable and has been reset. ***No Reference Position*** (`WR`) is not user-clearable and therefore has not been reset. This can be verified by sending the command again, without the clear parameter:

```
/1 1 warnings↵
@01 0 OK IDLE WR 01 WR
```

Only ***No Reference Position*** (`WR`) remains.

Setting Reference

The following section covers settings available in the ASCII protocol.

All the settings listed below are used with the `get` and `set` commands to read and change their values.

Settings are associated with either the device or its axes, which is referred to as the setting's scope. For device-scope settings, specifying an axis number other than 0 with `get` or `set` results in a `DEVICEONLY` rejection reason:

```
/1 1 get device.id↵
@01 1 RJ IDLE -- DEVICEONLY
```

Configuring a setting with a value outside of its specified range will result in a `BADDATA` reply. Furthermore, if the specified value is outside the valid range for any axis, none of the axes are set and a `BADDATA` reply is returned.

Read-only settings cannot be configured and reply with a `BADCOMMAND` error when the `set` command is used with them.

Settings are either persistent (non-volatile) or volatile. Changes to persistent settings will persist even when power is removed or `system reset` is received. Changes to volatile settings, on the other hand, will be lost when power is removed or `system reset` is received.

For the default values of the settings please refer to Bosch Rexroth Support - Device Settings.

accel

The acceleration and deceleration used when changing speeds

Valid Values

0 to 2,147,483,647

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies the acceleration and deceleration used for most motion trajectories (unless the deceleration has been set separately using `motion.decelonly`).

If the acceleration is changed while the device is moving, the device will immediately apply the new acceleration and continue to move to the target position.

Warning: A value of 0 specifies infinite acceleration, which is often not physically feasible and may cause the motor to stall. It may also result in potential damage to the product and reduced lifespan, especially if the axis is under heavy load.

Writing accel sets both motion.accelonly and motion.decelonly to the provided value. To modify the acceleration or deceleration independently, write motion.accelonly or motion.decelonly directly.
When queried, accel returns the value of motion.accelonly, regardless of the value of motion.decelonly.
See the Physical Units section for information about unit conversions.

cloop

These settings contain controls for closed-loop mode.
In closed-loop mode, the encoder-measured position (encoder.pos) is compared to the target position (pos), and the difference is used to adjust the motion trajectory by:

- running a position control loop to improve accuracy,
- handling stalls, and
- detecting unexpected displacements when the axis is not moving.

For more information on stalls and displacements, and to configure how they are handled, see cloop.recovery.enable.
On stepper motor products, closed-loop mode can be enabled or disabled using the cloop.enable setting. When closed-loop mode is disabled, all encoder settings are kept up-to-date and encoder-specific warning flags will continue to appear if there is a problem with the encoder. However, the position read from the encoder will not be used to control the position of the axis, and stalls and displacements will not be detected.

cloop.continuous.enable

The control for whether continuous closed-loop correction is enabled

Valid Values

0

Disabled

1

Enabled

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies whether or not the position control loop continues to run once the target has been reached and the axis reports IDLE.

When this setting is disabled, the position control loop runs only while the axis is in motion. During motion, the position control loop continuously reads the encoder position and adjusts to match the requested trajectory. When IDLE, the motor current is held constant and the axis remains stationary except for thermal drift and movement due to outside force.

When this setting is enabled, the position control loop runs both when the axis is moving and when it is stationary. The position control loop continuously reads the encoder position and adjusts to match the requested trajectory. Continuously running the control loop even while stationary can be useful for maintaining a position under varying load conditions. Enabling this setting has two potentially negative side effects:

1. The axis may move slightly when IDLE as the control loop attempts to compensate for fluctuations in the encoder reading.
2. The axis may become backdrivable. If this is a problem, try increasing driver.current.hold to match driver.current.run.

cloop.displace.tolerance

The minimum position deviation that registers as a displacement

Valid Values

0 to 2,000,000,000

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

The minimum deviation in the position of a stationary axis that will register as a displacement and set the ***Displaced When Stationary*** (WM) flag.

See cloop.recovery.enable for further details about stalls and displacements.

See the Physical Units section for information about unit conversions.

cloop.enable

The control for whether closed-loop mode is enabled

Valid Values

0

Disabled

1

Enabled

Scope

Axis

Write Access Level

Normal for stepper motor products

Persistence

Non-volatile

Specifies whether or not the encoder-measured position (encoder.pos) is used to adjust the motion trajectory.

Note: On products with an encoder, encoder.count, encoder.pos, and encoder.pos.error are kept up to date and encoder errors are reported regardless of whether closed-loop mode is enabled. However, when closed-loop mode is disabled this information does not affect the behaviour of the axis.

A stepper motor can occasionally slip (when the rotating electric field cannot produce enough torque to drive the load, so four steps of the stepper motor are skipped). On stepper motor products with a motor encoder, when an axis slips with closed-loop mode enabled:

- pos is shifted by four full steps to better reflect the encoder position, and
- the blue LED turns on for 250 ms.

A slip indicates that the device is operating near its limits; consider increasing driver.current.run, moving at a slower speed or lower acceleration, or decreasing the load on the stage.

cloop.recovery.enable

The control for whether closed-loop recovery mode is enabled

Valid Values

0

Disabled

1

Enabled

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies whether or not closed-loop recovery mode is enabled.

This setting only affects behaviour when closed-loop mode (cloop.enable) is enabled.

Closed-loop recovery mode controls the behaviour of the axis when either a stall or displacement occurs. This behaviour is summarized in Table 2.

A stall occurs after a moving axis fails to keep up with the motion trajectory for a duration of cloop.timeout.

The precise conditions under which a stall is reported vary based on the motor being used. In firmware versions 7.26 and below, a stall occurs after an axis fails to make forward progress towards its target. In firmware versions 7.27 and above, the stall criteria are described in `cloop.stall.detect.mode`.

A displacement occurs after a stationary axis is forced out of position. The precise conditions under which a displacement is reported vary based on the motor and encoder being used.

For stepper motor products:

- On devices with a motor encoder, a displacement occurs after `encoder.pos.error` on a stationary axis exceeds two steps of the motor for a duration of `cloop.timeout`.

Table 2. Stall and Displacement Behaviour

	Recovery Enabled	Recovery Disabled
Stall	• The axis attempts to recover from the stall and continue towards the target. • Flag: ***Stalled with Recovery*** (`WS`)	• After a stall, the axis stops moving. • Flag: ***Stalled and Stopped*** (`FS`) • The blue LED blinks once every two seconds.
Displacement	• After a displacement, the axis attempts to return to the pre-displaced position. • Flag: ***Displaced When Stationary*** (`WM`) • The blue LED flashes twice per second.	• After a displacement, the axis remains at the new position. • Flag: ***Displaced When Stationary*** (`WM`) • The blue LED flashes twice per second.

Note: If the ***No Reference Position*** (`WR`) is present, the axis will not recover from stalls, even if recovery is enabled.

`cloop.stall.action`

The action to take after detecting a stall if recovery is disabled

Valid Values

0

Instant stop

1

Controlled stop

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies the action to take after a stall is detected. This setting only affects behaviour when `cloop.recovery.enable` is set to 0 or recovery is impossible.

An instant stop will attempt to stop the axis with infinite deceleration.

A controlled stop will apply `motion.decelonly` until the axis has stopped. In firmware versions 7.28 and above, if the stalled motion was started by a `stream set` or `move` command that specifies `accel` via a custom *<accel>* parameter, the axis will apply the larger of `motion.decelonly` and the custom *<accel>*.

Warning: Stopping instantly may result in damage to the product and reduced lifespan. Use the instant stop feature with extreme caution if the axis is under heavy load. Under heavy load, attempting to stop instantly may cause the driver to shut off and the stage to continue moving in an uncontrolled manner.

Introduced in 7.27

cloop.stall.detect.mode

Determines what registers as a stall

Valid Values

0

Lack of forward progress

1

`encoder.pos.error` greater than `cloop.stall.tolerance`

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies the criterion used to detect stalls. This setting only affects behaviour when `cloop.enable` is set to 1.

In mode 0, a stall is reported when an axis fails to move towards its target.

In mode 1, a stall is reported when the encoder-measured position error `encoder.pos.error` exceeds the tolerance defined by `cloop.stall.tolerance`.

In either case, a stall is only reported when the criterion has persisted for longer than `cloop.timeout`.

Introduced in 7.27

cloop.stall.tolerance

The minimum position error that registers as a stall

Valid Values

0 to 2,000,000,000

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

The minimum position error while the axis is moving that will register as a stall. When the stall has persisted for `cloop.timeout` it will be reported.
This setting is only applicable if `cloop.enable` is set to 1
and `cloop.stall.detect.mode` is also set to 1.
See the Physical Units section for information about unit conversions.

Introduced in 7.27

`cloop.timeout`

The timeout used to report stalls and displacements

Valid Values

0 to 65,535

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies the timeout, in units of milliseconds, for reporting stalls and displacements. Stalls and displacements are described in
the `cloop.recovery.enable` documentation.

`comm`

These settings contain controls for, and information about, communication interfaces and advanced ASCII Protocol message features.

The ASCII protocol supports several different communication interfaces over which computers can send commands to the device and receive responses. Every Bosch Rexroth device has a previous port which can be used to communicate with the device over the serial RS-232 interface.

Each Bosch Rexroth device has a next port to which additional Bosch Rexroth devices can be connected in a daisy-chain. The next port of the first device in the chain communicates with the previous port of the second device in the chain over RS-232.

The entire chain can then be controlled from a computer by using any of the communication interfaces available on the first device.

The following settings are applicable regardless of which interface is being used:

- `comm.address` configures the ASCII address of the device, which determines which commands the device responds to.
- `comm.alert` controls whether the device sends alert messages.
- `comm.checksum` controls whether the device includes message checksums in responses.

The next sections detail communication settings relevant to specific communication interfaces.

RS-232

The previous and next ports use serial RS-232 communication at the baud rate specified by `comm.rs232.baud`. In order to successfully communicate, each daisy-chained device should be configured to use the same baud rate and the same protocol.

comm.address

The address of the device

Valid Values

1 to 99

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

The device responds to any commands with a device address that matches `comm.address`.

If `comm.address` is changed, the reply to the `set` command uses the new device address.

Example: Changing the device address

Change the device address from 1 to 5:

```
/01 set comm.address 5↵
@05 0 OK IDLE -- 0
```

comm.alert

The control for whether alert messages are sent

Valid Values

0

Disabled

Alert messages are not sent

1

Enabled

Alert messages are sent

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

When set to 1, the device sends alert messages. When set to 0, the device does not send alert messages.

When enabled, an alert message is sent once a movement is complete.

Example: Receiving alert message when a movement completes

Move device 1 to the absolute position 50,000.

```
/01 set comm.alert 1↵
@01 0 OK IDLE -- 0
/01 move abs 50000↵
@01 0 OK IDLE -- 0
!01 1 IDLE --
```

comm.checksum

The control for whether message checksums are included in messages

Valid Values

0

Disabled

Checksums are not included in messages

1

Enabled

Checksums are included in messages

2

Automatic

Checksums are included in Replies and Info messages sent in response to commands that have checksums

Introduced in 7.27

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

When set to 1, all messages (Replies, Alerts, and Info messages) sent from a device include a colon (`:`) followed by a two-digit hexadecimal checksum value immediately before the message footer.
When set to 2, the device includes checksums in responses to commands if and only if the commands include checksums. As such, alerts will never include checksums in this mode because they are not sent in response to
commands. Only Replies and Info messages may have checksums.

Mode 2 allows checksums to be used with a mix of software that may or may not understand checksums without changing any settings.

Commands may always include checksums, regardless of the value of this setting. If a command includes a checksum, it will be validated. However, the device will ignore and not respond to commands with invalid checksums.

See Checksums for more information about checksums.

Example: Receiving a checksum value in a reply

Send a movement command to a device that has `comm.checksum` enabled.

```
/01 move rel 10000↵
@01 0 OK IDLE -- 0:8D
```

comm.command.packets.max

The maximum number of packets in an ASCII command message

Valid Values

1 to 255

Scope

Device

Write Access Level

Read-only

Persistence

Non-volatile

Specifies the maximum number of packets that can be used when using Line Continuation with ASCII Commands.
If a command is split into more than this number of packets, it is rejected with the reason BADSPLIT.

Introduced in 7.26

comm.packet.size.max

The maximum number of bytes in one ASCII packet

Valid Values

80 to 65,535

Scope

Device

Write Access Level

Read-only

Persistence

Non-volatile

Specifies the maximum number of bytes allowed in a single ASCII packet sent to or from the device.

This includes the message type, the footer, and everything in between.

It applies to all message types; it also applies to packets sent to or from the device, as well as packets sent through the device to or from other devices on the daisy-chain.

Introduced in 7.26

comm.rs232.baud

The baud rate used on the RS-232 previous and next ports

Valid Values

9600, 19200, 38400, 57600, 115200

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

The device expects to receive, and will send, messages at this speed over its previous port. The next port also operates at this speed; therefore, if multiple devices are daisy-chained, they must all be configured to the same baud rate. When changing this setting, the reply will be sent at the old baud rate. The device will stop accepting commands at the old baud rate, but continue to forward replies from other devices. After the device

receives no communication for 200 ms, it will resume communication at the new baud rate.

Example: Changing the baud rate of two devices

Change the baud rate of two daisy-chained devices by sending the command to both devices at once:

```
/0 set comm.rs232.baud 9600↵
@01 0 OK IDLE -- 0
@02 0 OK IDLE -- 0
```

After 200 ms, both devices accept commands at 9600 baud.

comm.word.size.max

The maximum number of characters in a single word in an ASCII message

Valid Values

1 to 65,535

Scope

Device

Write Access Level

Read-only

Persistence

Non-volatile

Specifies the maximum size allowed for a word in an ASCII message sent to or from the device.

If a command sent to the device contains a word longer than this maximum, the command is rejected with the reason.

Introduced in 7.26

device

These settings contain information about the configuration of the physical device.

Each Bosch Rexroth model number that includes a controller (whether integrated or separate) has a device ID associated with it. Firmware uses this ID (the value of the device.id setting) to properly configure the settings and control options available to that device.

device.id

The ID number of the device

Valid Values

The value defined on the linear module label

Scope
Device

Write Access Level
Read-only

Persistence
Non-volatile

Specifies the model of Bosch Rexroth device.

driver

These settings contain controls for and information about the axis driver, which controls current to the motor.

See the driver commands section for commands related to the axis driver.

driver.current.hold

The current used when holding the position of the axis

Valid Values
0 to the minimum of driver.current.max and motor.current.max / √2

Scope
Axis

Write Access Level
Normal

Persistence
Non-volatile

The peak current, in 20 mA increments, that the driver may apply to each motor phase to maintain the motor's position. It is typical to drive a motor at its rated current when it is moving (driver.current.run), and to reduce the current when idle to hold the position. When the driver stops moving the axis, it waits 100 ms, then switches from applying run current to hold current. The default hold current is typically 25 % - 50 % of the run current.

For certain applications of stepper motors, the friction of the drive system may be sufficient to hold the position of the motor, and driver.current.hold can be set to 0.

Warning: On some products, increasing driver.current.hold may raise the temperature high enough to cause burns or damage the motor. Please contact Bosch Rexroth Support if you have questions or concerns about raising driver.current.hold.

Example: Setting the hold current

Set the hold current to 800 mA (800 mA / 20 mA = 40):

```
/set driver.current.hold 40↵
```

```
@01 0 OK BUSY -- 0
```

driver.current.max

The maximum current the driver can output continuously

Valid Values

1 to 214,748

Scope

Axis

Write Access Level

Read-only

Persistence

Non-volatile

The maximum continuous current, in 20 mA increments, that the axis driver is capable of outputting to a motor phase.

Example: Reading the maximum driver current

```
/get driver.current.max↵
@01 0 OK BUSY -- 100
```

The driver can output a maximum phase current of 100 × 20 mA = 2.0 A.

driver.current.run

The current used when moving the axis

Valid Values

0 to the minimum of driver.current.max and motor.current.max

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

The current, in 20.0 mA peak (14.1 mA RMS) increments, applied to each motor phase when moving. The default driver.current.run value for an axis is based on the current rating of the motor, as well as the temperature the device reaches during continuous operation. The driver.current.run value can be reduced to lower the temperature and power consumption of the device, but it will also lower the torque the motor can generate. It can be raised to increase the motor's torque, but this may also raise the temperature. See driver.current.hold for information on the current when the driver is not moving the axis.

Warning: On some products, increasing `driver.current.run` may raise the temperature high enough to cause burns or damage the motor; lowering the fraction of time the motor is moving is one way to offset the temperature rise. Please contact Bosch Rexroth Support if you have questions or concerns about raising `driver.current.run`.

Example: Setting the run current

Set the run current to 1.2 A RMS (1200 mA RMS / 14.1 mA RMS = 85):

```
/set driver.current.run 85↵
@01 0 OK BUSY -- 0
```

driver.dir

Whether the axis driver direction is inverted

Valid Values

0
Not inverted

0
Inverted

Scope

Axis

Write Access Level

Advanced for stepper motor products

Persistence

Non-volatile

Specifies whether the direction of the axis driver is inverted. This setting determines whether moving in the positive direction causes the axis to move towards or away from the motor (on linear products), or in a clockwise or counter-clockwise direction (on rotary products).

The move direction that produces a positive change in `pos` depends on how the motor is connected. As such, the value for this setting is often determined experimentally.

Note: On products with an encoder, always switch `driver.dir` and `encoder.dir` concurrently. Otherwise, the axis will stall.

driver.enabled

The state of the axis driver

Valid Values

0
Disabled

1
Enabled

Scope

Axis

Write Access Level
Read-only

Persistence
Volatile

Specifies whether the axis driver is enabled. If the driver is disabled, one or more of the following warning flags are present indicating the reason:

- ***Current Inrush Error*** (FC) (firmware versions 7.11 and above only)
- ***Driver Temperature/Current Error*** (FD)
- ***Motor Temperature Error*** (FM) (firmware versions 7.11 and above only)
- ***Driver Disabled*** (FO) (firmware versions 7.11 and above only)
- ***Overvoltage or Undervoltage Driver Disabled*** (FV)

Although the `driver.enabled` setting is Read-only, the `driver enable` and `driver disable` commands can be used to change the state.

Example: Checking driver status

```
/get driver.enabled↵
@01 0 OK IDLE FD 0
```

A value of 0 indicates that the driver is disabled. This is confirmed by the presence of the ***Driver Temperature/Current Error*** (FD) warning flag.

driver.temperature

The axis driver temperature

Valid Values
0 to 150

Scope
Axis

Write Access Level
Read-only

Persistence
Volatile

Specifies the temperature, in units of degrees Celsius, of the axis driver.
The ***Temperature High*** (WT) warning flag is set if the axis driver becomes too hot to guarantee normal axis operation.
See also `system.temperature`.

Example: Reading the temperature of the driver

```
/get driver.temperature↵
@01 0 OK IDLE -- 53.5
```

The driver temperature is presently 53.5°C.

encoder

These settings contain controls for, and information about, the position encoder.

Position encoders measure the position of the axis in units of counts. A count is the smallest distance an encoder can resolve, and is different for every encoder. To read this value directly, see encoder.count. The encoder.pos setting is the encoder-measured position translated into the same reference frame as pos and into distance units (see Physical Units). To read how far off the encoder-measured position is from the target position, see encoder.pos.error.

encoder.count

The position of the axis since power-up, measured in encoder counts

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Axis

Write Access Level

Read-only

Persistence

Volatile

Specifies the position of the axis, measured with the encoder, in units of encoder counts.

The value of encoder.count is 0 at power-up or after a system reset. From then on, the encoder count tracks the position of the axis as it moves. The physical unit of an encoder count depends on the encoder scale and the particular encoder that is used, but it represents the smallest distance that will register as a change in position. The value of encoder.count is unchanged by actions that change the position reference, such as setting pos or homing.

The relationship between encoder.count and encoder.pos is described in the encoder section.

encoder.dir

Whether the direction of the encoder is inverted

Valid Values

0

Not inverted

1

Inverted

Scope

Axis

Write Access Level
Advanced for stepper motor products

Persistence
Non-volatile

Specifies whether the direction of the encoder is inverted. This setting determines whether `encoder.count` and `encoder.pos` increase or decrease when the axis moves towards or away from the motor (on linear products), or in a clockwise or counter-clockwise direction (on rotary products).

The move direction that produces a positive change in `encoder.count` and `encoder.pos` depends on how the encoder is connected and which type of encoder is connected. As such, the value for this setting is often determined experimentally.

Note: Always switch `driver.dir` and `encoder.dir` concurrently. Otherwise, the axis will stall.

encoder.pos

The position of the axis measured with the encoder

Valid Values
-1,000,000,000 to 1,000,000,000

Scope
Axis

Write Access Level
Read-only

Persistence
Volatile

Specifies the position measured with the encoder.

The relationship between `encoder.pos` and `encoder.count` is described in the `encoder` section.

See the Physical Units section for information about unit conversions.

encoder.pos.error

The difference between the target and encoder-measured positions

Valid Values
-2,000,000,000 to 2,000,000,000

Scope
Axis

Write Access Level
Read-only

Persistence
Volatile

The difference between the target position and the encoder-measured position:
encoder.pos.error = pos - encoder.pos

See the Physical Units section for information about unit conversions.

ictrl

These settings control closed-loop current controllers.

Some Bosch Rexroth products support tunable current controllers for modifying the motor's performance. The current controllers on Bosch Rexroth products are already tuned to perform well in the vast majority of situations. You should only need to tune the current controller if the default tuning is not providing sufficient performance, or if you are configuring a non-Bosch Rexroth peripheral. If you do need to tune the current controller, we do not recommend modifying these settings manually. Instead, please contact Bosch Rexroth Support.

Warning: Changing current controller settings can affect the heat generated by the device and can lead to audible ringing or stalling. It is important to test any changes to make sure operation is acceptable. Ensure nothing else is interacting with the axis while updating and testing current controller settings.

ictrl.advance.a

The current controller advance term

Valid Values

0 to 18,446,744,073.0

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

A parameter used by some current controllers to update their internal state. It is unitless and to nine decimal places.

Introduced in 7.27

ictrl.advance.offset

The current controller advance offset term

Valid Values

0 to 32,768.0

Scope

Axis

Write Access Level
Advanced

Persistence
Non-volatile

A parameter used by some current controllers to update their internal state.

It has up to nine decimal places and can be converted to degrees with:
Angle = data × 360° / 32,768

Introduced in 7.27

ictrl.afcff.inductance

The current controller inductance term

Valid Values
0 to 18,446,744,073.0

Scope
Axis

Write Access Level
Advanced

Persistence
Non-volatile

The value of motor inductance used by some current controllers to update their internal state.

It has up to nine decimal places and can be converted to units of henries with:
L = data × 1638.4 / π

Introduced in 7.27

ictrl.afcff.ke

The current controller back EMF constant parameter

Valid Values
0 to 18,446,744,073.0

Scope
Axis

Write Access Level
Advanced

Persistence
Non-volatile

The back EMF constant of the motor is used by some current controllers to calculate maximum expected value of back EMF.

It has up to nine decimal places and can be converted to V·s/rad units with:
K_e = data × 16,384 / (10,000,000 × π)

Note that the angular component references the motor's electrical phase angle, not the motor's shaft angle.

Introduced in 7.27

ictrl.afcff.ki

The back EMF compensation integral gain

Valid Values

0 to 18,446,744,073.0

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

The integral gain coefficient applied in the back EMF compensation module of the current controller, if it has one.

It has up to nine decimal places and can be converted to V/(A·s) units with:
K_i = data × 40,000,000

Introduced in 7.27

ictrl.afcff.max

The back EMF compensation anti-windup limit

Valid Values

0 to 18,446,744,073.0

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

The maximum allowed output of the back EMF compensation module in the current controller, if it has one.

It is in units of millivolts and to nine decimal places.

Introduced in 7.27

ictrl.afcff.ss

The back EMF compensation damping factor

Valid Values

0 to 18,446,744,073.0

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

The damping factor used in the back EMF compensation module of the current controller, if it has one.

It is unitless and to nine decimal places.

Introduced in 7.27

ictrl.afcff.ss.max

The maximum value of the back EMF compensation damping term

Valid Values

0 to 18,446,744,073.0

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

The maximum value that the damping term is allowed to have in the back EMF compensation module of the current controller, if it has one.

It is unitless and to nine decimal places.

Introduced in 7.27

ictrl.delay

The delay through the current controller loop

Valid Values

0 to 65.535

Scope

Axis

Write Access Level

Read-only

Persistence

Non-volatile

The delay through the current controller loop, in units of milliseconds and to three decimal places.

Introduced in 7.27

ictrl.ff.kd

The feed-forward derivative gain

Valid Values

0 to 18,446,744,073.0

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

The derivative gain coefficient of the feed-forward (FF) module of the current controller, if it has one.

It has up to nine decimal places and can be converted to V·s/A (or henry) units with:
K_d = data / 40

Introduced in 7.27

ictrl.ff.kp

The feed-forward proportional gain

Valid Values

0 to 18,446,744,073.0

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

The proportional gain coefficient of the feed-forward (FF) module of the current controller, if it has one. It is in units of mV/µA (or kiloohm) and to nine decimal places.

Introduced in 7.27

ictrl.gain.coldmult

Current controller gain scaling factor used while IDLE

Valid Values

0.1 to 1

Scope

Axis

Write Access Level
Advanced

Persistence
Non-volatile

The scale factor for the current controller's feedback gain used when the axis is IDLE.

It is unitless and to nine decimal places.

Introduced in 7.27

`ictrl.period`

The period of the current controller loop

Valid Values
0 to 65.535

Scope
Axis

Write Access Level
Read-only

Persistence
Non-volatile

The period of the current controller loop, in units of milliseconds and to three decimal places.

Introduced in 7.27

`ictrl.pi.ki`

The PI integral gain

Valid Values
0 to 18,446,744,073.0

Scope
Axis

Write Access Level
Advanced

Persistence
Non-volatile

The integral gain coefficient of the proportional-integral (PI) module of the current controller, if it has one.

It has up to nine decimal places and can be converted to V/(A·s) units with:
$K_i = \text{data} \times 40{,}000{,}000$

Introduced in 7.27

ictrl.pi.kp

The PI proportional gain

Valid Values

0 to 18,446,744,073.0

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

The proportional gain coefficient of the proportional-integral (PI) module of the current controller, if it has one.

It is in units of mV/µA and to nine decimal places.

Introduced in 7.27

ictrl.type

The type of closed-loop current-controller to use

Valid Values

0
Controller 0

1
Controller 1

2
Controller 2

3
Controller 3

4
Controller 4

5
Controller 5

6
Controller 6

7
Controller 7

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

Specifies the type of controller used to control motor current.

Warning: This should be left at the default value. Only change this setting if you know what you are doing. Unexpected, erratic, damaging, or potentially dangerous behaviour may result if the incorrect controller is used.

Introduced in 7.27

io

These settings contain controls and information related to the digital input or output (IO) channels on the device.

See the io commands section for more information about working with IO channels.

io.di.port

The value of the digital input port

Valid Values

0 to 2^(the value returned by io info di) - 1

Scope

Device

Write Access Level

Read-only

Persistence

Volatile

The io.di.port setting combines the states of all the digital input channels into a single value by encoding them as a binary number. The value is calculated by summing (the state of the channel) × 2^(channel - 1) over each channel, from 1 to the value returned by io info di.

To read the states of individual digital input channels more easily, see the io get di command.

Introduced in 7.22

Example: Reading the states of the digital input channels

First, read the total number of digital input channels:

```
/io info di↵
@01 0 OK IDLE -- 3
```

Next, read the value of the digital input port:

```
/get io.di.port↵
@01 0 OK IDLE -- 3
```

Then, convert this to binary: 0b011. Since the least-significant two bits are 1, the channels 1 and 2 are currently high. Since the most-significant bit is 0, the state of channel 3 is currently low.

To verify this result, use the `io get di` command:

```
/io get di port↵
@01 0 OK IDLE -- 1 1 0
```

io.do.port

The value of the digital output port

Valid Values

0 to 2^(the value returned by `io info do`) - 1

Scope

Device

Write Access Level

Normal

Persistence

Volatile

The `io.do.port` setting combines all the digital output channel states into a single value by encoding them as a binary number. The value is calculated by summing (the state of the channel) × 2^(channel - 1) over each channel, from 1 to the value returned by `io info do`.

To control and monitor the states of individual digital output channels more easily, see the `io set do` and `io get do` commands.

Introduced in 7.22

Example: Setting the third digital output to high, and the rest to low

To create a bitwise representation, set the 3rd-least-significant bit to 1: 0b100. In decimal notation, this is 4.

```
/set io.do.port 4↵
@01 0 OK IDLE -- 0
```

To verify this, use the `io get do` command:

```
/io get do port↵
@01 0 OK IDLE -- 0 0 1 0
```

knob

These settings contain controls for, and information about, the manual control knob.

The knob is located on the body of the controller. It can be used to manually control the motion of the axis without sending commands to the serial port. These settings configure how the axis moves for each increment the knob is rotated.

Pressing the knob stops the axis (unless knob.enable is set to 0). The axis decelerates at the rate defined by motion.decelonly unless the axis is already stopping; if the axis is already stopping, the axis stops instantly.

Warning: Stopping instantly may result in potential damage to the product and reduced lifespan. Use sparingly if the axis is under heavy load.

knob.dir

The relation between knob and device movement directions

Valid Values

0
Clockwise

1
Counter-clockwise

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies the direction to rotate the knob in order to move the axis in the positive direction.

knob.distance

The distance moved for each knob increment

Valid Values

0 to 2,000,000,000

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies how far the axis moves with each knob increment when knob.mode is set to Displacement Mode.

If set to 0, each increment of the knob moves the axis to the next index position. See move index for more details.

See the Physical Units section for information about unit conversions.

knob.enable

The control for whether the knob is enabled

Valid Values

0
Disabled

1
Enabled

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

If this setting is set to 1, the knob can be used to move or stop the device. If this setting is set to 0, turning or pressing the knob is ignored.

knob.maxspeed

The maximum speed that can be reached by the knob in Velocity Mode

Valid Values

1 to resolution × 16,384 for stepper motor products

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies the maximum speed at which the axis will move as the knob is turned when knob.mode is set to Velocity Mode.

In firmware versions 7.09 and above, when the axis does not have a reference position (i.e. the ***No Reference Position*** (WR) warning flag is set), the speed achievable with the knob is limited to the minimum of knob.maxspeed and limit.approach.maxspeed.

The knob.speedprofile setting determines the rate at which the speed increases as the knob is turned.

See the Physical Units section for information about unit conversions.

knob.mode

The mode of the manual control knob

Valid Values

0

Velocity Mode

1

Displacement Mode

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies the mode of the manual control knob.

In Velocity Mode, each knob increment increases the speed of the axis. The maximum speed, `knob.maxspeed`, is reached after 16 increments and the magnitude of each speed increment is determined by the `knob.speedprofile` setting.

In Displacement Mode, each knob increment moves the axis a distance of `knob.distance`.

To switch between these two modes manually, press and hold the knob for 1 second.

knob.speedprofile

The control for how the velocity changes as the knob is turned

Valid Values

1
Linear

2
Quadratic

3
Cubic

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies the relationship between knob increments and velocity when `knob.mode` is set to Velocity Mode. For instance, if set to 2 (Quadratic), the velocity will change quadratically with each increment of the knob.

speed at i^{th} knob increment = `knob.maxspeed` × (i / 16) knob.speedprofile

The different profiles are shown below. No matter which profile is selected, `knob.maxspeed` will be reached in 16 increments.

Note: In firmware versions 7.09 and above, when the axis does not have a reference position (i.e. the ***No Reference Position*** (WR) warning flag is set), the speed achievable with the knob is limited to the minimum of limit.approach.maxspeed and knob.maxspeed.

limit

These settings contain controls for, and information about, the axis limits.

limit.approach.maxspeed

The maximum speed to use when there is no reference position

Valid Values

1 to resolution × 16,384 for stepper motor products

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

When the axis does not have a reference position (i.e. the ***No Reference Position*** (WR) warning flag is set), or during a home command, the axis travels at the lesser of limit.approach.maxspeed and maxspeed. To safely control fast-moving devices during homing, set limit.approach.maxspeed lower than maxspeed.

See the Physical Units section for information about unit conversions.

limit.away.action

The action taken when the away limit sensor becomes active

Valid Values

0

Disabled

Take no action when the away sensor becomes active.

1

Retract

Retract to the side of the sensor.

2

Retract and update pos

Retract to the side of the sensor, write the value of limit.away.preset to pos, move to the position specified by limit.away.offset, and clear the ***No Reference Position*** (WR) warning flag.

3

Retract and update pos if not yet homed

If the ***No Reference Position*** (WR) warning flag is set, retract and update pos then move to the position specified by limit.away.offset. Otherwise, ignore the sensor and take no action.

4

Retract and update pos without offset

Retract to the side of the sensor, write the value of limit.away.preset to pos and clear the ***No Reference Position*** (WR) warning flag without performing any offset movement.

Introduced in 7.27

5

Retract and update pos without offset if not yet homed

If the ***No Reference Position*** (WR) warning flag is set, retract and update pos without performing any offset movement. Otherwise, ignore the sensor and take no action.

Introduced in 7.27

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

Specifies the automatic action to perform when the axis activates the away sensor. The away sensor is active when limit.away.state is 1.

If a limit action is triggered and the ***No Reference Position*** (WR) warning flag is not present, the ***Unexpected Limit Trigger*** (WL) warning flag is set since travel within the valid range of the axis should not ordinarily trigger limit sensors after the reference position is set.

If multiple limit sensors are present, at most one should be configured to update pos when the sensor activates. Otherwise, the value of pos could be inconsistent depending upon which sensor was most recently encountered.

Example: Selecting a value for a linear device

Suppose the device moves linearly between two limit sensors. Passing either sensor is not desirable, because the stage could collide with the end of its travel range. Both limit sensors should therefore use a limit action that will always retract the axis (even if the axis is homed), so that travel is always restricted between the sensors. The limit sensor which is being used as the position reference should use either `2` or `4` for its limit action, so that the position reference is kept up to date when the sensor is encountered. The other limit sensor should use `1` for its limit action, so that it will restrict travel without interfering with the position reference.

limit.away.offset

The offset distance moved at the end of away limit actions

Valid Values

−2,000,000,000 to 2,000,000,000

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies the distance the axis will move when the away limit sensor is spontaneously encountered or at the end of `tools gotolimit` commands. The offset is relative to the `limit.away.preset` position and the move only occurs as part of some limit actions (specified by `limit.away.action`).

If the movement would take the axis outside the range `limit.min` to `limit.max`, the axis will not move the specified offset, and the ***Limit Error*** (`FE`) warning flag will be set.

The movement is similar to a `move rel` issued from the `limit.away.preset` position except that in some cases (e.g., an axis that homes to an index mark) the axis may not need to first travel to the `limit.away.preset` position before moving to the offset position.

Introduced in 7.27

Example: Automatically moving a specific distance after encountering the away limit sensor

Configure the axis to move -200 position units from the away limit sensor after encountering the sensor:

```
/set limit.away.offset -200↵
```

```
@01 0 OK IDLE -- 0
```

Sending `/tools gotolimit away pos 2 0↵` will move the axis to find the away limit sensor, after which `pos` will equal `limit.away.preset` - 200.

Example: Automatically moving to a specific position after encountering the away limit sensor

Consider an axis that should move to the centre of travel after encountering the away limit sensor. Suppose `limit.min` is 0, `limit.max` is 125000, and `limit.away.preset` is 125000. Then the midpoint of travel is (125000 - 0) / 2 = 62500 and `limit.away.offset` should be set to midpoint - `limit.away.preset` = 62500 - 125000 = -62500:

```
/set limit.away.offset -62500↵
@01 0 OK IDLE -- 0
```

Sending `/tools gotolimit away pos 2 0↵` will move the axis to find the away limit sensor, after which `pos` will equal 62500.

limit.away.posupdate

The control for how `limit.max` is updated after a limit action

Valid Values

0

Disabled

Do not update `limit.max`.

1

Set to current position

Write the current position (`pos`) to `limit.max`.

2

Set to current position and save

Write the current position (`pos`) to `limit.max` and save it to non-volatile memory so that it persists after the next power-up.

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

Depending on the value of this setting, the device may update the away sensor position (limit.max) after a limit action is complete. This setting has no effect if the limit action is disabled (limit.away.action is 0).

limit.away.preset

The value away limit actions write to pos

Valid Values

-1,000,000,000 to 1,000,000,000

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

The value written to pos after away limit actions (specified by limit.away.action). This occurs as part of the tools gotolimit command, or after triggering the away sensor.

See the Physical Units section for information about unit conversions.

limit.away.state

The current state of the away limit sensor

Valid Values

0

Inactive

The sensor is not active.

1

Active

The sensor is active.

Scope

Axis

Write Access Level

Read-only

Persistence

Volatile

If no limit sensor is connected, this setting is always 0.

limit.away.triggered

Whether the axis has triggered the away sensor

Valid Values

0

The away sensor has not been triggered.

1

The away sensor has been triggered.

Scope

Axis

Write Access Level

Read-only

Persistence

Volatile

On power-up this setting is set to 0. The setting is set to 1 after an away limit action or a limit sensor command completes.

limit.detect.decelonly

The deceleration to use when stopping after a limit sensor has been triggered

Valid Values

0 to 2,147,483,647

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

When a limit sensor is triggered, the axis slows down to a stop at this rate. A value of 0 specifies that the axis should stop immediately.

See the Physical Units section for information about unit conversions.

limit.detect.maxspeed

The maximum speed used when aligning with the edge of a limit sensor

Valid Values

1 to 68,719,476,704

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

After a limit sensor is triggered and the axis has stopped, it determines the precise edge of the sensor by moving away from the sensor at this speed.

See the Physical Units section for information about unit conversions.

limit.home.action

The action taken when the home limit sensor becomes active

Valid Values

0

Disabled

Take no action when the home sensor becomes active.

1

Retract

Retract to the side of the sensor.

2

Retract and update pos

Retract to the side of the sensor, write the value of limit.home.preset to pos, move to the position specified by limit.home.offset, and clear the ***No Reference Position*** (WR) warning flag.

3

Retract and update pos if not yet homed

If the ***No Reference Position*** (WR) warning flag is set, retract and update pos then move to the position specified by limit.home.offset. Otherwise, ignore the sensor and take no action.

4

Retract and update pos without offset

Retract to the side of the sensor, write the value of limit.home.preset to pos and clear the ***No Reference Position*** (WR) warning flag without performing any offset movement.

Introduced in 7.27

5

Retract and update pos without offset if not yet homed

If the ***No Reference Position*** (WR) warning flag is set, retract and update pos without performing any offset movement. Otherwise, ignore the sensor and take no action.

Introduced in 7.27

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

Specifies the automatic action to perform when the axis activates the home sensor. The home sensor is active when `limit.home.state` is 1.
If a limit action is triggered and the ***No Reference Position*** (`WR`) warning flag is not present, the ***Unexpected Limit Trigger*** (`WL`) warning flag is set since travel within the valid range of the axis should not ordinarily trigger limit sensors after the reference position is set.
If multiple limit sensors are present, at most one should be configured to update `pos` when the sensor activates. Otherwise, the value of `pos` could be inconsistent depending upon which sensor was most recently encountered.

Example: Selecting a value for a linear device

Suppose the device moves linearly between two limit sensors. Passing either sensor is not desirable, because the stage could collide with the end of its travel range. Both limit sensors should therefore use a limit action that will always retract the axis (even if the axis is homed), so that travel is always restricted between the sensors. The limit sensor which is being used as the position reference should use either `2` or `4` for its limit action, so that the position reference is kept up to date when the sensor is encountered. The other limit sensor should use `1` for its limit action, so that it will restrict travel without interfering with the position reference.

`limit.home.offset`

The offset distance moved at the end of home limit actions

Valid Values

−2,000,000,000 to 2,000,000,000

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies the distance the axis will move when the home limit sensor is spontaneously encountered or at the end of `home` and `tools gotolimit` commands. The offset is relative to the `limit.home.preset` position and the move only occurs as part of some limit actions (specified by `limit.home.action`).

If the movement would take the axis outside the range limit.min to limit.max, the axis will not move the specified offset, and the ***Limit Error*** (FE) warning flag will be set. The movement is similar to a move rel issued from the limit.home.preset position except that in some cases (e.g., an axis that homes to an index mark) the axis may not need to first travel to the limit.home.preset position before moving to the offset position.

Introduced in 7.27

Example: Automatically moving a specific distance after encountering the home limit sensor

Configure the axis to move 200 position units from the home limit sensor after encountering the sensor:

```
/set limit.home.offset 200↵
@01 0 OK IDLE -- 0
```

Sending /tools gotolimit home neg 2 0↵ or home with an appropriate value for limit.home.action will move the axis to find the home limit sensor, after which pos will equal limit.home.preset + 200.

Example: Automatically moving to a specific position after encountering the home limit sensor

Consider an axis that should move to the centre of travel after encountering the home limit sensor. Suppose limit.min is 0, limit.max is 125000, and limit.home.preset is 500. Then the midpoint of travel is (125000 - 0) / 2 = 62500 and limit.home.offset should be set to midpoint - limit.home.preset = 62500 - 500 = 62000:

```
/set limit.home.offset 62000↵
@01 0 OK IDLE -- 0
```

Sending /tools gotolimit home neg 2 0↵ or home with an appropriate value for limit.home.action will move the axis to find the home limit sensor, after which pos will equal 62500.

limit.home.posupdate

The control for how limit.min is updated after a limit action

Valid Values

0

Disabled

Do not update limit.min.

1

Set to current position

Write the current position (pos) to limit.min.

2

Set to current position and save

Write the current position (pos) to limit.min and save it to non-volatile memory so that it persists after the next power-up.

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

Depending on the value of this setting, the device may update the home sensor position (limit.min) after a limit action is complete. This setting has no effect if the limit action is disabled (limit.home.action is 0).

limit.home.preset

The value home limit actions write to pos

Valid Values

-1,000,000,000 to 1,000,000,000

Scope

Axis

Write Access Level

Advanced

Persistence

Non-volatile

The value written to pos after home limit actions (specified by limit.home.action). This occurs as part of the home command, the tools gotolimit command, or after triggering the home sensor.

See the Physical Units section for information about unit conversions.

limit.home.state

The current state of the home limit sensor

Valid Values

0

Inactive

The sensor is not active.

1

Active

The sensor is active.

Scope

Axis

Write Access Level

Read-only

Persistence

Volatile

If no limit sensor is connected, this setting is always 0.

limit.home.triggered

Whether the axis has triggered the home sensor

Valid Values

0

The home sensor has not been triggered.

1

The home sensor has been triggered.

Scope

Axis

Write Access Level

Read-only

Persistence

Volatile

On power-up this setting is set to 0. The setting is set to 1 after a home limit action or a limit sensor command (such as the home command) completes.

limit.max

The maximum position the axis can move to

Valid Values

-1,000,000,000 to 1,000,000,000

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

The maximum position the axis can move to. The range of travel (limit.min to limit.max) is different for each product.

Depending on the value of limit.away.posupdate, limit.max may be automatically updated to the current position of the away sensor, if it exists, as part of a limit action.
Warning: Exercise caution when modifying this setting, since it is possible to set the range to a value greater than the physical limits of the axis.

See the Physical Units section for information about unit conversions.

limit.min

The minimum position the axis can move to

Valid Values

-1,000,000,000 to 1,000,000,000

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

The minimum position the axis can move to. The range of travel (limit.min to limit.max) is different for each product.
Depending on the value of limit.home.posupdate, it may be automatically updated to the current position of the home sensor as part of a limit action.
Warning: Exercise caution when modifying this setting, since it is possible to set the range to a value greater than the physical limits of the axis.

See the Physical Units section for information about unit conversions.

maxspeed

The maximum speed the axis moves at

Valid Values

1 to resolution × 16,384 for stepper motor products

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

When a movement command is issued, the axis accelerates up to and travels at this speed. If a movement command explicitly specifies a speed for the motion (for instance, move vel), maxspeed is ignored, and the axis travels at the speed specified in the command. During a home command, as well as anytime there is no reference

position (i.e. the ***No Reference Position*** (WR) warning flag is set), the axis travels at the lesser of limit.approach.maxspeed and maxspeed.
If maxspeed is changed while the axis is moving, the axis will immediately apply the new speed and continue to move to the target position.
See the Physical Units section for information about unit conversions.

motion

These settings contain miscellaneous controls and information relating to the motion of the device that are not covered elsewhere.

motion.accelonly

The acceleration used when increasing speed

Valid Values

0 to 2,147,483,647

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies the acceleration used for most motion trajectories. The deceleration is set separately using motion.decelonly. The accel setting and motion.accelonly setting have the same value. If the accel setting is changed, the motion.accelonly setting is also updated.

Warning: A value of 0 specifies infinite acceleration, which is often not physically feasible and may cause the motor to stall. It may also result in potential damage to the product and reduced lifespan, especially if the axis is under heavy load.

See the Physical Units section for information about unit conversions.

motion.accel.ramptime

The size of the trajectory averaging window for motion smoothing

Valid Values

0 to 50.0

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

The size, in units of milliseconds and to one decimal place, of the averaging window applied to the motion trajectory.

A value of 0.0 or 0.1 will disable trajectory averaging.

Using this setting the controller applies a moving average filter to the commanded trajectory, smoothing the motion profile, and limiting jerk (the 3rd derivative of position). Motion smoothing can help minimize vibration, audible noise, and impact loading during acceleration and deceleration. This is particularly relevant to applications with low system stiffness, such as liquid payloads, belt drive actuators, or non-rigid mounting surfaces. It can also be useful to improve drive lifetime for applications with many rapid starts and stops. Higher values increase the averaging time span and amount of smoothing, at the cost of longer trajectory execution time. For given maximum acceleration and speed limits, `motion.accel.ramptime` will increase the trajectory execution time by the value of the setting in milliseconds.

The relationship between `motion.accel.ramptime`, `motion.accelonly`, `motion.decelonly`, and the achieved value of jerk depends on the shape of the given trajectory. Typically, a trajectory consists of 3 segments: constant acceleration (defined by `motion.accelonly`), followed by constant velocity (defined by `maxspeed`), then constant deceleration (defined by `motion.decelonly`) to a stop, forming a trapezoidal velocity profile. If the averaging window size, `motion.accel.ramptime`, is shorter than the duration of each of the three segments then the jerk will be limited to:
jerk = max(`motion.accelonly`, `motion.decelonly`) × (Microstep Size) × 10,000,000 / (`motion.accel.ramptime` × 1.6384 s^3)

For short movements, where `maxspeed` is either not reached or reached for a shorter time than `motion.accel.ramptime`, the jerk will be limited to:
jerk = (`motion.accelonly` + `motion.decelonly`) × (Microstep Size) × 10,000,000 / (`motion.accel.ramptime` × 1.6384 s^3)

See the Physical Units section for information about unit conversions.

motion.busy

Whether an axis is `BUSY`.

Valid Values

0

The axis is IDLE

1

The axis is BUSY

Scope

Axis

Write Access Level

Read-only

Persistence

Volatile

Specifies whether an axis is BUSY.

See Status for a description of when an axis is BUSY.

motion.decelonly

The deceleration used when decreasing speed

Valid Values

0 to 2,147,483,647

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

Specifies the deceleration used for most motion trajectories. The acceleration is set separately using motion.accelonly. If the accel setting is changed, the motion.decelonly setting is updated to the same value.

Warning: A value of 0 specifies infinite deceleration, which is often not physically feasible. It may also result in potential damage to the product and reduced lifespan, especially if the axis is under heavy load.

See the Physical Units section for information about unit conversions.

motion.index.dist

The distance between consecutive index positions

Valid Values

1 to 2,000,000,000

Scope

Axis

Write Access Level

Normal

Persistence

Non-volatile

The distance between positions reachable with the `move index` command.
For linear devices, `motion.index.dist` should be less than or equal to `limit.max` and `limit.max` should be positive for `move index` commands to be accepted.
See the Physical Units section for information about unit conversions.

motion.index.num

The current index number

Valid Values

0 to 2,147,483,647

Scope

Axis

Write Access Level

Read-only

Persistence

Volatile

If the current position is an index position, the index number will be returned, calculated as `pos` / `motion.index.dist` + 1. If `pos` is not a multiple of `motion.index.dist`, this setting will return 0.
For linear devices, if `pos` is negative, this setting will return 0.

motor

These settings contain controls and information about the physical properties of the motor.

To ensure that the motor driver operates safely, the motor settings should match the values given in the motor's specifications.

motor.current.max

The maximum peak current the motor can safely withstand

Valid Values

0 to 214,748

Scope

Axis

Write Access Level

Read-only

Persistence

Non-volatile

The maximum current, in 20 mA increments, that the axis motor can safely withstand. The value in milliamps is given by motor.current.max × 20.

motor.inductance

The inductance of each motor phase

Valid Values

0.000000001 to 18,446,744,073.0

Scope

Axis

Write Access Level

Read-only

Persistence

Non-volatile

The inductance of each phase of the axis motor, in units of millihenries (supports up to 9 decimal places).

Introduced in 7.20

motor.resistance

The resistance of each motor phase

Valid Values

0.000000001 to 18,446,744,073.0

Scope

Axis

Write Access Level

Read-only

Persistence

Non-volatile

The resistance of each phase of the axis motor, in units of ohms (supports up to 9 decimal places).

Introduced in 7.20

parking.state

Whether the axis is parked

Valid Values

0

The axis is not parked.

1

The axis is parked.

Scope

Axis

Write Access Level
Read-only

Persistence
Non-volatile

For more information about parking, see `tools parking`.

pos

The current absolute position of the axis

Valid Values
-1,000,000,000 to 1,000,000,000

Scope
Axis

Write Access Level
Normal

Persistence
Volatile

The `pos` setting is an absolute measure (with reference to the limits) of the position at which the controller expects the axis to currently be.

On devices with encoders, the value of the actual measured position of the axis is `encoder.pos`, and the difference between the expected and measured positions is `encoder.pos.error`.

See the Physical Units section for information about unit conversions.

resolution

The number of microsteps per motor step

Valid Values
1 to 256

Scope
Axis

Write Access Level
Normal

Persistence
Non-volatile

Defines the number of microsteps per step of the stepper motor. A typical Bosch Rexroth Small Handling motorized axis uses a stepper motor with 200 steps per motor revolution and a default microstep resolution of 64, therefore it takes 12,800 (200 × 64) microsteps to make one full revolution of the motor.

See the Physical Units section for information about unit conversions.

When the resolution is updated, the following settings are updated:

- `knob.distance`

- knob.maxspeed
- limit.approach.maxspeed
- limit.away.preset
- limit.detect.decelonly
- limit.detect.maxspeed
- limit.home.preset
- limit.min
- limit.max
- maxspeed
- motion.accelonly
- motion.decelonly
- motion.index.dist

These settings are updated with their *default* values (scaled to match the new resolution); the previous setting values are cleared.

After changing the resolution, the ***No Reference Position*** (WR) warning flag is set, indicating that the axis should be homed before any movement is performed.

Stored positions (see tools storepos) are not updated when the resolution is changed.

scope

These settings control the timing of scope captures.

See the scope commands section for more information about scope captures.

scope.delay

The delay before a scope capture is started

Valid Values

0.0 to 429,496,729.5

Scope

Device

Write Access Level

Normal

Persistence

Volatile

The time delay, in units of milliseconds and to one decimal place, after receiving the scope start command before data capture starts.

In firmware versions 7.08 and above, if a scope capture is in progress, setting scope.delay is rejected with STATUSBUSY.

scope.timebase

The time between consecutive scope samples

Valid Values

0.1 to 429,496,729.5

Scope
Device

Write Access Level
Normal

Persistence
Volatile

The time between consecutive scope samples, in units of milliseconds and to one decimal place.

In firmware versions 7.08 and above, if a scope capture is in progress, setting `scope.timebase` is rejected with `STATUSBUSY`.

stream

These settings contain information about streamed motion.

See the `stream` commands section for more information about streamed motion.

stream.numbufs

The number of stream buffers available

Valid Values
0 to 4,294,967,295

Scope
Device

Write Access Level
Read-only

Persistence
Non-volatile

The total number of stream buffers available.

Introduced in 7.05

stream.numstreams

The number of simultaneous streams that can be set up

Valid Values
0 to 4,294,967,295

Scope
Device

Write Access Level
Read-only

Persistence
Non-volatile

The number of streams that can be active simultaneously, whether they are initiated with `stream setup live` or `stream setup store`.

Introduced in 7.05

system

These settings contain controls for, and information about, system-wide properties.

system.access

The access level of the user

Valid Values

1
Normal

2
Advanced

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

Some commands require an access level of 2 (advanced) as they can potentially cause damage to the controller, motor, or device. To use those commands and settings, set system.access to access level 2 (advanced).

All settings are readable regardless of their access levels.

system.axiscount

The number of axes controlled by the device

Valid Values

1 to 4

Scope

Device

Write Access Level

Read-only

Persistence

Non-volatile

This setting returns the number of axes controlled by the device.

system.current.max

The maximum permitted supply input current

Valid Values

0.000 to 65.535

Scope

Device

Write Access Level

Read-only

Persistence

Non-volatile

Specifies the maximum supply current, in units of amps, that may be drawn by the device for powering its drivers. Supply current that is shared with daisy-chained devices does not contribute to this limit.

Drawing currents in excess of this value may trigger a ***Current Inrush Error*** (FC).

Introduced in 7.27

system.led.enable

The control for whether the indicator LEDs are enabled

Valid Values

0

Disabled

1

Enabled

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

A value of 0 disables all front panel LEDs on the device.

system.serial

The serial number of the device

Valid Values

0 to 4,294,967,295

Scope

Device

Write Access Level

Read-only

Persistence

Non-volatile

Every Bosch Rexroth product has a unique serial number.

Example: Reading the device's serial number

Read the device's serial number

```
/get system.serial↵
@01 0 OK IDLE -- 35542
```

The serial number is 35542.

system.temperature

The processor temperature

Valid Values

0 to 150

Scope

Device

Write Access Level

Read-only

Persistence

Volatile

Specifies the temperature, in units of degrees Celsius, of the processor (inside the controller). The ***Temperature High*** (WT) warning flag is set if the processor becomes too hot to guarantee normal operation.

See also driver.temperature.

Introduced in 7.06.

Example: Reading the temperature of the processor

```
/get system.temperature↵
@01 0 OK IDLE -- 53.5
```

The processor temperature is presently 53.5°C.

system.voltage

The power supply voltage

Valid Values

0 to 150

Scope

Device

Write Access Level

Read-only

Persistence

Volatile

The voltage the power supply is providing, in units of volts. It is normal for the voltage to differ slightly from the power supply's rated voltage.

Example: Reading the power supply voltage

Read the voltage provided by the power supply:

```
/get system.voltage
@01 0 OK IDLE -- 47.1
```

The power supply is providing 47.1 VDC to the device.

trigger

These settings contain information about triggers.

See the trigger commands section for more information about triggers.

trigger.numactions

The number of actions per trigger

Valid Values

0 to 4,294,967,295

Scope

Device

Write Access Level

Read-only

Persistence

Non-volatile

The total number of actions per trigger.

Introduced in 7.06

trigger.numtriggers

The number of triggers on the device

Valid Values

0 to 4,294,967,295

Scope

Device

Write Access Level

Read-only

Persistence

Non-volatile

The total number of triggers on the device.

Introduced in 7.06

user

These settings are available for users to store information on the device.

Some of the settings are stored in non-volatile memory, meaning they will persist after a `system reset` or power-up. Some are instead stored in volatile memory, meaning they will reset to 0 after a `system reset` or power-up. Volatile memory is preferable to use for values that will be overwritten with high frequency. The device does not reference these values and their meaning is entirely user-defined.

To store more flexible string-based data on the device, see the `storage` commands. Note that data stored with the `storage` commands are not accessible via other commands, such as `get`, `set`, `triggers`, etc.

`user.data.0`

Non-volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

`user.data.1`

Non-volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

`user.data.2`

Non-volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.data.3

Non-volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.data.4

Non-volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.data.5

Non-volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.data.6

Non-volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.data.7

Non-volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.data.8

Non-volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level
Normal

Persistence
Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.data.9

Non-volatile storage for user data

Valid Values
-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope
Device

Write Access Level
Normal

Persistence
Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.data.10

Non-volatile storage for user data

Valid Values
-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope
Device

Write Access Level
Normal

Persistence
Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.data.11

Non-volatile storage for user data

Valid Values
-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope
Device

Write Access Level

Normal

Persistence

Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.data.12

Non-volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.data.13

Non-volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level

Normal

Persistence

Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.data.14

Non-volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level

Normal

Persistence
Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.data.15

Non-volatile storage for user data

Valid Values
-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope
Device

Write Access Level
Normal

Persistence
Non-volatile

A setting for storing non-volatile user data.

Introduced in 7.23.

user.vdata.0

Volatile storage for user data

Valid Values
-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope
Device

Write Access Level
Normal

Persistence
Volatile

A setting for storing volatile user data.

Introduced in 7.23.

user.vdata.1

Volatile storage for user data

Valid Values
-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope
Device

Write Access Level
Normal

Persistence

Volatile

A setting for storing volatile user data.

Introduced in 7.23.

user.vdata.2

Volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level

Normal

Persistence

Volatile

A setting for storing volatile user data.

Introduced in 7.23.

user.vdata.3

Volatile storage for user data

Valid Values

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Scope

Device

Write Access Level

Normal

Persistence

Volatile

A setting for storing volatile user data.

Introduced in 7.23.

version

These settings contain firmware version information.

version

The firmware version of the device

Valid Values

7.01 to 7.99

Scope

Device

Write Access Level
Read-only

Persistence
Non-volatile

Example: Reading the firmware version

Read the device's firmware version:

```
/get version
@01 0 OK IDLE -- 7.01
```

The device has firmware version 7.01 installed.

`version.build`

The build number of the device's firmware

Valid Values
0 to 4,294,967,295

Scope
Device

Write Access Level
Read-only

Persistence
Non-volatile

The firmware build number uniquely identifies the firmware installed on the device. This is useful in the rare cases where multiple firmware builds share the same `version`. It is not normally necessary to consult the build number.

Example: Reading the build number

Read the firmware build number:

```
/get version.build↵
@01 0 OK IDLE -- 203
```

The build number is 203.

Advanced Features

The following advanced features and commands are only required in rare use cases.

Checksums

Checksums can be included in commands and messages in order to check if they were corrupted. Checksum values are generated using the Longitudinal Redundancy Check (LRC) method. They only provide a method for error detection, and do not provide a method for error correction.

A checksum in a message will be the last three bytes before the footer. The first byte of the checksum is a colon (`:`). The colon character is reserved for identifying the checksum, and should not appear elsewhere in message data. The following two bytes are the LRC checksum value of the message (excluding the header character) in hexadecimal format.

A device will verify a checksum if it is included in a command. If the checksum does not match the received command, no action will be taken, no reply will be sent, and the yellow LED(s) on the device will blink twice per second until the next command is received.

A device will include checksums in responses depending on the value of `comm.checksum`.

The LRC of a byte sequence is defined by the following algorithm:

```
Set LRC = 0
For each byte b in the buffer
do
    Set LRC = (LRC + b) AND 0xFF
end do
Set LRC = (((LRC XOR 0xFF) + 1) AND 0xFF)
```

For example if the message is:

```
/01 tools echo↵
```

the checksum is:

```
SUM = 48 + 49 + 32 + 116 + 111 + 111 + 108 + 115 + 32 + 101 + 99 + 104 + 111
SUM = 1137

LRC = (((SUM & 0xFF) ^ 0xFF) + 1) & 0xFF
LRC = 143
```

```
LRC = 0x8F
```

and the final message is:

```
/01 tools echo:8F↲
```

Verification

To verify a message checksum the 8-bit sum of all the bytes in the message is calculated and added to the transmitted checksum, which has been converted to an integer. The message is valid when the 8-bit result of the sum is zero. The colon in the message is only used as a separator and is otherwise ignored. Using the example above:

```
(1137 + 0x8F) & 0xFF = 0
```

Example Checksum Code

The following examples show how to calculate a message checksum and verify a received message in several languages.

Example: C

Calculating a message checksum:

```
#define CSUM_NOTPRESENT (-2)
#define CSUM_NOSPACE    (-1)
#define CSUM_FAIL       (0)
#define CSUM_OK         (1)

int csum_message(char *message, unsigned int max_length)
{
    unsigned char c_sum = 0;  // assuming 8-bit chars
    char *p = message + 1;    // skip the type character

    // is there room for the checksum?
    if (strlen(message) + 6 < max_length)
    {
        while (*p != 0x00)
        {
            // calculate the checksum
            c_sum += (unsigned char)*p++;
        }
```

```
        // negate, increment and char size truncates
        c_sum = ~c_sum + 1;

        // add the checksum to the message
        sprintf(p, ":%02X\r\n", c_sum);
        return CSUM_OK;
    }

    return CSUM_NOSPACE;
}
```

Verifying a message checksum:

```
int csum_verify(char *message)
{
    unsigned char c_sum = 0;  // assuming 8-bit chars
    char *p = message + 1;    // skip the type character

    while (*p != 0x00)
    {
        // calculate the sum
        c_sum += (unsigned char)*p++;

        // found the checksum field, process
        if (*p == ':')
        {
            // convert the sent checksum
            c_sum += strtol(++p, NULL, 16);
            return ((c_sum == 0) ? CSUM_OK : CSUM_FAIL);
        }
    }

    return CSUM_NOTPRESENT;
}
```

Example: Python

Calculating a message checksum:

```
def csum_message(msg):

    c_sum = 0

    for c in msg[1:]:
        # calculate the sum of the message to be transmitted
        c_sum += ord(c)

    # take the ones compliment and truncate to 8 bits
    c_sum = (256 - (c_sum & 0xFF)) & 0xFF
```

```
    # return the full message
    return '%s:%02X\r\n' % (msg, c_sum)
```

Verifying a message checksum:

```
def csum_verify(msg):

    c_sum = 0

    if msg.find(':') < 0:
         # return nothing if the checksum isn't present
        return None

    # separate out the message and checksum
    x_msg, x_sum = msg.split(':', 1)

    for c in x_msg[1:]:
        # recalculate the sum of the received message
        c_sum += ord(c)

    # add in the received checksum and truncate to a 8-bit result
    c_sum = (c_sum + int(x_sum, 16)) & 0xFF

    # return true if the message passed checksum verification
    return (c_sum == 0)
```

Message IDs

A message ID in an optional field that can be included in any command, and causes responses (including reply and info responses) to the command to include the same message ID. Include the message ID in the command structure after the device address and axis number and before the command. Both the device number and axis number must be explicit to include a message ID. The valid range of message IDs is 0 to 99. The message ID in responses will be 2 digits, and will also follow the device address and axis number, and precede the reply flag.

If the message ID in a command is `--`, then the device will not send any response to that command.

Typically message IDs are not needed, because the response to a command can be easily identified as it immediately follows that command. However, there are a few cases where they are useful. One case is where commands to devices are sent from multiple sources in a daisy-chain, for example a computer and a microcontroller, so the

source of a response is not definite. Another case is when responses from devices are not desired.

Example: Including a message ID

Specify a message ID of 8 in a command to axis 1 of device 2:

```
/2 1 8 move rel 10000↵
@02 1 08 OK IDLE -- 0
```

Example: Sending a command with a message ID to multiple devices

Send a command with message ID 25 to all devices and axes:

```
/0 0 25 stop↵
@01 1 25 OK BUSY -- 0
@01 2 25 OK IDLE -- 0
@02 1 25 OK BUSY -- 0
```

To include a message ID while sending a command to all devices and axes in the chain, use `0` for the device address and axis number.

Example: Suppressing responses

Set a command's message ID to `--` to request no response:

```
/1 1 -- set maxspeed 200000↵
```

Line Continuation

In nearly all cases an ASCII protocol message fits within a single packet. A packet can be at most `comm.packet.size.max` characters long, begins with a ***message type*** character (`/`, `@`, `#`, or `!`), and is terminated with a newline character or characters (↵). For example the following are ASCII protocol messages transmitted in a single packet:

```
/get pos↵
@01 1 OK IDLE -- 0
```

The command `get pos` is sent to the device in the first packet and a reply is sent back in the second.

For firmware versions 7.11 and below, if a message could not fit into a single packet it was truncated, but there were no meaningful commands where this would happen. However, for firmware versions 7.12 and above, there are cases where the device may produce replies or info messages that cannot fit into a single packet. When this happens the device will divide the message across multiple packets, sending the remainder of the message in subsequent info packets. The message will be divided at the last whitespace possible while still adhering to the `comm.packet.size.max` character packet limit. To indicate that this has happened, the device will append the line continuation character, `\`, to every packet that does not complete the message, and prefix each subsequent info packet that continues the message with `cont`.

For instance, reading a servo parameter from all axes at once on an X-MCC4, which has four axes, may produce a line continuation message:

```
/servo live get kd↵
@01 1 OK IDLE -- 11111.123456789 22222.123456789 33333.123456789\
#01 1 cont 44444.123456789
```

If there is no whitespace at which to split the message and still adhere to the `comm.packet.size.max` character packet limit, the message or portion of the message will be truncated and the ***Value Truncated*** (`NT`) warning flag will be set.

For firmware versions 7.26 and above, Commands also support line continuation. The continuation format for commands is similar to, but not exactly the same as, the continuation format for replies and info messages. To indicate that line continuation is in use for a command, the sender must append the line continuation character, `\`, to every packet that does not complete the message, and prefix every subsequent command packet that continues the message with `cont` followed by an integer. The integer should be 1 for the second packet in the message (the first continuation packet) and increment with each subsequent packet. At most `comm.command.packets.max` packets can be used for each command message. All the packets in a message must be addressed to the same axis number and have the same message ID. The device only replies once all packets have been received.

Example: Splitting a command into four packets

```
/1 0 tools\↵
/1 0 cont 1 echo\↵
/1 0 cont 2 hello\↵
/1 0 cont 3 world↵
@01 0 OK IDLE -- hello world
```

The actual command, after the device reassembles the packets, is `tools echo hello world`.

Example: Incorrect line continuation in a command

```
/1 0 tools echo\↵
/1 0 cont 2 hello world↵
@01 0 RJ IDLE -- BADSPLIT
```

The message is invalid because `cont 2` appeared in the first continuation packet where `cont 1` was expected.

Example: Line continuation in commands with checksums

```
/1 0 tools echo\:13↵
/1 0 cont 1 abcd:B0↵
@01 0 OK IDLE -- abcd
```

Each packet has its own checksum. The `\` characters appear before the checksums.

Appendix A: Available Serial Ports

Windows

To find available serial devices on Windows:

1. Start the **Device Manager** by opening **Search** or **Run** from the **Start Menu**, typing `Device Manager`, and pressing **Enter**
2. Expand the **Ports (COM & LPT)** category
    - In this example there are two serial ports available (**COM3** and **COM8**), which are both USB adaptors.
    - To determine which one corresponds to the USB serial cable connected to the Bosch Rexroth device, unplug and plug in the cable, to see which one appears and disappears.

Linux

To find available serial devices on Linux:

1. Find all serial ports
   - Open a terminal and execute the following command:

     ```
     dmesg | grep -E ttyU\?S↵
     ```

   - The available ports should be listed similar to the following:

     ```
     [ 2.029214] serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
     [ 2.432572] 00:07: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
     [ 2.468149] 0000:00:03.3: ttyS4 at I/O 0xec98 (irq = 17) is a 165
     50A
     ```

     ```
     [ 13.514432] usb 7-2: FTDI USB Serial Device converter now attach
     ed to ttyUSB0
     ```

     In this example, there are 3 serial ports available: `ttyS0`, `ttyS4`, and `ttyUSB0` (a USB adaptor).
   - To determine which one corresponds to the USB serial cable connected to the Bosch Rexroth device, try repeating the command with and without the cable connected to the computer, to see which one appears and disappears.
2. Check the port permissions
   - Using the ports found in the example above, execute the following command:

     ```
     ls -l /dev/tty{S0, S4, USB0}↵
     ```

   - The permissions for each port should be listed similar to the following:

     ```
     crw-rw---- 1 root dialout 4, 64 Oct 31 06:44 /dev/ttyS0
     crw-rw---- 1 root dialout 4, 68 Oct 31 06:45 /dev/ttyS4
     ```

     ```
     crw-rw---- 1 root dialout 188, 0 Oct 31 07:58 /dev/ttyUSB0
     ```

     This shows that a user must be `root` or a member of the `dialout` group to access the ports.
3. Check and update group permissions
   - List the groups the current user is in with the following command:

     ```
     groups↵
     ```

   - The output will be similar to the following:

     ```
     adm cdrom sudo dip plugdev users lpadmin sambashare
     ```

     Notice that `dialout` is not in the list
   - If necessary, add the current user to the `dialout` group with the following command:

```
sudo adduser $USER dialout↵
```

Note: Group membership will not take effect until the next logon

macOS

To find available serial devices on macOS:

1. Open a terminal and execute the following command:

```
ls /dev/cu.*serial*↵
```

   - The available ports should be listed similar to the following:

```
/dev/cu.usbserial-FTB3QAET
```

```
/dev/cu.usbserial-FTEJJ1YW
```

     In this example, there are 2 serial ports available, both of which happen to be USB adaptors.
   - There may be other devices that match this query, such as keyboards or some web cameras. To determine which one corresponds to the USB serial cable connected to the Bosch Rexroth device, try repeating the command with and without the cable connected to the computer, to see which one appears and disappears.

General notes

▶ Vertical Installation

The drive is not self-locking, so in vertical use a load may cause movement. Vertical use is only recommended for cases where the load falling is acceptable, safe, and will not cause damage.

Vertical use will exert additional force on the drive components, and may limit service life.

Intended use

▶ The product is intended exclusively for use moving and positioning loads.

▶ The product is intended exclusively for professional use and not for private use.

▶ It's intended that users read and understand all documentation to the product before operation.

Misuse

▶ Use of the product in any other way than as described under "Intended use" is considered to be misuse and is not permitted. If products are installed or used in unsuitable safety-relevant applications, this misuse may lead to personal injury and/or damage to property.

▶ The product may only be used in safety-relevant applications if this use has been expressly specified and permitted in the product documentation.

▶ The product is not intended for users to service. Any unauthorized disassembly of the product is misuse and voids any warranty.

▶ Operating in an environment that is unsuitable for the product is misuse. Unsuitable operating environments include (but are not limited to):

- wet environments
- high-humidity environments
- extreme high or low temperatures
- extreme high or low pressures
- radiation
- corrosive environments
- in proximity to strong magnets

▶ Bosch Rexroth AG will not accept any liability for injury or damage caused by misuse of the product. The risks associated with any misuse of the product shall be borne by the user alone.

▶ Misuse of the product includes:

- transporting people
- use in medical devices or applications

General safety instructions

▶ Safety rules and regulations of the country in which the product is used must be complied with.

▶ All current and applicable accident prevention and environmental regulations must be adhered to.

▶ The product may only be used when it is in technically perfect condition.

▶ The technical data stated in the product documentation must be complied with.

▶ The product must not be put into service until it has been verified that the final product (for example a machine or system) into which the product has been installed complies with the country-specific requirements, safety regulations and standards for the application.

▶ Pinch prevention and labeling are recommended.

▶ Motors can generate heat, and access to them should be limited during operation.

▶ At high travel speeds a certain amount of noise is caused by the product. If necessary, appropriate measures should be taken to protect hearing.

▶ Special safety requirements for specific sectors (e.g. foodstuffs, chemicals, semiconductor processing) as provided for in laws, directives and standards must be complied with.

Bosch Rexroth Corporation
Corporate Headquarters
14001 South Lakes Drive
Charlotte, NC 28273
Telephone (800) REXROTH
(800) 739-7684
info@boschrexroth-us.com
www.boschrexroth-us.com

Find your local contact person here:
www.boschrexroth-us.com/contactus

USL00073/2022-11