EVA Robotics Evodrive ST-17

Stepper Servo Controller

$995.00

In Stock

Qty Available: 1

New From Surplus Stock

Open Web Page

https://www.artisantg.com/95936-2

EVA ROBOTICS

EvoDrive

PROGRAMMERS MANUAL

EVA Robotics | FW-UM001 Rev E | April 2013

Disclaimer

The EvoDrive Programmers Manual is provided AS IS, without warranty of any description.

EVA Robotics Pty Ltd does not warrant, guarantee, or make any express or implied representations regarding the use of this manual in terms of correctness, accuracy, reliability, currency, or otherwise.

EVA Robotics Pty Ltd reserves the right to revise the content of this manual without obligation to notify any person or organisation of such revision or changes.

Having said that, EVA Robotics takes great pride in the quality of its products and we believe that great documentation is fundamental. If you have any feedback regarding this manual, we would love to hear about it.

Manufactured by

EVA Robotics Pty Ltd
ACN 139 161 112

Web: www.evarobotics.com

1. Important Information

1.1 Intended Use

The EvoDrive series of products are intended to control and drive sub-horsepower stepper motors. They are intended to be used as components in larger systems or products.

The EvoDrive products have been designed to all applicable safety standards but they are not intended to implement safety functions. The risk of unexpected or un-braked movements cannot be sufficiently mitigated without additional safety provisions.

For this reason, personnel must never be in any danger zone actuated by an EvoDrive product unless additional safety equipment prevents personal danger. This applies to operation of, as well as service and maintenance of, the product or system.

In all cases, the applicable safety regulations and the specified operating conditions, such as environmental conditions and specified technical data must be observed.

The EvoDrive may be one component of more complex systems, all machinery, EMC and safety regulations that apply to the system as a whole must be observed before installation and operation.

To prevent personal injury and damage to property, damaged EvoDrive products must not be installed or operated.

Changes and modifications to the EvoDrive are prohibited and will void any warranty and liability.

The EvoDrive is not intended for use in explosive environments.

1.2 Qualification of personnel

The safe and successful integration of the EvoDrive requires a certain level of competency in designing and working with motor drive technologies.

Only engineers/technicians who are familiar with and understand the contents of the EvoDrive documentation are authorised to integrate EvoDrive controllers.

The system designers must be able to identify, assess, and control the potential risks inherent in the operation of mechatronic systems.

The designers must be familiar with the relevant standards and regulations that apply to the end product or system.

1.3 Safety Labels

This manual may contain warnings and cautions which, when properly followed, can prevent personal injury and damage to the EvoDrive.

Caution. This label will be used to bring attention to a general caution or warning that provides important information to protect against injury to an operator or damage to equipment.

Electrical Hazard. This label will be used to warn the reader of an electrical hazard.

Mechanical Hazard. This safety label will be used to warn the reader of a mechanical hazard.

1.4 General Safety Instructions

Hazardous Voltages

While the EvoDrive does not operate at hazardous voltages, the use of open frame power supplies, or non-standard wiring practices, such as routing EvoDrive cables together with high voltage cables, may expose the EvoDrive to hazardous voltages under fault conditions.

This will result in damage to the EvoDrive and may result in injury or death to the operator.

Loss of Control

The system designer must assess the hazards and risks inherent in safety critical functions, and implement controls to mitigate the risk.

Some examples of failure modes that should be considered include:

- Unexpected delays in commands or responses
- Loss of power and restart
- Failure of sensors and signals
- Over-Temperature cut-outs and fire hazards

Failure to observe the relevant safety and risk management standards may result in death or injury to the operator.

High Temperatures

The EvoDrive will typically run much cooler than other stepper motor controllers when in closed loop mode.

When operating in an open-loop mode it is still possible to generate high temperatures when driving or holding the motor with high current levels.

In this situation, the stepper motor and EvoDrive casing temperatures can both exceed 60°C.

Contents

2. Introduction

This programmer's manual details the commands and control systems used on EvoDrive stepper motor controllers. This manual should be read and understood together with the EvoDrive Integration Manual relevant to your model.

The EvoDrive family of stepper motor controllers fundamentally change how stepper motors are driven and therefore how they are used.

As an open-loop controller, the EvoDrive provides the equivalent of 512 micro-step accuracy with exceptionally smooth current control. This results in incredibly quiet and smooth motion control.

As a closed-loop controller, the EvoDrive moves away from traditional drive technology and transforms the humble stepper motor into a true servo motor, with a high dynamic bandwidth and exceptional efficiency.

This is achieved through a series of cascaded control systems, some traditional, and some proprietary which together form what we call the Phase Vector Control System.

The EvoDrive command set provides a practical balance of power and simplicity.

Intuitive use of set points and intelligent design removes the need for disparate operating modes. The ability to change settings and operating states on the fly allows system designers to achieve complex operations with ease.

Short mnemonic commands for common operations save on control bandwidth. Descriptive commands for settings and programming allow easy debugging and programming.

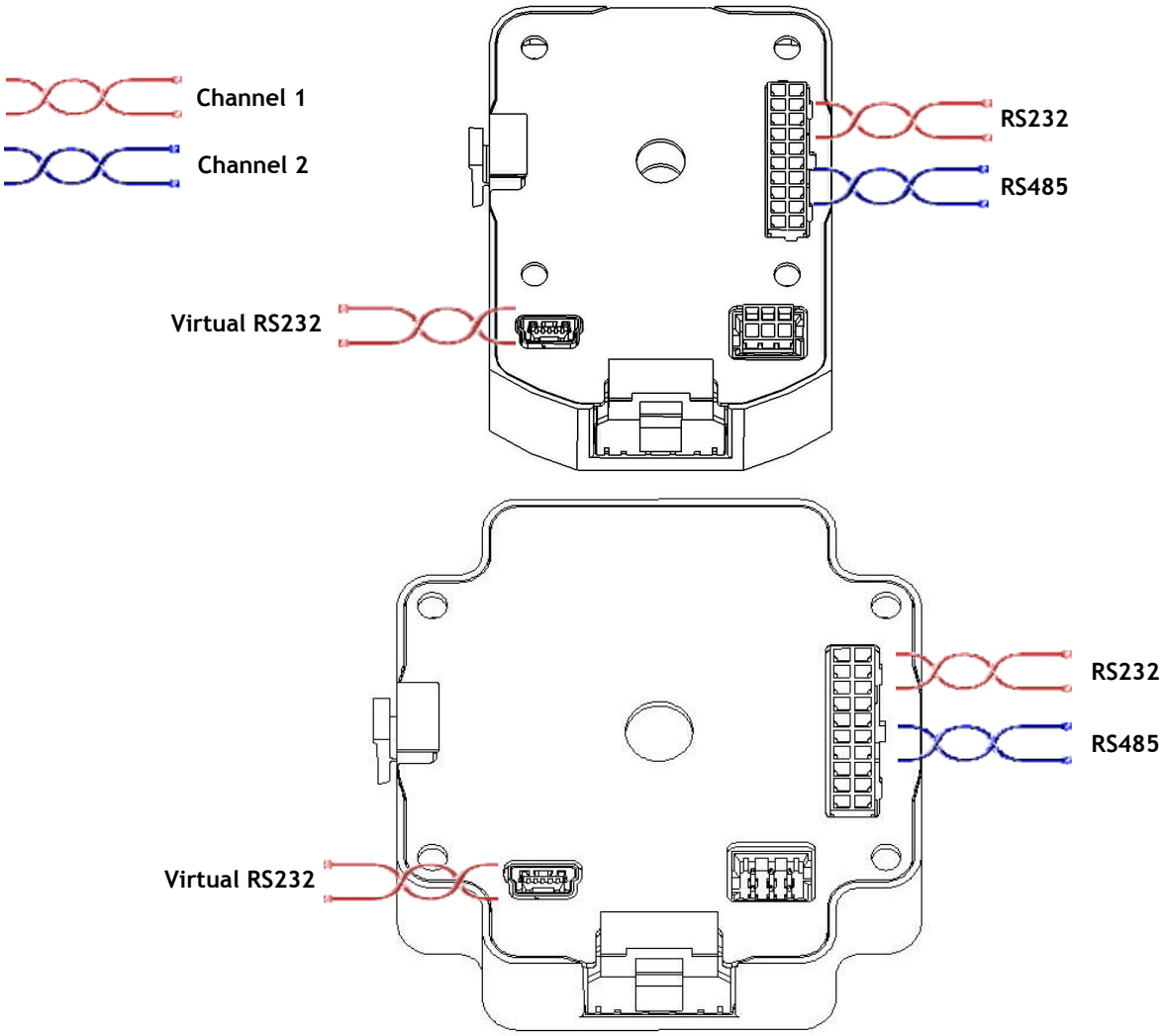
3. Communications

3.1 EvoDrive ST-17 / EvoDrive ST-23

The EvoDrive ST-17 and EvoDrive ST-23 both support two channels of simultaneous communications – Channel 1 and Channel 2. Each channel supports different communication interfaces.

While Channel 1 and Channel 2 can be used simultaneously, the interfaces within a channel are mutually exclusive. For example, you can use the RS485 and RS232 interfaces simultaneously, but you cannot use the RS232 and Virtual RS232 interfaces at the same time.

All interfaces support the same commands and syntax.



Channel 1 – RS232

RS232 is a standard serial interface with RX, TX, and GND pins accessible via the I/O header.

This interface provides simple connection to embedded devices over a cable, as well as any PC or laptop with a serial port.

Channel 1 – Virtual RS232

With the proper drivers installed, the USB port appears as a virtual COM port on a PC or laptop. The latest drivers are available for download from www.evarobotics.com.

The virtual RS232 interface allows full control of the EvoDrive when powered by an external power supply. This provides a very fast development path as well as a powerful debugging tool.

When the EvoDrive is not powered on, the unit can still be configured or programmed via USB. This is particularly useful in a production environment, or to ensure a safe means of servicing the EvoDrive in the field.

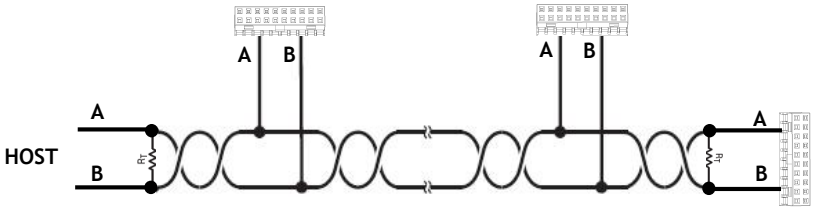
Channel 2 – RS485

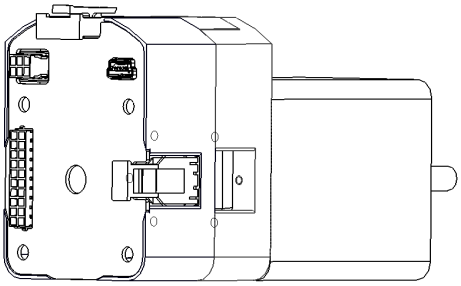
The RS485 interface is a half-duplex RS485 bus with A, B, and GND pins accessible via the I/O header.

The RS485 interface allows up to 34 EvoDrive units to be daisy chained together. The maximum communications rate over the RS485 bus is 250kbps.

The RS485 bus should be implemented as a twisted pair cable with terminating resistors at each end (100Ω – 120Ω). With proper cable termination, cable lengths up to 350 metres are possible without loss of data integrity.

An example bus layout is shown on the next page.





When using both communications channels care must be taken that you do not send conflicting commands. The most recent command on either channel will take priority.

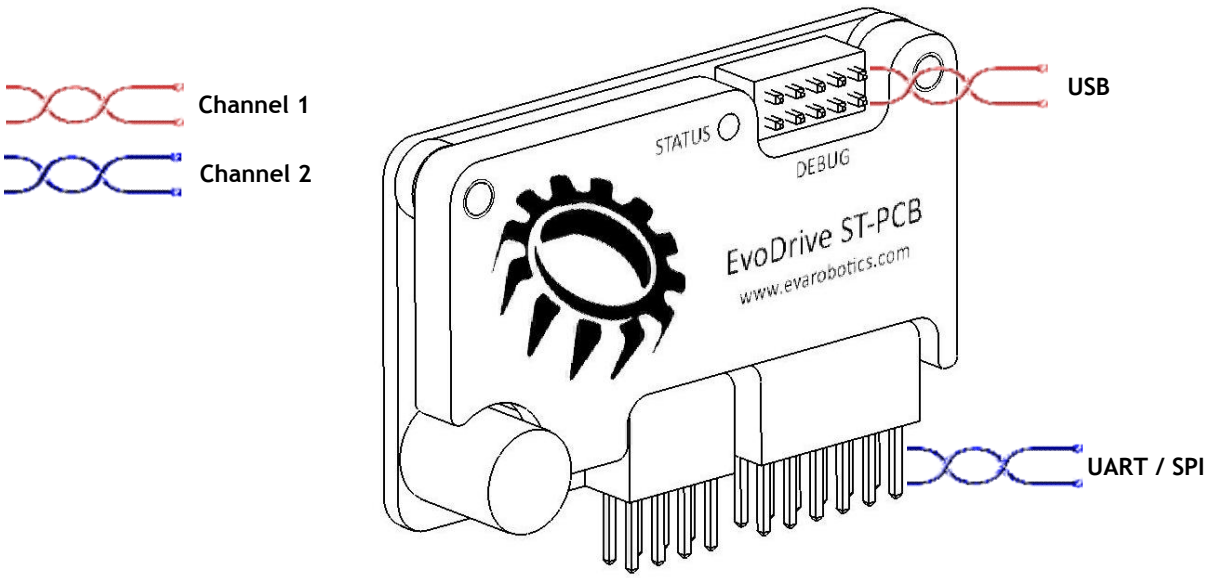
It is best practice to use one channel for all communications, or use movement commands on one channel and only use the other channel for queries and updates

3.2 EvoDrive ST-PCB

The EvoDrive ST-PCB supports two channels of simultaneous communications – Channel 1 and Channel 2.

Channel 1 is dedicated to the USB Debug port. This allows the EvoDrive to be monitored and configured in-situ, even while the Host is actively controlling it.

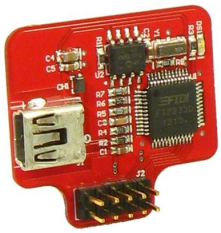
Channel 2 is the embedded communications channel between the Host and the EvoDrive ST-PCB and can be configured as UART (232/485) or as SPI.



When using both communications channels, care must be taken that you do not send conflicting commands. The most recent command on either channel will take priority.

It is best practice to use one channel for all communications, or use movement commands on one channel and only use the other channel for queries and updates.

Channel 1 – USB



The USB Debug port is a dedicated comms channel that can be accessed via the USB Debugger (FW-A103).

The USB Debugger provides a virtual RS232 port, which accepts all the commands detailed in this manual. All commands are processed independently of Channel 2, which allows the EvoDrive to be monitored and configured using EvoLink or any serial terminal program while the EvoDrive is in situ.

Channel 2 – UART

The UART interface is a standard UART connection at CMOS voltage levels. The RX, TX, and GND pins are accessible via the header pins. This interface is designed for direct interfacing with embedded devices.

The UART interface supports both the ASCII and Binary command formats detailed in this manual.

The UART interface is enabled by default. When using the UART interface the digital control pins are also accessible for use. When the EvoDrive ST-PCB is configured for SPI communications, the UART and digital input pins are disabled.

The default UART interface parameters are

- Baud Rate **9600**
- Data Bits **8**
- Stop Bits **1**
- Parity **None**
- Max input buffer **64 characters**

All standard BAUD rates up to 115200 bps are supported and can be configured via EvoLink or the **BAUD** command.

When configured for RS485 communications, Digital Control pin 4 is dedicated to turn-around control and can be used to drive the enable pin of a RS485 transceiver.

Channel 2 – SPI

The SPI interface supports the full-duplex synchronous serial protocol as defined by Motorola. The SPI interface is intended for high-speed embedded communications. It supports Binary format commands only.

The MISO, MOSI, SCLK, and CS lines are accessible via the header pins. The SPI interface shares these pins with the digital input pins, therefore when using the SPI interface the digital input and UART interfaces are disabled.

The SPI interface uses 8-bit data frames at clock speeds up to 4MHz. The SPI frame format is configurable with two parameters, SPI Polarity and SPI Phase.

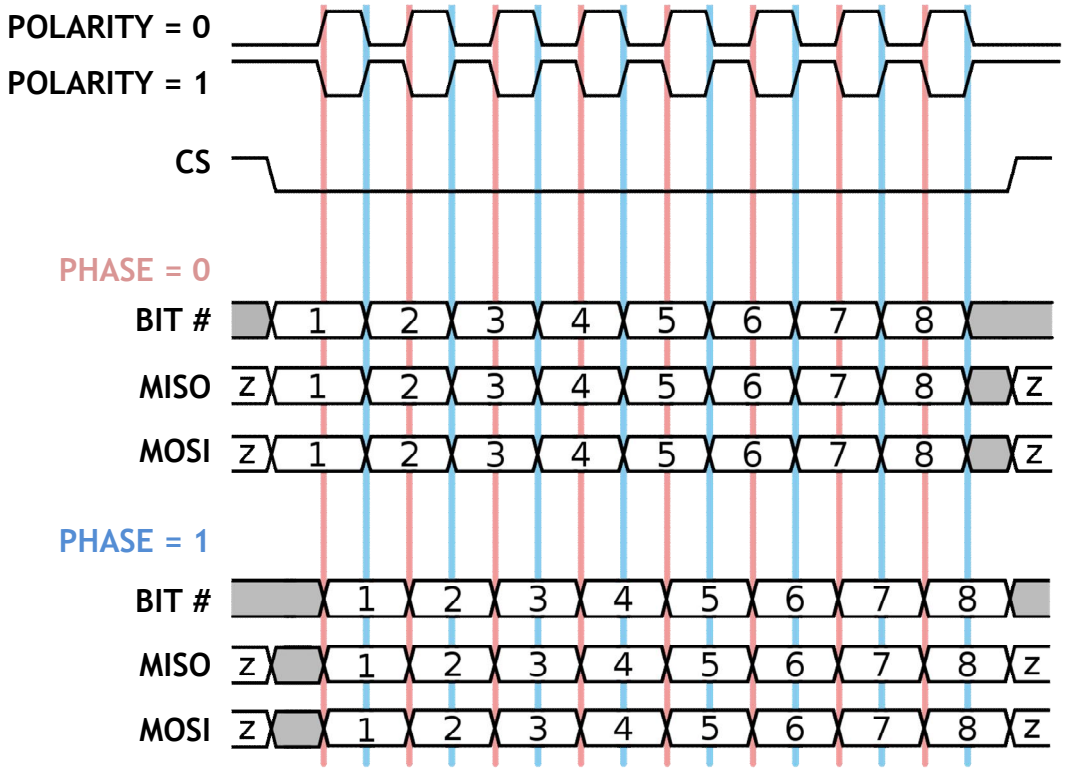
SPI Polarity, the equivalent to Motorola's CPOL bit, controls the polarity of the clock signal.

- When Polarity = 0, the clock idle state is low.
- When Polarity = 1, the clock idle state is high.

SPI Phase, the equivalent to Motorola's CPHA bit, controls which clock edge latches the data bits.

- When Phase = 0, data is read on the first clock edge
- When Phase = 1, data is read on the second clock edge

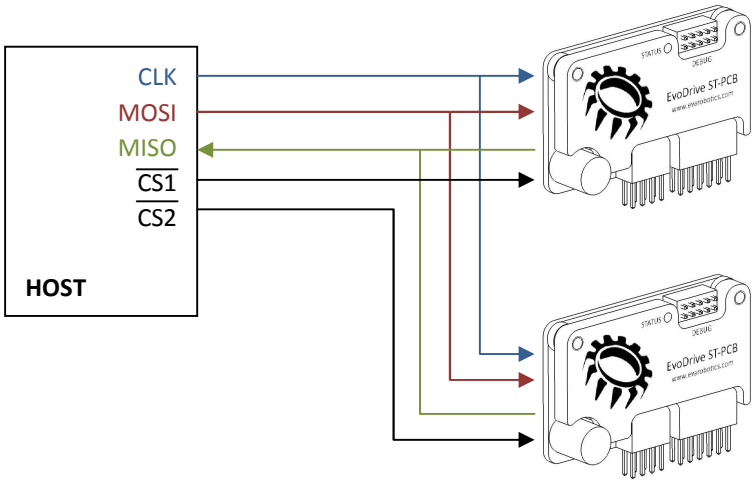
The SPI Polarity and Phase values are configured via EvoLink or via the **CPOL** and **CPHA** commands.



SPI Multi-drop bus

The SPI interface allows for multiple slave EvoDrive ST-PCB modules to be controlled from a single master using a parallel bus configuration.

The MISO and MOSI signals are routed to each EvoDrive ST-PCB. The CS line (active low) selects the EvoDrive that the host wishes to control. As a safety measure, each EvoDrive can also be configured with a unique address which confirms the command is for the signalled EvoDrive.



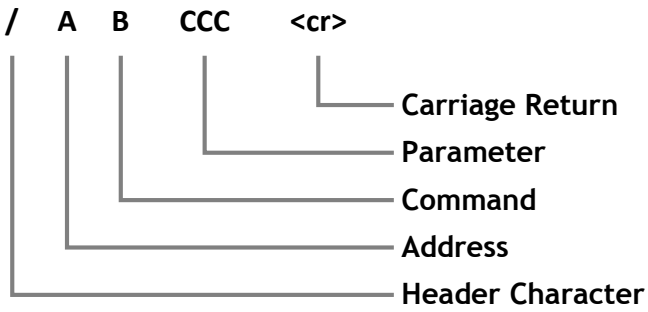
4. ASCII Command Format (Terminal Mode)

4.1 Command Structure

All EvoDrive products employ a common ASCII based command set. The commands are not case sensitive and spaces are ignored to make terminal use easy.

The commands are designed to achieve a balance between easy programming, and keeping command latency to a minimum.

The basic command structure is shown below



Header (/)

The header character is used to start a new command string.

If sent in the middle of a command then all previous characters are discarded.

Address

The address is used to identify the EvoDrive when communicating over a bus or network. This can be any alphanumeric character (**1…9**, and **A…Y**), allowing up to 34 devices on one bus.

- **0** is reserved for the host.
- **Z** is a broadcast address; the command will be executed by all devices.

Command

The commands can be short, 1-2 character mnemonics for commonly used functions, or longer command words for less common configuration actions.

Parameters

Parameters are integer numbers immediately following a command. Not all commands use parameters.

Where a command uses more than one parameter then they will be enclosed with brackets **( )**.

Carriage Return

The carriage return (0x0D) is a terminating character and triggers the EvoDrive to execute the command.

A carriage return must terminate all commands. Line feeds will be ignored.

4.2 Stacked Commands

There are two broad categories of commands, Stackable and Non-Stackable.

Stackable

Stackable commands are short, 1-2 character commands with single numeric parameters. They do not return any value individually.

They can be stacked up into one command string for fast execution, for example

/1 PA5000 PD2000 PV300 P1350 <cr>

Each command is executed in turn and immediately one after the other.

Movement commands do NOT wait for a movement to complete before executing subsequent commands. For example, the following command will

/1 P5000 P1000 <cr>

1. Start a move to 5000
2. Immediately start a new move to 1000

Because these commands are executed immediately, the motor will start to move to position 5000, then

immediately commanded to move to position 1000. The effect will simply be a movement to 1000.

See the Wait Commands (Section 13.6) for methods to pause the execution of commands.

Non-Stackable Commands

Non-stackable commands return a value or change a setting that would interrupt normal operation of subsequent commands. Non-stackable commands can only be sent one at a time, for example

/1 BAUD(1,9600) <cr>

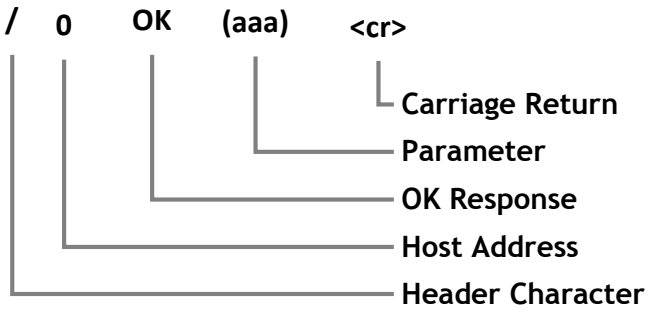
Replies with

/0 OK() <cr>

Non-stackable commands will return a value or a confirmation of whether or not the operation was successful.

4.3 Reply Format

The EvoDrive replies are similar in format to the commands. They are always addressed to the Host (Address 0) and terminated with a carriage return (0x0D).



Stackable commands do not return replies individually. On successful completion of a stack of commands, the EvoDrive will return an empty OK response:

/0 OK() <cr>+

Query commands will return a value using the OK response. Where multiple values are returned they will be separated by commas within the OK response. For example:

/0 OK(12) <cr>

/0 OK(123, 456) <cr>

If any command encounters an error, then an Error response is returned and any remaining stacked commands discarded.

The Error response will contain an error code and any relevant information. Examples are shown below:

Unknown Command

/0 ERROR(10, cmd) <cr>

Where **10** is the status code for Unknown Command and **cmd** is the offending command string.

Invalid Parameter

/0 ERROR(11, 1234) <cr>

Where **11** is the status code for Invalid Parameter and **1234** is the offending parameter value. Note that this value may be 0 or -1 for missing parameters.

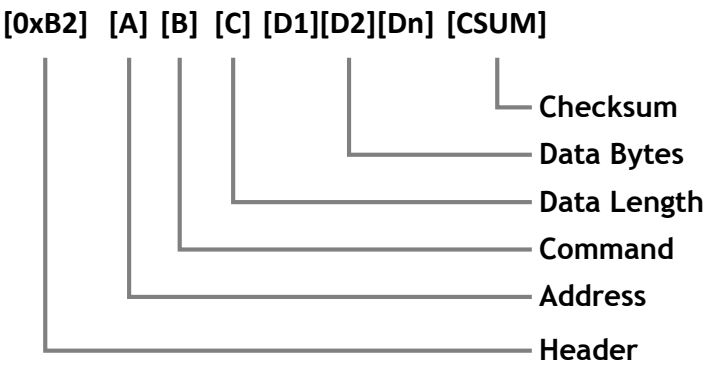
5. Binary Command Format (OEM Mode)

5.1 Command Structure

Both channels also allow a binary command format to be used in place of the ASCII commands. The binary format allows for consistently low command latency and error checking using an LRC checksum.

In this section, the square brackets […] are used to indicate a value is one byte (8 bits) long.

The basic command structure is shown below



Header (0xB2)

The header byte (always 0xB2) is used to start a new command.

If sent prior to the data bytes then all previous bytes are discarded.

Address

The address is used to identify the EvoDrive. This can be any alphanumeric character (**1…9**, and **A…Y**), allowing up to 34 devices on one bus.

- **0** is reserved for the host.
- **Z** is a broadcast address; the command will be executed by all devices.

Note: The value sent is the ASCII value of the address. For example, address 1 will be sent as [**0x31**]. This is to help with debugging.

Command

The commands are always single byte values. For example, [**0x73**] sets the drive current level (Equivalent to the ASCII **ID** command).

Most commands either set or query a variable. When setting a variable, bit 8 is always cleared. When reading a variable bit 8 is set, and all other bits remain the same.

For example,

To set the drive current level the command is [**0x73**]

To read the drive current level the command is [**0xF3**]

Data Length

This is always a single byte. The value (in binary) is the number of data bytes to follow.

For example, if there are 5 data bytes in the command, the value will be [**0x05**].

If there is no data then send a value of [**0x00**].

Data

What data is sent will depend on the command.

Numerical data is sent as a binary value in two's complement format. Values greater than one byte in length are sent with the MSB first.

32 bits is the maximum data size that can be sent. Negative values must send all 4 bytes. For example,

12 is sent as [**0x0C**]

12 500 is sent as [**0x30**][**0xD4**]

-12 500 is sent as [**0xFF**][**0xFF**][**0xCF**][**0x2C**]

Other data types, such as strings, are sent as the ASCII representation of the string. For example, **“V101.2.7”** is sent as

[0x56][0x31][0x30][0x31][0x2E][0x32][0x2E][0x37]

Checksum

The checksum is a longitudinal redundancy check (LRC) checksum that adds a level of error checking to the binary format.

The checksum is calculated by performing an exclusive OR operation on each byte of the command:

```
CSUM = 0x00;
For each BYTE in Command
{
   CSUM = CSUM ^ BYTE
}
```

The checksum is calculated for all bytes in the command, including the header.

For Example, the command to query the status is

[0xB2][0x31][0x80][0x00][0x03]

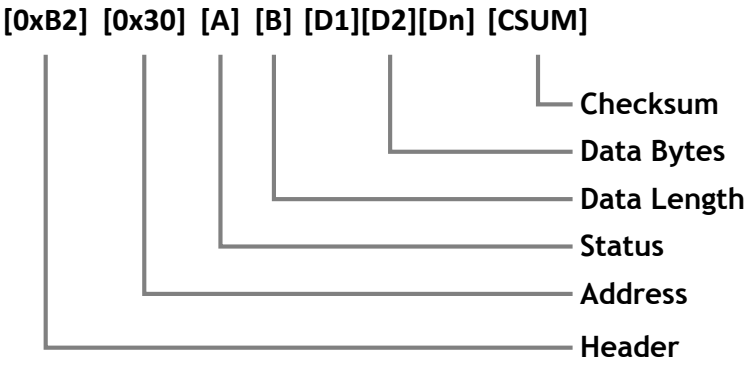
Where [**0x03**] is the Checksum. It was calculated as

CSUM = 0xB2 ^ 0x31 ^ 0x80 ^ 0x00

5.2 Reply Format

The EvoDrive replies are in a similar format to the commands. They are always addressed to the Host (Address 0) and terminated with an LRC checksum.

All commands generate a reply. Commands which are not query commands, and so do not expect any data returned, will return an empty reply (Data Length will be 0).



Header / Address

The header byte is always [0xB2].

The address byte is always [0x30].

Status

The status byte contains general status information. It is a composite byte made up of many status flags:

Bit 8: Busy Flag

The busy flag is only set to 1 when the EvoDrive is actively performing a move. Note that if the motor is moving under constant torque control, or from external force, the busy flag is 0.

Bit 7: Clockwise Motion Flag.

Set when the motor is turning clockwise. This includes unplanned motion from external forces.

Bit 6: Counter-Clockwise Motion Flag

Set when the motor is turning counter clockwise. This includes unplanned motion from external forces.

Bit 5: Enabled Flag

This bit is set while the motor power output is enabled. Note: This bit does not reflect the state of the external enable pin.

Bits 4 - 1: Status code

The lower nibble provides the current status code, as defined in section 11.2.

Data Length

If data is to be returned, the data length informs the host how many data bytes to expect. The value is the binary number of data bytes.

For example, if 4 bytes are to be returned, the data length would be [0x04].

Data Bytes

What data is sent will depend on the query.

Numerical data is sent as a binary value in two's complement format. Values greater than one byte in length are sent with the MSB first. For example,

12 is sent as [**0x0C**]

12 500 is sent as [**0x30**][**0xD4**]

-12 500 is sent as [**0xFF**][**0xFF**][**0xCF**][**0x2C**]

Other data types, such as strings, are sent as the ASCII representation of the string. For example, "**V101.2.7**" is sent as

[**0x56**][**0x31**][**0x30**][**0x31**][**0x2E**][**0x32**][**0x2E**][**0x37**]

Checksum

The checksum is attached to the end of the reply and is calculated in the same way as for commands from the Host.

Error Replies

If any command encounters an error, then the status code is returned in the status byte of an empty reply.

For example, the command to set the torque output to 105% is,

[0xB2][0x31][0x54][0x01][0x69][0xBF]

The reply is

[0xB2][0x30][0x1B][0x00][0x99]

In this reply, the status byte is [**0x1B**], which is read as:

Bit 8: **0** ➔ Not busy

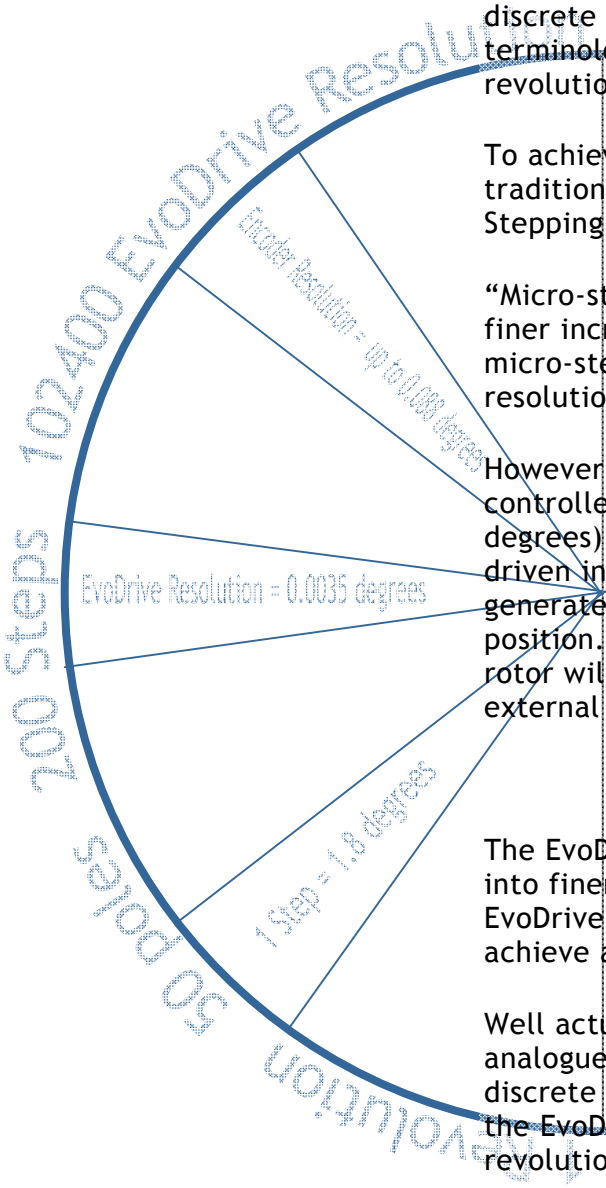
Bit 7: **0** ➔ Not moving clockwise

Bit 6: **0** ➔ Not moving counter-clockwise

Bit 5: **1** ➔ Drive output is enabled

Bits 4-1: **0x0B** ➔ Parameter out of range.

6. Resolution and Units



There are several units of rotation commonly used with stepper motors.

On a typical 1.8 degree stepper motor there are 200 discrete electrical “steps”. In other motor terminology, this equates to 50 poles (or electrical revolutions) per mechanical revolution.

To achieve greater accuracy these “steps” are traditionally broken up into finer increments. “Half-Stepping” will divide the traditional “step” into 2.

“Micro-stepping” can break the full “step” into much finer increments. It is not uncommon to see 256 times micro-stepping drivers; these have a maximum resolution of 51 200 “micro-steps” per revolution.

However, the practical resolution of any open-loop controller is always limited to one “step” (eg 1.8 degrees). This is because while the rotor can be driven in fine increments, the motor does not generate torque without slipping away from this position. So when driving the motor to a position the rotor will often over or undershoot depending on the external loads, deceleration, and detent torques.

The EvoDrive does not attempt to break up the “step” into finer increments and then play them out; the EvoDrive calculates the optimum use of currents to achieve any arbitrary rotor position.

Well actually… While the EvoDrive tries to be as analogue as possible, it is digital and therefore a discrete system. So the maximum theoretical “steps” the EvoDrive can resolve is 102 400 “steps” per revolution (equivalent to 512 microstepping).

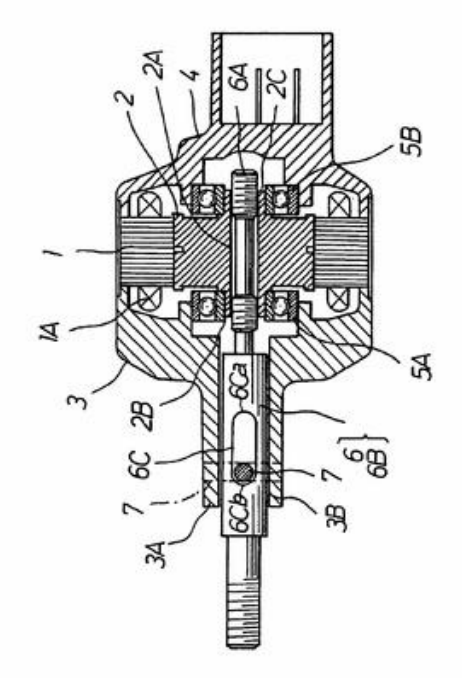
However in a true closed-loop controller the practical resolution is up to half the encoder resolution. So with a 20 000 count encoder the maximum reliable system resolution is 10 000 ‘steps’ per revolution.

So what unit to use and does it really reflect what the motor is doing?

The EvoDrive allows the system designer to specify the number of "steps" per revolution using the **RES** command.

This custom resolution defines a unit we call the **UserStep** - the reference for all commands and values returned by the EvoDrive.

- This allows a meaningful and practical unit to be applied to the system to simplify design and debugging.
- It also allows control by legacy systems which already use an establish system of units.



Example - Linear Motion

An application uses the EvoDrive to drive a lead screw with a pitch of 2 mm. That is, one revolution will result in a linear movement of 2mm.

By setting the resolution to **2000** UserSteps per revolution, all EvoDrive set-point and query commands will be in micrometre units at the output of the lead screw.

Note that no accuracy is lost. If using an encoder, the accuracy will still be as good as half the encoder resolution.

UserSteps simply allows the command parameters to have some meaningful units and the design to be simplified as a result.

7. Control System

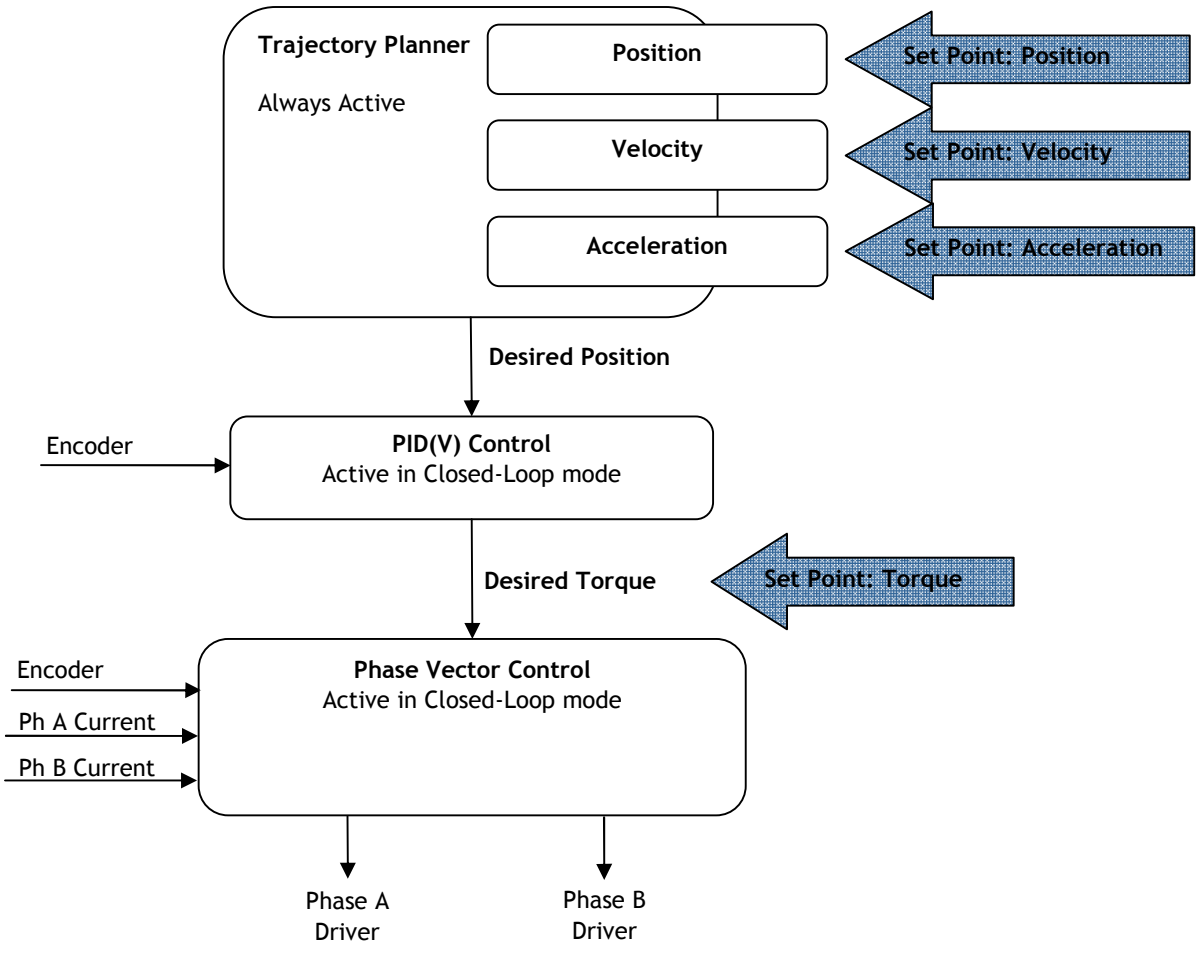
The EvoDrive runs multiple control systems that are enabled and disabled automatically depending on the control requirements at the time.

Some are always active and provide the basic open-loop (no encoder) control.

The others activate when running as a closed-loop controller (encoder attached) and provide additional benefits and features.

All stages of the control system are controlled by the set-points. Set-points, and therefore the way the motor is controlled, can be changed on-the-fly.

A basic diagram of the control flow, and the set-points, is shown below. When a set point is written, it will reset all set points above it, and override all set points below it.



Trajectory Planner

The output of the trajectory planner is a position/velocity curve that the motor should follow to achieve a set point.

When the position set-point is written, the trajectory will be defined by the acceleration, final velocity and deceleration profile parameters (see **PA, PV,** and **PD** commands, Section 13.4).

When the velocity set-point is written, the motor will accelerate according to the profile but will run continuously at the set-point velocity.

When the acceleration set-point is written, the motor will accelerate continuously at the specified value.

PID Controller

The PID controller outputs a torque set-point; this is the torque required to follow the desired trajectory.

The PID controller will control the torque output according to the PID Gains. The PID Gains are tuneable (see **PID** command, Section 13.2) to achieve a certain system response.

The PID controller is only active in closed-loop mode (when an encoder is attached). In open-loop control the output torque set-point is equivalent to the drive current specified by the **ID** command.

In open-loop mode, the controller output will essentially follow the desired trajectory with the specified drive current regardless of the motors actual position. If the current is insufficient or the profile too fast then the motor may lose sync.

Torque Set Point

The torque set-point is always active and is normally set by the PID controller.

Setting the torque set-point manually (see **T** command, Section 13.5) will have little effect in open-loop mode.

Setting the torque set point in closed-loop mode will cause the EvoDrive to produce a constant torque output, regardless of profile or motion settings.

Position Holding

In closed-loop mode, the PID controller will actively maintain the position set point, within the collision error-band bounds, using whatever torque is necessary.

When in open-loop mode, the maximum holding current will be applied regardless of the motor position. The EvoDrive will rely on this current to hold the set position and cannot dynamically adjust for any slip or loss of synchronisation.

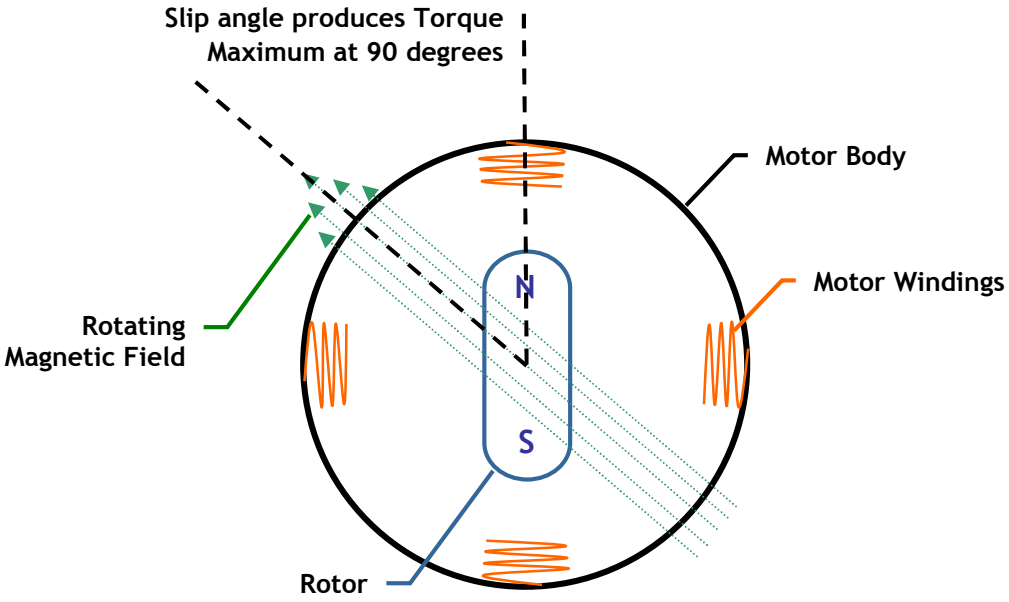
Phase Vector Control

The Phase Vector Controller is only active in closed-loop mode, and consists of a further five control sub-systems. Together with the drive electronics it is referred to as the Vector Drive.

The purpose of the Vector Drive is to produce a net torque output from the stepper motor. To explain how it works, we will use a very simplistic example of a stepper motor.

At its simplest level a stepper motor is basically a magnet on the rotor (motor shaft) that follows a magnetic field. As the magnetic field rotates, the rotor (the motor shaft) rotates.

Traditional stepper motor controllers generate a magnetic field that rotates according to the trajectory. As the rotor slips away from this field, a torque is generated and the rotor follows.



Normally the strength of this magnetic field, which is a function of the current, should be strong enough to overcome any expected acceleration or obstacles. If it is not, then the rotor can't keep up with the field and the motor loses sync.

However any current not being used to accelerate or overcome obstacles is not doing any work and so is simply converted to heat. This tends to be very inefficient.

When the motor loses sync the magnetic field keeps rotating anyway. In stepper motors this results in a complete loss of torque and the rotor will vibrate within its electrical pole, or within 4 'steps' of its current position.

The Vector Drive overcomes these traditional problems by continuously monitoring the position of the rotor and generating the magnetic field so it always sits 90 degrees ahead of the rotor. This means

- The motor will never loose sync,
- The vast majority of current is converted into useful torque (very efficient).
- The field strength is dynamic. This means the torque output can be controlled accurately resulting in a high dynamic range and a much quieter motor.

8. Commissioning and Tuning

This section is intended to provide some advice, rules-of-thumb, and considerations to be made when commissioning an EvoDrive stepper motor controller.

8.1 Closed Loop vs. Open Loop (ENC command)

A key question when commissioning a new design is whether to use closed loop or open loop control.

Below are some advantages, disadvantages, and considerations to consider when choosing a control mode.

Open Loop

- Open loop is cheaper since there is no need for an encoder.
- The EvoDrive will implement a quiet and smooth open loop control.
- Little to no tuning is required to achieve tight, accurate control.
- It is very inefficient because the maximum drive current is used all the time.
- The motor and controller may get very hot.
- If the drive current cannot overcome obstacles, the motor will lose sync with a sudden loss in torque.
- The motor must be driven much more conservatively to avoid losing sync.
- Collisions are not detected.

Closed Loop

An encoder is required for closed loop control. Two encoder resolutions are available from EVA Robotics as well as the option of using a third party encoder.

- More accurate position control.
- The motor will run smoother and quieter.
- The motor has greater dynamic range (it can achieve greater speeds for less current)
- The motor and controller runs much cooler and more efficiently.
- Torque output can be monitored and/or controlled.

- The motor will not lose sync – it will attempt to push through obstacles.
- Collision detection is possible.
- Less safety margin is required when selecting a motor.
- Tuning may be required to achieve a desired system performance. Though the default tuning is usually suitable.

Encoder Resolution

Any quadrature encoder can be used as long as it has a resolution greater than 8192 counts per revolution.

The encoder provides two resolutions that define the control performance:

Position resolution: This is equal to half the encoder resolution. So with a 8192 count encoder the output can be controlled to ± 0.044 degrees.

Control resolution: This is the torque resolution available to the control loop to control the motor. An 8192 encoder provides a control resolution of ± 2.5%.

The control resolution doesn't sound very accurate but this is actually more than enough for most applications. Because the control is closed loop, as long as you have enough torque overhead the motor will follow the trajectory profile at a much finer resolution than in open loop.

As the encoder resolution increases so does the control resolution. For example a 20 000 count encoder has a torque resolution of ± 1% and will satisfy the most demanding torque control applications.

8.2 Drive Current (ID command)

- In open-loop mode the drive current is a constant current given to the motor when moving.
- In closed-loop mode the drive current is the maximum current that can be applied to the motor.
- The drive current must not exceed the rated current of the motor.
- It must be strong enough to overcome any obstacles or acceleration requirements placed on the motor.
- The higher the drive current, the lower the maximum drive speed the motor will be able to reach.
- Typically the drive current will be approximately 50% - 75% of the rated motor current. This will vary with the motor brand and method of rating.

8.3 Holding Current (IH command)

The holding current is a constant current applied to the motor when maintaining a position.

- If the holding current is too strong, the EvoDrive, and the motor, may overheat.
- In open-loop mode, this current must be strong enough to maintain the final position.
- In closed loop mode it is also applied to ensure accuracy, but extra power may be applied to maintain the position – See Collision Handling, section 8.4.

8.4 Trajectory Profile (PA, PD, and PV commands)

The trajectory profile is set by the Acceleration (PA), Deceleration (PD) and Final Velocity (PV) commands.

- The initial velocity and stop velocity values are always 0.
- The acceleration and deceleration values should be selected according to the load and the capability of the motor.
- If acceleration or deceleration is too steep, the motor may not track the desired trajectory. In open-loop mode this will cause the motor to loose sync. In closed loop mode the motor will

attempt to catch up, but may trigger a collision alarm.
- If the final velocity is too high then the motor may not be able to track the trajectory. This will result in a loss of sync in open-loop mode or a collision alarm in closed loop mode.

8.5 Collision Handling (COL command)

Collision detection is only active in closed loop mode. The EvoDrive uses two parameters to detect a collision state: an Error Band and an Error Time.

When the motor position differs from the desired trajectory by more than the Error Band and for a time longer than the Error Time, a collision alarm is raised and the motor will stop.

- The Error Band is in UserSteps and is typical of the collision detection seen in most motor controllers.
- The Error Time allows the motor position error to be filtered. It is particularly useful when driving a sticky or over-constrained drive-train, or when trying to accelerate the motor very quickly. It allows the motor some time to overcome the obstacle and catch up with the trajectory.
- It is usually best to have a small Error Band, but a large Error Time, so the EvoDrive is given a chance to regain control of the motor position.

Before triggering a collision alarm (before the Error Band and Error Time are exceeded) the motor will attempt to correct its position. This is also true when holding a position.

If an external influence forces the motor from its holding position, the EvoDrive will apply power against the force to try and regain its position. This power may exceed the holding current in certain circumstances. If the Error Band and Error Time are exceeded while trying to hold the position, the motor will give up trying to maintain the position to avoid over-heating the motor.

The behaviour of the EvoDrive after a collision can also be defined. There are two options:

- Mode 1 (default): The motor is allowed to brake itself regeneratively. No active setpoint is maintained and the motor is allowed to move.
- Mode 0: The EvoDrive changes to a velocity setpoint of 0. That means it will attempt a 'graceful' ramp down to a standstill, and hold that position. This is useful when driving a high inertia load, or if the collision alarm occurs at high speed.
  However if the error is due to a faulty encoder then the motor may spin uncontrollably for a short time.
  Furthermore, most collision alarms occur when the motor is being stalled, and therefore a ramp down is not necessary.

8.6 Soft Limits (Limits command)

The EvoDrive can support soft motion limits (firmware Vx.7.0 onwards), which are configured using the **Limits()** command.

There are two configurable limits, a low and high position. The allowable motion envelope lies between these two limits.

When the limits are enabled, the EvoDrive implements the following behaviour:

- Position commands must send the motor to a point within the motion envelope. Note that if the motor is currently outside the envelope, the motor will be allowed to move to a point within the envelope.
- Velocity commands will move at the speed specified and decelerate to the applicable soft limit. In reality, the velocity command becomes a position command - going to the soft limit at the desired speed.
- Digital control (e.g. Step/Dir signals) will allow motion into, and within, the motion envelope. However the motor will not move out of the envelope.

- Torque control, acceleration control, free-wheeling (from collision or disabled states) are not limited by the soft limits.
- Regardless of the mode, even if not restricted by the soft limits, the drive status and output status will report a soft-limit error whenever the motion is outside of the limits.

8.7 PID Gains (PID command)

A PID Controller is used in closed-loop mode to control the torque of the motor so it tracks its desired trajectory.

The vast majority of applications will not require the PID gains to be re-tuned.

The performance characteristics of the motor are defined by the settings of the PID Gains. The PID Gains must be tuned for the mechanical system being driven. The inertia, friction, damping, and resonances all affect the dynamic performance of the system.

PID tuning is not widely understood and should only be performed by an engineer with knowledge of control theory. There are several ways to approach tuning, but in general…

- The P gain (Proportional) sets the output power based on the position error. So the bigger the error, the bigger the P Gain output
- The I Gain (Integral) sets the output power based on the integration of the error. So the longer an error exists, the higher the I Gain output will be.
- The D Gain (Derivative) sets the output power based on the derivative of the error. So the faster the Error is changing, the higher the D Gain output.
- A gain of 0 disables that part of the PID controller.
- The most common PID controller is a PI controller (D Gain = 0).
- I've heard of P, PI, PID, or even PD controllers, but *never* an ID controller.
- Tuning the step response of the system is a good way to start, but only if the step is small.
- Tune P only (I = 0 and D = 0) so it gives good position holding force, and reasonable performance when following a profile.
- Add I Gain to improve response time and steady state error.
- Add D Gain to damp out system resonance and oscillation. This isn't usually necessary as most systems naturally have high friction or damping.

8.8 Determinism (POLL Command)

The **POLL** command sets the polling rate used by triggered commands and wait commands.

The polling rate defines how deterministic (or how responsive) the EvoDrive is to events.

Digital Input Debounce

Digital inputs are debounced at twice the polling frequency. That is, an input must be steady for two polling cycles before triggering a Program (See section 13.9) or a Wait on Input command (Section 13.6).

For example, if the inputs are sampled every 10ms (**POLL(10)**) then a Wait on Input command (**WPI**) won't be executed until the input is steady for approximately 20ms.

10ms is the default setting. The debounce setting is important for filtering input noise such as switch bounce and electrical interference such as mains frequencies.

Wait Commands

The Wait commands cause the EvoDrive to wait for an external event. These events may be waiting for the motor to reach a certain position, to stop, waiting for an input, or waiting for a period of time to elapse.

These states are monitored at the Polling rate.

Command Response

All commands, external as well as internal (triggered programs and wait commands), are monitored every 5ms.

For external commands (i.e. Commands sent to the EvoDrive) the delay to execute the command is

[transmission time] + [Command Response]

For example, sending the command **/1P1000<cr>** at 9600bps means the command will be executed in 9 – 14ms:

- Transmission Time: approx 8ms
- Command Response: 1 – 5ms

For internal commands (i.e. Triggered programs and wait commands) the delay to execute the command is

[polling or debounce time] + [1 - 5ms]

For example, when waiting on a position (**WP**) with a polling time of 10ms, the delay to execute the remainder of the command string after hitting the trigger position will be 1 – 15ms:

- Poll Time: 0ms – 10ms
- Command Response: 1 – 5ms

When using a digital input as a trigger the input must be debounced, which means one more polling cycle must be added to the delay time.

For example, when waiting on a digital input with a polling time of 10ms, the delay to execute the triggered program (**PROG**) or remainder of the command string (**WPI** or **WNI**) will be 11 – 25ms :

- Poll Time: 10 – 20ms
- Command Response: 1-5ms

Command Precedence

It is important to note that external commands take precedence over internal commands from the time the EvoDrive address is sent (i.e. **/1**…) to when the command has finished executing.

So if the programmer is typing a command via a terminal program, no internal commands or triggers will be honoured until after the external command has been completed.

8.9 Resonant Damping (DAMP Command)

Low speed resonant damping only applies under closed loop control.

The construction of stepper motors is optimised for course full step drivers. Because of this, most types of stepper motor have a natural tendency to pull toward the next full step (e.g. 1.8 degree step). This is called the detent torque, and it can present problems when trying to maintain a position between steps, or when turning slowly.

Detent torque can be seen as regular and repeated potholes in the road. When moving very slowly, the tires want to find the bottom of each pothole, and take some force to roll back out of them. As you speed up, the force required is steadier, but they cause vibration and uncomfortable resonances. At high speeds, the potholes become less noticeable.

Low speed resonant damping is enabled by default. The DAMP command enables or disables a parallel control system that counteracts the detent torque and provides extremely accurate and smooth low speed motion.

That sounds great – so why would you want to turn it off? Well, you probably won't want to, but…

Resonant damping is only active at speeds less than 2.5 revolutions per second. So for applications which accelerate quickly up to faster speeds it will have little effect.

It also uses more current, which can result in a hotter motor and may increase the tone pitch the motor generates.

So if the application involves low speed motion and is not sensitive to the detent torque, then the motor will run quieter and cooler without resonant damping enabled.

8.10 Motor Calibration (CAL Command)

The EvoDrive is capable of driving any stepper motor with up to 3 Amps. However, stepper motors vary widely in construction and winding design.

The EvoDrive is pre-tuned to suit the high-torque stepper motors available from EVA Robotics. This tuning will also suit the majority of commercially available high-torque motors.

Other types of stepper motors may produce a loud, high-pitch squeal, or low speed motion may be erratic. This can be corrected by auto-tuning the EvoDrive current control using the **Cal()** command:

- The Evodrive must be disabled (**EN0**)and the motor stationary before performing a calibration.
- Send the calibration command **Cal()**
- The motor will squeak for a short time (less than a second) and the EvoDrive will reply with an **OK()**.
- At this point the new calibration is active, but not committed to non-volatile memory. This allows you to enable the motor and try out the new calibration. The calibration can be repeated if necessary.
- To commit the new calibration to non-volatile memory send the **Save()** command.

This calibration is NOT related to the PID gains. The motor calibration tunes the internal current control of the EvoDrive, the PID gains tune the motion control of your system.

For more advanced calibration, which includes a calibration of the motor structure and encoder errors, use the Auto Calibrate feature in EvoLink. This calibration can improve smoothness and speed regulation. See the Integration Manual for more information.

9. I/O Configuration

9.1 Digital Input (CNC) Control

The EvoDrive has four dedicated digital input lines. On the EvoDrive ST-PCB, these are configured for digital control by default.

Two modes of digital input control are supported on all models:

- **Step/Dir** control: The Step input will move the motor one step, in the direction defined by the Dir input.
- **CW/CCW** control: The CW input will move the motor one step clockwise. The CCW input will move the motor one step counter-clockwise.

On each digital input the positive edge, the negative edge, or both can be used to step the motor.

To enable electronic gearing and replacement of legacy steppers, the step size is defined by the RES command. For example, if the User Resolution is 400, then one edge on the control input would move the motor 1/400th of a revolution.

Configuration

The digital inputs are configured using the Triggered Program Commands (See section 13.9). The following commands configure the inputs:

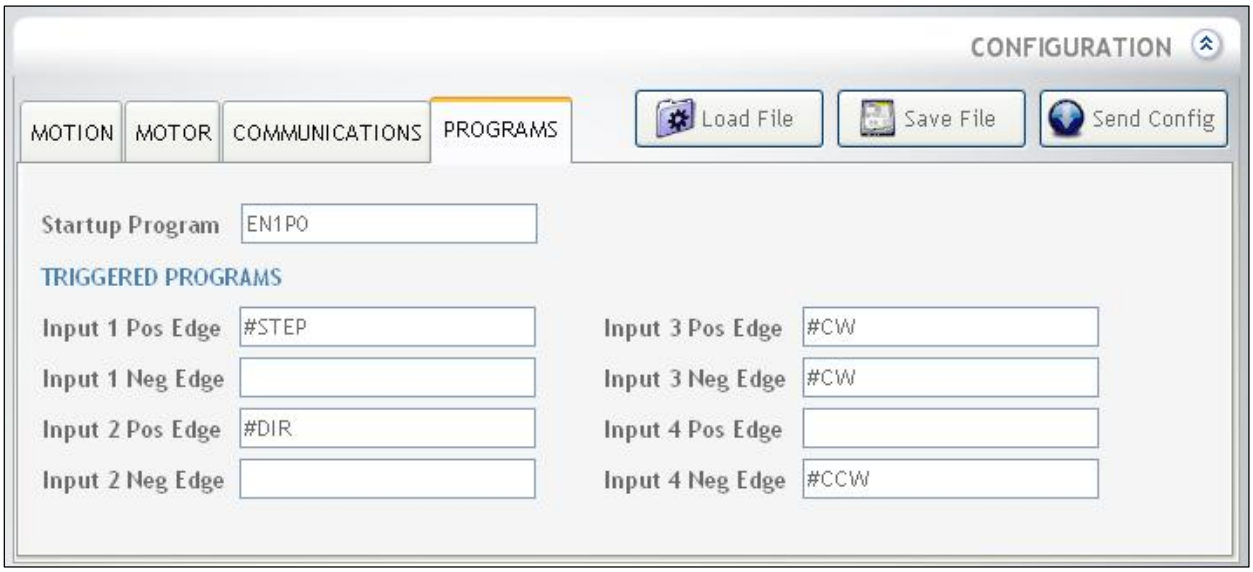
#STEP : The input edge is used to step the motor in Step/Dir mode. This can be the positive, negative, or both edges.

#DIR : The input is used as the direction reference in Step/Dir mode. High is clockwise, Low is counter-clockwise.

#CW : The input edge is used to step the motor clockwise.

#CCW : The input edge is used to step the motor counter-clockwise.

For simplicity, the following example shows the program configuration using EvoLink.



Note that both control modes can be active at the same time.

In this example, Inputs 1 and 2 will operate the motor in a Step/Dir mode, where a positive edge on Input 1 will step the motor in the direction defined by the level of Input 2.

Input 3 will step the motor in a clockwise direction on both positive and negative edges.

Input 4 will step the motor in a counter-clockwise direction on negative edges.

Operating Considerations

The control inputs will only work when the EvoDrive is actively holding a position. Therefore, in the example above, the startup program (S0) has been modified to **EN1P0**, so immediately after power on the EvoDrive is in position mode.

The control inputs will not work after a collision error or power loss. If necessary, a third digital input can act as a reset input, to place the EvoDrive back into position mode with the command **EN1R0**.

When configured as a digital control input, the input pin is not de-bounced to allow for fast step rates. If

controlling the motor via a switch, we recommend using the Triggered Programs feature rather than the digital input control. For example, a program of **R1** performs the same function as a clockwise step input.

9.2 Status Outputs

The EvoDrive ST-17 and ST-23 have four dedicated digital output lines, which by default are configured as normal digital outputs.

These outputs can be configured to reflect the operating state of the EvoDrive via the **DSTAT** command. One of eight status signals can be selected:

Status	Description
1	**Vector Drive Enabled** The output is held closed while the vector drive is enabled. This does not indicate motion or power to the motor, only that the drive outputs are enabled.
2	**Motion Control Busy** The output is held closed while the motor is performing a move. Note that this does not include unplanned motion, such as motion due to external forces or while under torque control.
3	**Motor is Rotating** The output is held closed whenever the motor shaft is rotating, whatever the cause may be.
4	**Error Alarm** The output is held closed while any error exists. This includes all errors other than command errors, such as incorrect command.
5	**Collision Alarm** The output is held closed while a collision error exists. When the collision error is cleared (by retrying a move or disabling the motor) the output is opened.
6	**Motor is Rotating Clockwise** The output is held closed whenever the motor shaft is rotating clockwise.
7	**Motor is Rotating Counter-Clockwise** The output is held closed whenever the motor shaft is rotating anti-clockwise.
8	**Soft Limit Alarm** The output is held closed whenever the motor is outside the soft limits defined by the **Limits()** command.

The digital outputs are active low, therefore a closed output means the output is a low or sinking output.

10. Initialisation and Homing

On power-up, the EvoDrive initialises the vector drive and internal systems. It will then run its start-up program (see Triggered Programs, Section 13.9) which by default will enable the vector drive and hold its initial position.

Enabling the vector drive will

- Enable the encoder inputs if applicable.
- Initialise the encoder if applicable, see below.
- The control set-point is cleared.
- The Vector Drive outputs are enabled.

Encoder Initialisation

If an encoder is attached and configured, the vector drive must initialise the encoder before it can be used.

This process waits for the motor to be still (up to a settling time) and will produce one or two sharp knocking sounds or kicks. The motor may move up to a step during this process. Ideally, the motor shaft should be allowed to move during this initialisation. It can drive a load as long as it allows the rotor to turn.

If the encoder is not present it will result in an initialisation error (see section 11.2).

If an external load is forcing the motor to turn during initialisation or the motor takes too long to come to a stop, then it may also result in an initialisation error.

If this small motion might present a danger to personnel then the start-up program must be modified to allow for a controlled, safe initialisation.

Homing

After initialisation, the position is set to 0. Normally the application will want to home the motor to a known position. This known position could be an external sensor or it could be a hard stop.

There is no specific homing command for the EvoDrive. Homing is achieved through use of the Wait Commands (see Section 13.6). Some examples are given below.

To home to an external sensor, the Wait for Input commands can be used. For example

/1 V100 WNI1 Z P0<CR>

This command will move the motor slowly in a clockwise direction until a negative edge is seen on input 1. Within 20ms of seeing the edge the position will be set to zero. The **P0** command will slow the motor and return it to the place where the sensor edge was seen.

When running in closed-loop mode the EvoDrive can be homed against a hard stop. Because the EvoDrive will not lose sync when it hits a hard stop there are no problems with bouncing or jitter while homing.

/1 V-100 WS Z P10<CR>

This command will move the motor slowly in an anti-clockwise direction until it hits and drives against an obstacle. The obstacle causes the motor to stop, satisfying the **WS** (Wait for Stop) command and the position is set to 0. The motor is then moved off the hard stop slightly by moving to position 10.

Note that the EvoDrive will drive against the hard stop with all available torque given the drive current configured with the **ID** command. The hard stop must be able to hold that load, or the command could be modified to use a lower drive current. For example

/1 ID500 V-100 WS Z P10 ID1000 <CR>

11. Error States

11.1 Status LED

The Status LED is a combination red and blue LED which provides instant status feedback.

The Blue in the status LED is on whenever the Vector Drive is enabled and ready to control the motor.

The Red in the status LED will pulse a heartbeat (two quick successive pulses) every 5 seconds while there are no errors.

If there is an error condition then the Red LED will flash the status number every five seconds.

For example if there is an Error 3 (over-current alarm), the Blue LED will be off and the Red LED will flash 3 times (ON for 1 second, then OFF for 1 second). It will repeat after a 5-second delay.

11.2 Status Codes

The status code is pulsed by the status LED.

It can also be read programmatically with the Query command **?** (see Section 13.7) and in the binary status byte (see Section 5.2).

Status Code	Description
0	**No Error**
1	**Initialisation Error** This may be reported when enabling the Vector Drive if the encoder could not be initialised. Ensure the motor shaft is free to move, all cables are set properly, the encoder is powered and firmly attached to the motor shaft.
2	**Collision** A collision was detected while moving, or the motor was forced to move while holding a position. For a full explanation on collision detection see Section 8.5. To disable collision detection, or to change the sensitivity, see the **COL** command (Section 13.2).
3	**Over Current.** The motor currents exceeded the maximum. This will automatically disable the Vector Drive. To change the over-current threshold use the **IX** command (Section 13.3)
4	**Over Temperature.** This error will be reported when the core temperature reaches 70 degrees Celsius. When the temperature reaches 90 degrees Celsius the Vector Drive will be disabled.
5	**Motor Loaded During Initialisation** This error indicates that a force is being applied to the motor shaft while initialising the encoder. Ensure the shaft is free to move and re-enable the Vector Drive using the **EN** command (Section 13.10)

Status Code	Description
6	**Power Loss Detected** The Vector Drive power is off or dropped significantly enough to cause a loss of position. The EvoDrive must be re-initialised using the **EN** command (Section 13.10) If this error repeats it indicates that the power supply is not capable of supplying the necessary current on demand.
7	**Flash Error** This indicates a problem reading from or writing to the non-volatile memory. If this problem persists then the EvoDrive will need replacement.
8	**Soft Limits Error** This indicates that the motor has reached a soft limit. Or the EvoDrive has been commanded to move beyond a soft limit.
10	**Unknown Command** This error is momentary so it won't be displayed on the status LED.
11	**Invalid Parameter in Command** This error is momentary so it won't be displayed on the status LED.
12	**Cannot set the Control Set-Point** This error is momentary so it won't be displayed on the status LED. This is normally seen when trying to write to a control set-point while the Vector Drive is disabled.
14	**Checksum error** This error is momentary so it won't be displayed on the status LED. It is returned if a binary command is received sent with a mismatch on the checksum.

11.3 Boot Loader Mode

If the red status LED is blinking quickly (faster than once per second), with no pause to indicate a status code, then the firmware has become corrupted.

The EvoDrive has entered a boot loader mode to allow firmware to be loaded. The latest firmware as well as upgrade tools can be downloaded from www.evarobotics.com.

The EvoDrive will not respond to normal commands or queries while in the boot loader mode.

12. Configuration Variables

The EvoDrive configuration is stored in non-volatile memory as a series of configuration variables. Most of these variables are modified using configuration commands (see Section 13.2). Using configuration commands is the preferred method because the EvoDrive will check the commands for valid ranges, and convert units to the proper value.

However, it may become necessary to configure the Configuration Variables directly, for example, when loading a default configuration in a manufacturing environment.

Furthermore, some configuration variables provide advanced tuning and configuration not available in the standard configuration commands.

The value of all configuration variables can be read by using the command:

Read(<name>)<cr>

Likewise, all configuration variables can be set by sending the command:

Write(<name>, <value>)<cr>

where <name> is the name listed in the following table, and <value> is the new value for the variable.

Only change the configuration variables using these commands if you are sure of the values. These commands do not check the value for correctness, and incorrect values may damage the EvoDrive or your equipment. If you are unsure of the values, use the configuration commands in Section 13.2 or use EvoLink to change the values.

Advanced Configuration Variables

These variables are not exposed in the normal command set because they are not expected to change in the vast majority of circumstances. However, in particular applications, EVA Robotics may advise you to change one of these variables to improve performance.

These are shown in **black** in the following table

Other Configuration Variables

These variables are normally manipulated by the configuration commands, or they relate to settings that should not be tuned manually.

These are shown in grey in the following table.

Configuration Name	Description
MODEL	This defines the EvoDrive model: 17 = ST-17 23 = ST-23 101 = ST-PCB
PCBMODE	This applies to the ST-PCB only. It sets the communication mode used by the ST-PCB: 1 = UART Communications 2 = SPI Communications 3 = UART with the Input 4 pin assigned to RS485 control.
ADDRESS	This is the EvoDrive address in decimal format. For example, if the address is '1' (HEX 0x31) then this variable is set to 49.
BAUD232	This is the BAUD rate, in bps, of the Channel 1 interface (if applicable).
BAUD485	This is the BAUD rate, in bps, of the Channel 2 interface (if applicable).
PHAZERO PHAGAIN PHAPIDP PHAMAXI PHAMINI	These are calibration variables that apply to Phase A of the stepper motor. They are specific to the motor used and should not be changed manually or transferred between EvoDrive units. Always use EvoLink, or the CAL command, to calibrate the motor.
PHBZERO PHBGAIN PHBPIDP PHBMAXI PHBMINI	These are calibration variables that apply to Phase B of the stepper motor. They are specific to the motor used and should not be changed manually or transferred between EvoDrive units. Always use EvoLink, or the CAL command, to calibrate the motor.
DQLEAD	This is a factory test variable and will be shipped as 30. Do not change this without instructions from EVA Robotics.
USEENC	This is a 1 if an encoder is configured, 0 otherwise.
ENCRES	This is the encoder resolution. If there is no encoder attached, this value will be 115200.

Configuration Name	Description
ENCOFF ENCFLIP ENCCALA ENCCALK ENCCALP	These are calibration variables that apply to the specific motor and encoder combination being driven. They should not be changed manually or transferred between EvoDrive units. Always use EvoLink to calibrate the motor and encoder.
ENCSETL	This is the settling time, in milliseconds, to wait for the motor to reach a standstill before enabling the drive on reset. If there is load on the motor on power up or reset leading to unpredictable behaviour, collisions, or initialisation errors, then increasing this value may help.
USERRES	This is the value of the User Resolution, set by the RES command. All commands will use this value as the number of steps per revolution.
MOTORRES	This is the number of actual mechanical steps per revolution. 200 = 1.8 deg motor 400 = 0.9 deg motor
BLDC	This is defines the type of motor being driven and will always be 0.
ERRBAND	This is the error band in system units (102,400 steps per rev). This is set by the COL command.
ERRTIME	This is the error time in milliseconds. This is set by the COL command.
ERRMODE	This defines the behaviour of the EvoDrive when a collision has been detected. 0 = Set Velocity to 0. This implements a graceful deceleration of the load, useful when driving high inertia loads. However, if there is an encoder fault it may cause the motor to spin unpredictably. 1 = Brake the motor. This turns off the motor drive outputs and allows the motor to regeneratively brake itself. This is the safest behaviour in most applications.
IDRIVE	This is the allowable drive current (in milli-amps), as set by the ID command.
IHOLD	This is the holding current (in milli-amps), as set by the IH command.
IMAX	This is the maximum drive current threshold (in milli-amps), as set by the IX command.
HOLDTIME	In open loop mode, when the motor comes to a stop, the Evodrive continues to apply full drive current for a short time to reduce overshoot and improve the final accuracy. This is the value of that holding time in milliseconds.
HEATMAX	This is the temperature limit, in degrees Celcius, where the EvoDrive will trigger an Over Temperature alarm.
PROFILEA	This is the acceleration value for the motion profile. It is in system units (102,400 steps per revolution).
PROFILEV	This is the final velocity value for the motion profile. It is in system units (102,400 steps per revolution).
PROFILED	This is the deceleration value for the motion profile. It is in system units (102,400 steps per revolution).
PGAIN	This is the PID controllers P Gain as set in the PID command.
IGAIN	This is the PID controllers I Gain as set in the PID command.
DGAIN	This is the PID controllers D Gain as set in the PID command.
VGAIN	This is an optional velocity feed-forward gain for the PID controller. It is not usually used, but may help damp oscillation in very high speed applications.
VSLIP	Setting this variable to 1 allows the motor position to slip while trying to maintain a constant velocity set point. The slippage is only allowed when the position error is greater than the collision error band. This is useful in situations where allowing the motor to ‘catch-up’ to its position would cause excessive oscillation.
IOPOLL	This is the IO Polling time in milliseconds, as set be the POLL command.

Configuration Name	Description
DAMP	This variable enables (1) or disables (0) the low speed damping. It is the same as using the DAMP command.
DAMPIIR	This variable controls how quickly the low speed damping mode is applied as the motor speed changes. It affects how smoothly the low speed damping control is applied, and is useful for avoiding sudden torque changes when quickly accelerating or changing direction. 0 = Filtering disabled 1-20 = Level of filtering. 20 (default) is maximum filtering.
DAMPID	This is the maximum current level (in milli-amps) to use for low speed damping. 0 means the current level is determined automatically.
CPOL	This is the SPI Clock Polarity value as set be the CPOL command.
CPHA	This is the SPI Clock Phase value as set be the CPHA command.
VERBOSE	If set to 1, the EvoDrive replies have extra information inserted after the /0: **/0 [<Add>, <Cmd>] <Reply>** Where <Add> is the EvoDrives address. <Cmd> is the most recently executed command <Reply> is the standard reply.
DOSTAT1	This is the status type to be output on digital output 1 - as set by the DSTAT command.
DOSTAT2	This is the status type to be output on digital output 2 - as set by the DSTAT command.
DOSTAT3	This is the status type to be output on digital output 3 - as set by the DSTAT command.
DOSTAT4	This is the status type to be output on digital output 4 - as set by the DSTAT command.
LIMIT1	This is the lower soft limit as set by the Limit() command. This is in system units (102400 steps/rev)
LIMIT2	This is the upper soft limit as set by the Limit() command. This is in system units (102400 steps/rev)

13. Commands

13.1 Syntax

All commands given in this manual are written in the following syntax

COMMAND	Commands, and some parameters, are written in a bold **COMMAND** font. This shows exactly how the command should be sent.
<prmtr>	The < > placeholder indicates a parameter that must be sent with the command, where **pmtr** is the name of the parameter. The parameter will be described for each command.
[prmtr]	The [ ] placeholder indicates an optional parameter. These parameters do not need to be sent with the command.
|	Separates mutually exclusive options for a single parameter. For example **V<1000|2000>** indicates **V1000** or **V2000** are acceptable commands, but **V2500** is not an option and so will return an error.
( )	Parenthesis (brackets) are used literally as typed. Non stackable commands use these to enclose multiple parameters.
<cr>	Indicates a Carriage Return (ASCII 0x0D).
/1	Where header characters are shown in examples, the address will be shown as 1.
0x00	Where binary values are provided they are given in hexadecimal format.

No part of the command string, including the address, is case sensitive, therefore where commands are shown as capitalised it is for clarity only.

13.2 Configuration Commands

These commands are used to configure the EvoDrive. All configuration commands are non-stackable.

Settings are effective immediately, however they are not saved to non-volatile memory until the **Save()** command is sent.

On start-up the EvoDrive loads the saved configuration from non-volatile memory.

Set Address

Format	**Address**(**<address>**)
Description	This command sets the unit address. All commands sent after this must use the new address. The default address at start-up is '1'
Parameters	**<address>** New address for the unit. Must be a single character in the ranges **1** ➔ **9** or **A** ➔ **Y**
Returns	**/0 OK()<cr>** if successful, **/0 ERROR(11, <address>)<cr>** if **<address>** is invalid.
Example	To change an EvoDrive units address from 1 to B send **/1 Address(B)<cr>** This will return **/0 OK()<cr>** All subsequent commands must start with B. So to make the change permanent send **/B Save()<cr>**
Binary Cmd	**0x61**
Binary Data	1 Byte: Address as ASCII Character (e.g. 0x31 for '1')
Binary Reply	Status only

Set BAUD Rate

Format	**Baud(<1|2>, <BAUD>)**
Description	This command sets the baud rate for the specified channel. The new BAUD rate does not take effect until the EvoDrive is reset, therefore this must be committed using a **Save()** command. The default baud rate is 9600
Parameters	**<1|2>** The communications channel (1 or 2) to apply the new BAUD rate to. See section 3.1 for a definition of the communications channels. **<BAUD>** The new BAUD Rate for the channel. This can be 9600, 19200, 38400, 57600, 115200, 230400, 460800, or 921600.
Returns	**/0 OK()<cr>** if successful, **/0 ERROR(11, <param>)<cr>** if the parameter is invalid.
Example	To change the UART BAUD Rate to 115200, send **/1 Baud(2, 115200)<cr>** This will return **/0 OK()<cr>** To commit, send **/1 Save()<cr>** After reset, all subsequent commands will use the new BAUD Rate.
Binary Cmd	**0x67**
Binary Data	Send the ASCII string **(<1|2>, <BAUD>)**, including the brackets, as the data bytes in the binary command.
Binary Reply	Status only

Set Channel 2 Protocol

Format	**Prot(<1|2|3>)**
Description	Applies to EvoDrive ST-PCB only.

	This command sets the communications protocols to be used on the ST-PCB header pins. The new setting does not take effect until the EvoDrive is reset, therefore this must be committed using a **Save()** command. See the Integration Manual for a description of the header pin assignments and their alternate modes.
Parameters	**< 1|2|3 >** 1: UART and Digital Control 2: SPI Only 3: UART and partial Digital Control Input 4 acts as RS485 turn-around control.
Returns	**/0 OK()<cr>** if successful, **/0 ERROR(11, <param>)<cr>** if the parameter is invalid.
Example	To set the EvoDrive ST-PCB to use SPI only, send **/1 Prot(2)<cr>** This will return **/0 OK()<cr>** To commit, send **/1 Save()<cr>** After reset, the SPI pins are enabled and the UART and digital inputs are disabled.
Binary Cmd	**0x65**
Binary Data	1 bytes: The value of 1, 2 or 3.
Binary Reply	Status only

Set SPI Clock Polarity

Format	**CPOL(<0|1>)**
Description	Applies to EvoDrive ST-PCB only. This command informs the EvoDrive ST-PCB of the polarity of the master SPI clock. The new setting does not take effect until the EvoDrive is

	reset, therefore this must be committed using a **Save()** command. See Section 3.1 for a description of the SPI settings.
Parameters	**< 0|1 >** 0: Clock idle state is low 1: Clock idle state is high
Returns	**/0 OK()<cr>** if successful, **/0 ERROR(11, <param>)<cr>** if the parameter is invalid.
Example	If using an SPI clock with idle high, send **/1 CPOL(1)<cr>** This will return **/0 OK()<cr>** To commit, send **/1 Save()<cr>** After reset, the SPI will expect a clock with a high idle state.
Binary Cmd	**0x6E**
Binary Data	1 bytes: The value of 0 or 1
Binary Reply	Status only

Set SPI Clock Phase

Format	**CPHA(<0|1>)**
Description	Applies to EvoDrive ST-PCB only. This command informs the EvoDrive ST-PCB which clock edge to read the data on. The new setting does not take effect until the EvoDrive is reset, therefore this must be committed using a **Save()** command. See Section 3.1 for a description of the SPI settings.
Parameters	**< 0|1 >** 0: Data is read on the first clock edge 1: Data is read on the second clock edge.

Returns	**/0 OK()<cr>** if successful, **/0 ERROR(11, <param>)<cr>** if the parameter is invalid.
Example	If data should be read on the first clock edge, send **/1 CPHA(0)<cr>** This will return **/0 OK()<cr>** To commit, send **/1 Save()<cr>** After reset, the SPI will read and write data using the first clock edge.
Binary Cmd	**0x6F**
Binary Data	1 bytes: The value of 0 or 1
Binary Reply	Status only

Set User Resolution

Format	**Res(<UserSteps>)**
Description	This command defines the number of UserSteps in one revolution of the motor. See section 6 for an explanation of the unit UserSteps.
Parameters	**<UserSteps>** The number of UserSteps that make up one revolution of the motor.
Returns	**/0 OK()<cr>** if successful, **/0 ERROR(11, <param>)<cr>** if the parameter is invalid.
Example	To set the reference UserSteps to 1000, send **/1 Res(1000)<cr>** This will return **/0 OK()<cr>** All subsequent commands will now reference this new resolution.
Binary Cmd	**0x62**
Binary Data	Up to 4 bytes: The value of **<UserSteps>**

Binary Reply	Status only

Set Encoder Resolution

Format	**Enc(<EncRes>)**
Description	This command configures the resolution of the encoder, if any. Sending a resolution of 0 disables the encoder feedback. This command also determines the control mode. When an encoder is configured, the control mode is closed-loop. When there is no encoder, the control mode is open-loop. Changing the encoder resolution will disable the drive output. The EvoDrive must be re-enabled, to initialise the new encoder, using the **EN** command.
Parameters	**<EncRes>** The number of counts, after quadrature, per revolution of the encoder. Setting a value of 0 will disable the encoder.
Returns	**/0 OK()<cr>** if successful, **/0 ERROR(11, <param>)<cr>** if the parameter is invalid.
Example	To configure EvoDrive to use a 8192 count encoder, send **/1 Enc(8192)<cr>** This will return **/0 OK()<cr>** This will disable the drive output. Re-enable using the **EN** Command.
Binary Cmd	**0x64**
Binary Data	Up to 4 bytes: The value of **< EncRes >**
Binary Reply	Status only

Set Motor Step Size

Format	**Step(< 9 | 18 | 36 | 72 >)**
Description	To operate correctly the EvoDrive must know how many poles the stepper motor has. This command is used to configure the mechanical step size for the stepper motor being controlled. The default is 18 (1.8º stepper motor).
Parameters	**< MotorStep >** The degrees moved in one motor step expressed in tenths of degrees. i.e. 9 = 0.9º and 18 = 1.8º
Returns	**/0 OK()<cr>** if successful, **/0 ERROR(11,** <param>**)<cr>** if the parameter is invalid.
Example	To configure EvoDrive to use a 0.9º stepper motor, send **/1 Step(9)<cr>** This will return **/0 OK()<cr>** This will disable the EvoDrive output. Re-enable using the **EN** Command.
Binary Cmd	**0x63**
Binary Data	1 byte: The value of **< MotorStep >**
Binary Reply	Status only

Set PID Control Gains

Format	**PID(<PGain>, <IGain>, <DGain>)**
Description	This command sets the gain values for the PID Controller. For guidance on tuning the PID Gains see Section 8.6. PID Gains will only affect the motor response in closed loop mode.
Parameters	**< PGain >** The value of the Proportional Gain. **< IGain >** The value of the Integral Gain. **< DGain >** The value of the Differential Gain.

Returns	**/0 OK()<cr>** if successful, **/0 ERROR(11,** <param>**)<cr>** if a parameter is invalid.
Example	To change the PID Controller into a PI Controller with gains P = 100 and I = 20, send **/1 PID(100, 20, 0)<cr>** This will return **/0 OK()<cr>** The new tuning will be implemented immediately.
Binary Cmd	To set P Gain: **0x68** To set I Gain: **0x69** To set D Gain: **0x6A**
Binary Data	Up to 4 bytes: The value of the respective gain
Binary Reply	Status only

Enable Resonant Damping

Format	**DAMP(<1|0>)**
Description	This command enables or disables low speed resonant damping. For more information on resonant damping see Section 8.9.
Parameters	**< 1|0 >** Enable (1) or Disable (0) the resonant damping function.
Returns	**/0 OK()<cr>** if successful, **/0 ERROR(11, <param>)<cr>** if a parameter is invalid.
Example	To disable resonant damping, send **/1 DAMP(0) <cr>** This will return **/0 OK()<cr>** The change will be effective immediately.
Binary Cmd	**0x6D**
Binary Data	1 byte: The value of 1 or 0
Binary Reply	Status only

Set Collision Detection

Format	**Col(<ErrorBand>, <Time>, [Mode])**
Description	This only applies when running in closed loop mode. This command sets the threshold for collision detection. To trigger a collision alarm the motor must deviate from the target position by more than **<ErrorBand>** UserSteps. And be in error for more than **<Time>** milliseconds. When the collision alarm is raised, the motor will stop in a way defined by the optional parameter **[Mode]**: **0:** Set Velocity set point to 0. This implements a 'graceful' deceleration of the load. However if the collision is due to a faulty encoder it may cause the motor to spin unpredictably. **1:** Brake the motor. This turns off the motor current and allows the motor to regeneratively brake itself. This is the safest behaviour in most applications. If not defined, then the previous setting for **[Mode]** is unchanged.
Parameters	**<ErrorBand>** The error threshold in UserSteps. **<Time>** Milliseconds to allow the PID controller to correct the position before raising the alarm. **[Mode]** The behaviour mode after a collision.
Returns	**/0 OK() <cr>** if successful, **/0 ERROR(11,** <param>**)<cr>** if a parameter is invalid.
Example	To set the collision detection to 100 UserSteps and allow 2 seconds before stopping the motor, send **/1 Col(100, 2000)<cr>** This will return **/0 OK()<cr>**
Binary Cmd	To set Error Band: **0x6B** To set Error Time: **0x6C** To set Error Mode: **N/A**
Binary Data	Up to 4 bytes: The value of **<ErrorBand>** or **<Time>** respectively.
Binary Reply	Status only

Set Soft Limits

Format	**Limits(<LowLimit>, <HighLimit>)**
Description	This command sets soft limits for the motors motion. The motor movements will be restricted to the window defined by <LowLimit> and <HighLimit>. Motion into the envelope is allowed. These limits restrict Position, Velocity, and Digital Control modes. Motion is allowed in other modes, however the soft-limit error status is set when the movement goes outside the envelope. See section 8.6 for more information.
Parameters	**< LowLimit >** The lower limit of the motion envelope. **< HighLimit >** The upper limit of the motion envelope.
Returns	**/0 OK() <cr>** if successful, **/0 ERROR(11, <param>)<cr>** if a parameter is invalid.
Example	To set the allowed range of motion from -1000 to 0 steps, send **/1 Limits(-1000, 0)<cr>** This will return **/0 OK()<cr>**
Binary Cmd	To set Lower Limit: **0x77** To set Upper Limit: **0x78**
Binary Data	Up to 4 bytes: The value of **<LowLimit>** or **<HighLimit>**
Binary Reply	Status only

Set Polling Time

Format	**Poll(<PollTime>)**
Description	This command sets the input poll time. The poll time sets the rate at which inputs and triggers are monitored. The default poll time is 10 milliseconds. See section 8.8 for more information.
Parameters	**<PollTime>** The poll time in milliseconds. The valid range for poll time is 1ms to 250ms.
Returns	**/0 OK() <cr>** if successful, **/0 ERROR(11, <param>)<cr>** if a parameter is invalid.
Example	To set the poll time to 50 milliseconds, send **/1 Poll(50)<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x60**
Binary Data	Up to 4 bytes: The value of **<PollTime>**
Binary Reply	Status only

Set Status on Digital Outputs

Format	**DStat(<Output>, <StatusType>)**
Description	EvoDrive ST-17 and EvoDrive ST-23 only. This command configures the digital output <Output> to reflect the operating status <StatusType> of the EvoDrive. See section 9.2 for more information.
Parameters	**<Output>** The digital output to configure (1-4). **<StatusType>** The status type used to drive the output (1-5). 0 configures the output as a normal digital output.

Returns	**/0 OK() <cr>** if successful, **/0 ERROR(11, <param>)<cr>** if a parameter is invalid.
Example	To configure output 1 to close whenever the motor is moving, send **/1 DStat(1, 3)<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x5D**
Binary Data	Send the ASCII string **(<Output>, <StatusType>)**, including the brackets, as the data bytes in the binary command.
Binary Reply	Status only

Get Address

Format	**?ADDR**
Description	This command recovers the unit address of the EvoDrive. Because the address is required for commands, this query is normally sent with the broadcast address – **Z**. For example, **/Z ?ADDR**
Parameters	None.
Returns	**/0 OK(<address >)<cr>**
Binary Cmd	**0xE1**
Binary Data	None
Binary Reply	1 Byte: Address as ASCII Character (e.g. 0x31 for '1')

Get BAUD Rate

Format	**?BAUD(<1|2>)**
Description	This command recovers the BAUD Rate for the specified channel.
Parameters	**<1|2>** The communications channel (1 or 2) to query the BAUD rate. See section 3.1 for a definition of the communications channels.
Returns	**/0 OK(<baud>)<cr>**
Binary Cmd	**0xE7**
Binary Data	1 Byte: The value of the channel to query, 1 or 2.
Binary Reply	Up to 3 Bytes: The value of **<baud>**

Get Channel 2 Protocol

Format	**?PROT**
Description	Applies to EvoDrive ST-PCB only. This command returns the current value of the channel 2 protocol setting, as set by the **PROT** command.
Parameters	None.
Returns	**/0 OK(<1|2|3>)<cr>**
Binary Cmd	**0xE5**
Binary Data	None
Binary Reply	1 Byte: The value of the protocol setting, 1,2 or 3.

Get SPI Clock Polarity

Format	**?CPOL**
Description	Applies to EvoDrive ST-PCB only. This command returns the current value of the SPI clock polarity, as set by the **CPOL** command.
Parameters	None.
Returns	**/0 OK(<0|1>)<cr>**
Binary Cmd	**0xEE**
Binary Data	None
Binary Reply	1 Bytes: The value of the polarity setting, 0 or 1.

Get SPI Clock Phase

Format	**?CPHA**
Description	Applies to EvoDrive ST-PCB only.

	This command returns the current value of the SPI clock phase, as set by the **CPHA** command.
Parameters	None.
Returns	**/0 OK(<0|1>)<cr>**
Binary Cmd	**0xEF**
Binary Data	None
Binary Reply	1 Byte: The value of the phase setting, 0 or 1.

Get User Resolution

Format	**?RES**
Description	This command returns the current value of UserSteps, the configured number of UserSteps pre revolution set by the **RES** command.
Parameters	None.
Returns	**/0 OK(<UserSteps>)<cr>**
Binary Cmd	**0xE2**
Binary Data	None
Binary Reply	Up to 3 bytes: The value of **<UserSteps>**

Get Encoder Resolution

Format	**?ENC**
Description	This command returns the resolution of the encoder. This is the value configured by the **ENC** command.
Parameters	None.
Returns	**/0 OK(<EncRes>)<cr>**
Binary Cmd	**0xE4**
Binary Data	None

Binary Reply	Up to 3 bytes: The value of **< EncRes >**

Get Motor Step Size

Format	**?STEP**
Description	This command returns the motor step size in tenths of a degree. This is the value configured by the **STEP** command.
Parameters	None.
Returns	**/0 OK(< 9 | 18 | 36 | 72 >)<cr>**
Binary Cmd	**0xE3**
Binary Data	None
Binary Reply	1 Byte: The value of the motor step size.

Get PID Gains

Format	**?PID**
Description	This command returns the current PID Gains configured by the **PID** command.
Parameters	None.
Returns	**/0 OK(<PGain>, <IGain>, <DGain>)<cr>**
Binary Cmd	To query P Gain: **0xE8** To query I Gain: **0xE9** To query D Gain: **0xEA**
Binary Data	None
Binary Reply	Up to 4 bytes: The value of the respective gain.

Get Resonant Damping State

Format	**?DAMP**
Description	This command returns the enabled or disabled state of the low speed resonant damping function.
Parameters	None.
Returns	**/0 OK(<0|1>)<cr>**
Binary Cmd	**0xED**
Binary Data	None
Binary Reply	1 Byte: Value of 0 or 1.

Get Poll Time

Format	**?POLL**
Description	This command returns the poll time in milliseconds as configured by the **POLL** command.

Parameters	None.
Returns	**/0 OK(<PollTime>)<cr>**
Binary Cmd	**0xE0**
Binary Data	None
Binary Reply	1 Byte: The value of **< PollTime >.**

Get Soft Limits

Format	**?Limits**
Description	This command returns the current soft limit settings configured by the **Limits()** command.
Parameters	None.
Returns	**/0 OK(<LowLimit>, <HighLimit>)<cr>**
Binary Cmd	To query Low Limit: **0xF7** To query High Limit: **0xF8**
Binary Data	None
Binary Reply	Up to 4 bytes: The value of **<LowBand>** or **<HighLimit>** respectively**.**

Get Collision Thresholds

Format	**?COL**
Description	This command returns the current Collision thresholds configured by the **COL** command.
Parameters	None.
Returns	**/0 OK(<ErrorBand>, <Time>, <Mode>)<cr>**
Binary Cmd	To query Error Band: **0xEB** To query Error Time: **0xEC** To query Error Mode: **N/A**

Binary Data	None
Binary Reply	Up to 4 bytes: The value of **<ErrorBand>** or **<Time>** respectively**.**

Save the Current Configuration

Format	**Save()**
Description	This command writes all configuration values to non-volatile memory. This ensures the current configuration will be used as the default settings on power up. This command may interrupt the control system momentarily if executed while the motor is moving.
Parameters	None.
Returns	**/0 OK()<cr>** if successful, **/0 ERROR(<code>)<cr>** if unsuccessful.
Example	To save the current configuration to non-volatile memory, send **/1 Save()<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x7F**
Binary Data	None
Binary Reply	Status only

13.3 Current Limit Commands

These three stackable commands control the current thresholds used to drive the stepper motor. There are three tuneable levels

- Maximum Current: An over-current alarm threshold.
- Drive Current: Applies while the motor is moving.
- Holding Current: Applies while maintaining a position.

Set Max/Alarm Current

Format	**IX<mA>**
Description	This command sets the maximum allowable current the EvoDrive can generate. This value will limit the input of the **IH** and **ID** commands. It also sets the alarm threshold for an over-current alarm. So this value should be set higher than the expected drive current. The vector drive will be disabled if the drive current exceeds this value. Note that excessive current could be generated while braking or on start-up, so allow suitable margin when setting this value. The default maximum current is 3000 mA.
Parameters	**<mA>** The maximum current value in milliamps.
Returns	**/0 ERROR(11, <param>)<cr>** if the parameter is invalid.
Example	To set the maximum current threshold to 2A , send **/1 IX(2000)<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x75**
Binary Data	The value of **<mA>**
Binary Reply	Status only

Set Drive Current

Format	**ID<mA>**
Description	This command sets the maximum drive current that can be applied to move the motor. In open loop mode the drive current value will always be applied. In closed loop mode the drive current value is the maximum that can be applied. The default drive current is 1000 mA.
Parameters	**<mA>** The drive current value in milliamps.
Returns	**/0 ERROR(11, <param>)<cr>** if the parameter is invalid.
Example	To set the drive current to 1.7A , send **/1 ID(1700)<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x73**
Binary Data	The value of **<mA>**
Binary Reply	Status only

Set Holding Current

Format	**IH<mA>**
Description	This command sets the maximum holding current that can be applied to the motor while stationary. In open and closed loop mode the holding current will always be applied while the motor is holding position. This may generate excessive heat if set too high. The default holding current is 250 mA.
Parameters	**<mA>** The maximum holding value in milliamps.
Returns	**/0 ERROR(11, <param>)<cr>** if the parameter is invalid.

Example	To set the maximum holding current to 500mA , send **/1 IH(500)<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x74**
Binary Data	The value of **<mA>**
Binary Reply	Status only

Get Maximum Current

Format	**?IX**
Description	This command returns the over-current trip point set by the IX command.
Parameters	None.
Returns	**/0 OK(<mA>)<cr>** where <mA> is the maximum allowable current before triggering an over-current alarm.
Binary Cmd	**0xF5**
Binary Data	None
Binary Reply	Up to 2 bytes: The value of **<mA>**

Get Drive Current

Format	**?ID**
Description	This command returns the maximum drive current setting.
Parameters	None.
Returns	**/0 OK(<mA>)<cr>** where <mA> is the maximum allowable drive current in milliamps.
Binary Cmd	**0xF3**
Binary Data	None
Binary Reply	Up to 2 bytes: The value of **<mA>**

Get Holding Current

Format	**?IH**
Description	This command returns the maximum hold current setting.
Parameters	None.
Returns	**/0 OK(<mA>)<cr>** where <mA> is the maximum allowable hold current in milliamps.
Binary Cmd	**0xF4**
Binary Data	None
Binary Reply	Up to 2 bytes: The value of **<mA>**

13.4 Profile Commands

These commands set the acceleration, velocity, and deceleration values used to generate trajectory profiles.

These commands are stackable.

Set Profile Acceleration

Format	**PA<accel>**
Description	This command sets the profile acceleration used to achieve a target velocity or position.
Parameters	**<accel>** The acceleration value in UserSteps / sec^2.
Returns	**/0 ERROR(11, <param>)<cr>** **if the parameter is invalid.**
Example	To set the profile acceleration value to 1000 UserSteps/s^2, send **/1 PA1000<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x71**
Binary Data	Up to 4 bytes: The value of **<accel>**
Binary Reply	Status only

Set Profile Velocity

Format	**PV<vel>**
Description	This command sets the profile final velocity used to achieve a target position.
Parameters	**<vel>** The final velocity value in UserSteps / sec.
Returns	**/0 ERROR(11, <param>)<cr>** if the parameter is invalid.

Example	To set the profile final velocity value to 100 UserSteps/s^2, send **/1 PV100<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x70**
Binary Data	Up to 4 bytes: The value of **<vel>**
Binary Reply	Status only

Set Profile Deceleration

Format	**PD<decel>**
Description	This command sets the profile deceleration used to slow the motor to a lower velocity or a final position. The deceleration value is always positive, even though it effects a negative acceleration. The default deceleration value is 1 024 000. (10 revolutions per second^2)
Parameters	**<decel>** The deceleration value in UserSteps / sec^2.
Returns	**/0 ERROR(11, <param>)<cr>** if the parameter is invalid.
Example	To set the profile deceleration value to 500 UserSteps/s^2, send **/1 PA500<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x72**
Binary Data	Up to 4 bytes: The value of **<decel>**
Binary Reply	Status only

Get Profile Settings

Format	**?PROF**
Description	This command returns the current profile settings in UserStep units.
Parameters	None.
Returns	**/0 OK(<Accel>, <Vel>, <Decel>)<cr>**
Binary Cmd	To query profile acceleration: **0xF1** To query profile velocity: **0xF0** To query profile deceleration: **0xF2**
Binary Data	None
Binary Reply	Up to 4 bytes: The value of the respective profile setting.

13.5 Set Point Commands

These commands set the set-points used in the control system as detailed in Section 7.

The control system set points are dynamic, in that their value can be changed on the fly.

A set-point becomes active immediately after it is written. The active set-point can also be changed on the fly. When the active set-point is changed it overrides any previous set-point.

The motor will begin to work toward the active set-point immediately after the set-point is written to.

All set-point commands are stackable.

Position Set-Point (Absolute)

Format	**P<position>**
Description	This command sets the value of the Position set-point. A trajectory will be planned according to the profile values to drive to this position.
Parameters	**<position>** The position set-point as an absolute UserSteps value.
Returns	**/0 ERROR(11, <position>)<cr>** if **<position>** is invalid.
Example	To begin moving to position 320 000 UserSteps, send **/1 P320000<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x50**
Binary Data	Up to 4 bytes: The value of **<position>**
Binary Reply	Status only

Position Set-Point (Relative)

Format	**R<position>**
Description	This command sets the value of the Position set-point, relative to the current position. A trajectory will be planned according to the profile values to drive to this position.
Parameters	**<position>** The position set-point as a relative UserSteps value.
Returns	**/0 ERROR(11, <position>)<cr>** if **<position>** is invalid.
Example	To move from position 320 000 to 300 000 , send **/1 R-20000<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x52**
Binary Data	Up to 4 bytes: The value of **<position>**
Binary Reply	Status only

Velocity Set-Point

Format	**V<velocity>**
Description	This command sets the value of the Velocity set-point. The current profile will be used to accelerate the motor to this velocity. The motor will then run continuously.
Parameters	**<velocity>** The velocity set-point as a UserSteps/sec value.
Returns	**/0 ERROR(11, <velocity>)<cr>** if **<velocity>** is invalid.
Example	To spin the motor continuously at 500 UserSteps / sec, send **/1 V500<cr>** This will return **/0 OK()<cr>**

Binary Cmd	**0x56**
Binary Data	Up to 4 bytes: The value of **<velocity>**
Binary Reply	Status only

Acceleration Set-Point

Format	**A<accel>**
Description	This command sets the value of the acceleration set-point. The motor will accelerate continuously at the specified rate until the motor loses sync or it reaches the limits of its torque.
Parameters	**<accel>** The acceleration set-point as a UserSteps/s^2 value.
Returns	**/0 ERROR(11, <accel>)<cr>** if **<accel>** is invalid.
Example	To accelerate the motor continuously at 1234 UserSteps / s^2, send **/1 A1234<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x41**
Binary Data	Up to 4 bytes: The value of **<accel>**
Binary Reply	Status only

Torque Set-Point

Format	**T<torque>**
Description	This command sets the value of the Torque set-point. This torque will be applied to the rotor continuously.
Parameters	**<torque>** The torque set-point as a percentage of the total available torque. -100% to 100% Note that the total available torque (100%) is defined by the maximum drive current set by the **ID** command.
Returns	**/0 ERROR(11, <torque>)<cr>** if the torque value is invalid.
Example	To apply 50% torque to the rotor, send **/1 T50<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x54**
Binary Data	Up to 4 bytes: The value of **<torque>**
Binary Reply	Status only

13.6 Wait Commands

Stacked commands are normally executed immediately one after the other. The wait commands are used to pause execution of a stacked command string until a certain condition is met.

While waiting, the EvoDrive still accepts query commands (?... commands). If a non-query command is received then any waiting commands are cancelled and the new command will be executed immediately.

When the EvoDrive enters the first wait state, an OK Reply is sent to acknowledge the command.

Note: because the wait commands delay execution of stacked commands, they do not apply to the binary command format.

Wait for a Time

Format	**WT<ms>**
Description	This command pauses the execution of the command string until **<ms>** milliseconds has elapsed.
Parameters	**<ms>** The amount of time in milliseconds to wait before continuing execution.
Returns	**/0 ERROR(11, <ms>)<cr>** if **<ms>** is invalid.
Example	Accelerate the motor for 5 seconds, then maintain and report the speed, **/1 A1000 WT5000 A0 ?V<cr>** This will return, **/0 OK()<cr>** immediately **/0 OK(5015)<cr>** after 5 seconds.
Binary Cmd	N/A
Binary Data	N/A
Binary Reply	N/A

Wait for the motor to Stop

Format	**WS**
Description	This command will pause the execution of the command string until the motor is at a standstill. This command allows 500 milliseconds for the motor to start moving if it is starting from a standstill. The stopped state will be satisfied when the velocity is less than 0.2 revolutions per second.
Parameters	None
Returns	None
Example	To move to Position 10 000 then return to Position 0, send **/1 P10000 WS P0<cr>** This will return, **/0 OK()<cr>** immediately
Binary Cmd	N/A
Binary Data	N/A
Binary Reply	N/A

Wait for a Position

Format	**WP<pos>**
Description	This command will pause the execution of the command string until the motor passes position **<pos>**.
Parameters	**<pos>** The position in UserSteps to wait for before continuing execution.
Returns	**/0 ERROR(11, <pos>)<cr>** if **<pos>** is invalid.
Example	Spin the motor at a constant speed until at the home position, then slow down and return to home. **/1 V5000 WP0 P0<cr>** This will return, **/0 OK()<cr>** immediately
Binary Cmd	N/A
Binary Data	N/A
Binary Reply	N/A

Wait for a Digital Input - Negative Edge

Format	**WNI<input>**
Description	This command will pause the execution of the command string until a negative edge is seen on input **<input>**.
Parameters	**<input>** The digital input to watch. This can be 1, 2, 3, or 4.
Returns	**/0 ERROR(11, <input>)<cr>** if **<input>** is invalid.

Example	Spin the motor at a constant speed until a negative edge is seen on Input 1. Home to that position. **/1 V1000 WNI1 Z V0 WS P0 <cr>** This will return, **/0 OK()<cr>** immediately
Binary Cmd	N/A
Binary Data	N/A
Binary Reply	N/A

Wait for a Digital Input - Positive Edge

Format	**WPI<input>**
Description	This command pauses the execution of the command string until a positive edge is seen on input **<input>.**
Parameters	**<input>** The digital input to watch. This can be 1, 2, 3, or 4.
Returns	**/0 ERROR(11, <input>)<cr>** if **<input>** is invalid.
Example	Move to Position 500 000. But perform an emergency stop if a positive edge is seen on Input 3, **/1 P500000 WPI3 X<cr>** This will return, **/0 OK()<cr>** immediately Note: If the positive edge is not seen on input 3 then the next command received will cancel the wait state.
Binary Cmd	N/A
Binary Data	N/A
Binary Reply	N/A

13.7 Query Commands

The query commands return the value of a particular setting or parameter.

Get Status

Format	**?**
Description	This command returns the current status of the EvoDrive. This status includes motor direction, and status code.
Parameters	None.
Returns	**/0 OK(<dir>, <status>)<cr>** where **<dir>** is the direction of rotation. 1 is CW, -1 is CCW, 0 is stationary. **<status>** is the EvoDrive Status. See Section 11.2 for details.
Binary Cmd	**0x80**
Binary Data	None
Binary Reply	Status only

Get Active Set-Point

Format	**?SP**
Description	This command returns the active set-point type and its value.
Parameters	None.
Returns	**/0 OK(<type>, <value>)<cr>** where **<type>** is the set-point type: N = None, T = Torque, P = Position,

	V = Velocity, A = Acceleration **<value>** is the set-point value expressed in the applicable units.
Binary Cmd	N/A
Binary Data	N/A
Binary Reply	N/A

Get Current Position

Format	**?P**
Description	This command returns the current position in UserSteps. If operating in Open Loop mode then this position is the same as the current target position.
Parameters	None.
Returns	**/0 OK(<position>)<cr>**
Binary Cmd	**0xD0**
Binary Data	None
Binary Reply	Up to 4 bytes: The value of **<position>**

Get Current Velocity

Format	**?V**
Description	This command returns the current velocity in UserSteps/sec. If operating in Open Loop mode then this velocity is the same as the target velocity.
Parameters	None.
Returns	**/0 OK(<velocity>)<cr>**
Binary Cmd	**0xD6**

Binary Data	None
Binary Reply	Up to 4 bytes: The value of **<velocity>**

Get Current Acceleration

Format	**?A**
Description	This command returns the current acceleration in UserSteps/sec^2. If operating in Open Loop mode then this acceleration is the same as the target acceleration.
Parameters	None.
Returns	**/0 OK(<accel>)<cr>**
Binary Cmd	**0xC1**
Binary Data	None
Binary Reply	Up to 4 bytes: The value of **<accel>**

Get Current Torque

Format	**?T**
Description	This command returns the current torque output as a percentage of maximum. The sign of the value reflects the direction the torque is being applied in. If in open loop control then this value reflects the maximum possible torque the motor can produce given the motor currents. If in closed loop control then the value is the actual torque being applied to the rotor.
Parameters	None.
Returns	**/0 OK(<torque>)<cr>**

Example	**/0 OK(100)<cr>** indicates as much torque as possible is being applied in a clockwise direction. **/0 OK(-32)<cr>** indicates 32% of the maximum torque is being applied in the counter-clockwise direction.
Binary Cmd	**0xD4**
Binary Data	None
Binary Reply	Up to 4 bytes: The value of **<torque>**

Get Serial Number

Format	**?SER**
Description	This command returns the EvoDrive serial number. The serial number is unique to each EvoDrive unit and should be quoted when seeking service support.
Parameters	None.
Returns	**/0 OK(<Serial>)<cr>** where **<Serial>** is an Alphanumeric string up to 10 characters long.
Binary Cmd	**0x84**
Binary Data	None
Binary Reply	Returns **<Serial>** as a string of ASCII characters.

Get Firmware Version

Format	**?VER**
Description	This command returns the EvoDrive firmware version.
Parameters	None.
Returns	**/0 OK(<version>)<cr>** where **<version>** is in the format: V A.B.C **A** – is used for Compatibility. Firmware versions with the same A number are all compatible with the same

	hardware version. **B** – increments on the introduction of new features or a major release. **C** – increments on the introduction of minor releases and bug fixes.
Binary Cmd	**0x85**
Binary Data	None
Binary Reply	Returns **<version>** as a string of ASCII characters.

13.8 Digital I/O Commands

Digital output commands do not apply to EvoDrive ST-PCB.

Open Digital Output (High / Off)

Format	**DOP<output>**
Description	This command sets the output contact open. The outputs are active low, therefore this sets the output high, or off.
Parameters	**<output>** The output contact to open (1-4).
Returns	**/0 ERROR(11, <output>)<cr>** if the output number is invalid.
Example	**/1 DOP 1<cr>** will open output 1 (high / off). **/1 DOP 0<cr>** will return **/0 ERROR(11,0)**.
Binary Cmd	**0x5B**
Binary Data	1 byte: The value of **<output>**
Binary Reply	Status only

Close Digital Output (Low / On)

Format	**DCL<output>**
Description	This command sets the output pin closed. The outputs are active low, therefore this sets the output low, or on.
Parameters	**<output>** The output contact to close (1-4).
Returns	**/0 ERROR(11, <output>)<cr>** if the output number is invalid.
Example	**/1 DCL 4<cr>** will close output 4 (low / on). **/1 DCL 0<cr>** will return **/0 ERROR(11,0)**.
Binary Cmd	**0x5C**
Binary Data	1 byte: The value of **<output>**
Binary Reply	Status only

Set All Digital Outputs

Format	**DO<mask>**
Description	This command sets the digital outputs according to the bits in the **<mask>** value. **<mask>** is a decimal (4-bit) representation of the outputs. Its value is 0 ➔ 15 where Output 4 is represented by the MSB. A value of 1 in the **<mask>** closes the respective output (sets the pin low).
Parameters	**<mask>** decimal 4-bit value to write to the outputs.
Returns	**None.**
Example	**/1 DO 0<cr>** will turn off all outputs. **/1 DO 8<cr>** will turn output 4 on, all others off.
Binary Cmd	**0x5A**
Binary Data	1 byte: The value of **<mask>**
Binary Reply	Status only

Get Digital Outputs

Format	**?DO**
Description	This command gets the current value of the digital outputs expressed as a 4-bit decimal value. A value of 1 in the **<outputs>** value indicates the respective output is closed (low/on).
Parameters	None.
Returns	**/0 OK(<outputs >)<cr>** where **< outputs >** is a decimal representation of the outputs. Output 4 is the MSB**.**
Example	**/0 OK(7)<cr>** indicates outputs 1-3 are on, 4 is off.
Binary Cmd	**0xDA**
Binary Data	None
Binary Reply	1 Byte: The value of **<outputs>**

Get Digital Inputs

Format	**?DI**
Description	This command gets the current value of the digital inputs expressed as a 4-bit decimal value. Note: The digital inputs are assigned to digital control by default. They are also disabled when using the SPI protocol.
Parameters	None.
Returns	**/0 OK(<mask>)<cr>** where **<mask>** is a decimal representation of the inputs. Input 4 is the MSB.
Example	**/0 OK(14)<cr>** indicates inputs 2-4 are on, 1 is off.
Binary Cmd	**0xDB**
Binary Data	None
Binary Reply	1 Byte: The value of **<mask>**

13.9 Triggered Programs

The EvoDrive can hold 10 programs, up to 32 characters each, to be executed on certain triggers.

When set, these programs are always active and will execute on their trigger regardless of the current activity or motor state.

The program names and their triggers are listed below.

Name	Trigger	Default Program
S0	Power-on or Reset.	EN1
I0	Encoder Index Pulse	None
P1	Positive edge on Input 1	None
N1	Negative edge on Input 1	None
P2	Positive edge on Input 2	None
N2	Negative edge on Input 2	None
P3	Positive edge on Input 3	None
N3	Negative edge on Input 3	None
P4	Positive edge on Input 4	None
N4	Negative edge on Input 4	None

Triggered programs can be used to repeatedly zero the motor on a home sensor.

They can also be used to implement interlocks to control access to shared work zones or lock out safety systems.

The digital input programs can be used to configure the input for use as a digital input control. This is the default configuration of the ST-PCB. See Section 9.

The S0 program is used to execute initialisation commands such as enabling the motor and performing automatic homing.

All programs are saved to non-volatile memory by running the **Save()** command.

Set Program

Format	**Prog(<id>, [program])**
Description	This command sets a triggered program. The program can be up to 32 characters long and will be executed exactly the same way as any normal command string. The header, address and terminating carriage return characters should not be included in the program. By sending this command without a **[program],** the triggered program **<id>** is cleared.
Parameters	**<id>** Program name to configure. **[program]** Optional. Program to run when the trigger is seen. When omitted the program **<id>** is cleared.
Returns	**/0 OK()<cr>** if successful, **/0 ERROR(11, <param>)<cr>** if any parameter is invalid.
Example	To configure EvoDrive to zero its position every time a rising edge is seen on Input 1, send **/1 Prog(P1, Z)<cr>** This will return **/0 OK()<cr>** To configure EvoDrive to set output 3 high and stop the motor if there is a negative edge on Input 4, send **/1 Prog(N4, DO3 V0)<cr>** This will return **/0 OK()<cr>** To configure Input 2 to step the motor on a positive edge, send **/1 Prog(P2, #STEP)<cr>** This will return **/0 OK()<cr>** To stop program N4 executing, send **/1 Prog(N4)<cr>** This will return **/0 OK()<cr>**
Binary Cmd	**0x95**

Binary Data	Send the ASCII string **(<id>, [program])**, including the brackets, as the data bytes in the binary command. Note: only ASCII commands can be stored as triggered programs.
Binary Reply	Status only

Get Program

Format	**?Prog(<id>)**
Description	This query will return the commands saved to program **<id>**.
Parameters	**<id>** Program name to query.
Returns	**/0 OK(<program>)<cr>** if successful, **/0 ERROR(11, <id>)<cr>** if <id> is invalid.
Example	To query the default startup program, send **/1 ?Prog(S0)<cr>** This will return **/0 OK(EN1)<cr>** If a program does not have any commands to execute the query will return **/0 OK()<cr>**
Binary Cmd	**0x97**
Binary Data	Send the ASCII string **(<id>)**, including the brackets, as the data bytes in the binary command.
Binary Reply	Returns the program string as ASCII characters.

Run (Trigger) Program

Format	**Run(<id>)**
Description	This command will manually trigger the program **<id>** to execute.
Parameters	**<id>** Program name to trigger.

Returns	**/0 OK()<cr>** if successful, **/0 ERROR(11, <id>)<cr>** if **<id>** is invalid.
Example	To manually run the startup program, send **/1 Run(S0)<cr>** This will return **/0 OK()<cr>** Then Program S0 will execute.
Binary Cmd	**0x96**
Binary Data	Send the ASCII string **(<id>)**, including the brackets, as the data bytes in the binary command.
Binary Reply	Status only.

13.10 Miscellaneous Commands

Zero Position

Format	**Z**
Description	This stackable command will set the current position to 0. Note: This does not change any previous command. For example, if moving to position 1000, and this command was sent at position 400, The motor finish it's move but it will report it is at position 600.
Parameters	None
Returns	**/0 OK()<cr>**
Binary Cmd	**0x92**
Binary Data	None
Binary Reply	Status only

Scale Position

Format	**ZS**
Description	This stackable command will set the User Resolution so that the zero position (set by the Z command) and the current position **<pos>** are satisfied. Use this command to define how far the current position should be from the zero position. For example, An axis is commanded to move in percent units (0-100), one sensor may be used to set the 0 position (**Z**), and another sensor could be used to set the 100 percent position (**ZS100**). RES will be adjusted to fit that scale.
Parameters	**<pos>** Position value to scale to.
Returns	**/0 OK()<cr>**
Binary Cmd	**0x99**
Binary Data	Up to 4 bytes. The value of **<pos>**
Binary Reply	Status only

Emergency Stop

Format	**X**
Description	This command will immediately disable the vector drive (see **EN** Command) and regeneratively brake the motor. The vector drive output must be re-enabled to allow further movements.
Parameters	None.
Returns	**/0 OK()<cr>**
Binary Cmd	**0x94**
Binary Data	None
Binary Reply	Status only

Enable the Vector Drive

Format	**EN<1|0>**
Description	This command enables or disables the vector drive. When disabling the drive, the motor outputs are disabled and the motor is allowed to coast to a stop. When enabling the motor the following will occur - The encoder inputs are enabled if applicable - The encoder is initialised if applicable - The current set-point is cleared - The motor outputs are enabled, but until a set-point is given the motor is allowed to coast.
Parameters	**<1|0>** Enable(1) or Disable (0) the Vector Drive.
Returns	**/0 ERROR(<code>)<cr>** if an error occurs **/0 ERROR(11,<param>)<cr>** if the parameter is invalid
Example	To enable the vector drive, and apply a holding current, send **/1 EN1 P0<cr>**

	This will return **/0 OK()<cr>**
Binary Cmd	**0x93**
Binary Data	1 Byte: The value of 1 or 0
Binary Reply	Status only

Reset EvoDrive

Format	**Reset()**
Description	This command will reset the EvoDrive. The reset is equivalent to a full power-on.
Parameters	None
Returns	None
Binary Cmd	**0x91**
Binary Data	None
Binary Reply	None

Calibrate Motor

Format	**Cal()**
Description	This command will auto-tune the current control parameters to match your stepper motor. See Section 8.10 for more information.
Parameters	None
Returns	None
Binary Cmd	**0x98**
Binary Data	None
Binary Reply	Status Only

Read Configuration Variable

Format	**Read**(**<name>**)
Description	This command returns the value of the configuration variable. See Section 12 for a listing of variables.
Parameters	**<name>** The name of the configuration variable to read.
Returns	**/0 OK(<value>)<cr>** if successful, **/0 ERROR(11, <name>)<cr>** if **<name>** is invalid.
Example	To read the current value of the IOPOLL variable, send **/1 Read(IOPOLL)<cr>** This will return **/0 OK(10)<cr>**, by default.
Binary Cmd	N/A
Binary Data	N/A
Binary Reply	N/A

Write Configuration Variable

Format	**Write**(**<name>, <value>**)
Description	This command sets the configuration variable <name> to the value <value>. See Section 12 for a listing of variables. **Important**: This command does not check the value for correctness. If you are unsure about the correct value, use the applicable configuration command or EvoLink to set the value.
Parameters	**<name>** The name of the configuration variable to write. **<value>** The value to assign to the configuration variable.
Returns	**/0 OK()<cr>** if successful,

	/0 ERROR(11, <name|value>)<cr> if either parameter is invalid.
Example	To set the value of the IOPOLL to 20, send **/1 Write(IOPOLL , 20)<cr>** This will return **/0 OK()<cr>** Note: This change takes effect immediately, but it is not saved permanently. To make the change permanent send **/1 Save()<cr>**
Binary Cmd	N/A
Binary Data	N/A
Binary Reply	N/A

14. Quick Reference

- ASCII commands without a return value are stackable.
- ASCII commands with return values are non-stackable.
- All command strings return **/0 OK()** on their successful completion.
- All 'step' based units are in UserSteps (See **RES** Command).
- All binary commands must be wrapped in the binary command format (See section 5)

ASCII Command Error Reply: **/0 ERROR(10, <cmd>)**

ASCII Parameter Error Reply: **/0 ERROR(11, <param>)**

		ASCII (Terminal Mode)		Binary (OEM Mode)		
Description	**Page**	**Command**	**Returns**	**Command**	**Data Sent**	**Data Returned**
Set Point Commands						
Set absolute position	83	**P** <position>	**-**	**0x50**	<position>	Status only
Set relative position	84	**R** <position>	**-**	**0x52**	<position>	Status only
Set velocity	84	**V** <velocity>	**-**	**0x56**	<velocity>	Status only
Set acceleration	85	**A** <accel>	**-**	**0x41**	<accel>	Status only
Set Torque	86	**T** <torque>	**-**	**0x54**	<torque>	Status only
Profile Commands						
Acceleration Profile	80	**PA** <accel>	-	**0x71**	<accel>	Status only
Deceleration Profile	80	**PD** <decel>	-	**0x72**	<decel>	Status only
Velocity Profile	81	**PV** <vel>	-	**0x70**	<vel>	Status only
Save profile settings	75	**Save()**	/0 OK()	**0x7F**	-	Status only
Current Limit Commands						
Set the over-current trip point	76	**IX**<mA>	-	**0x75**	<mA>	Status only
Set the max holding current	77	**IH**<mA>	-	**0x74**	<mA>	Status only
Set the max drive current	77	**ID**<mA>	-	**0x73**	<mA>	Status only
Save the set point limits	75	**Save()**	/0 OK()	**0x7F**	-	Status only

		ASCII (Terminal Mode)		Binary (OEM Mode)		
Description	**Page**	**Command**	**Returns**	**Command**	**Data Sent**	**Data Returned**
Query Commands						
Get current status	91	**?**	/0 OK(<dir>, <status>)	**0x80**	-	Status only
Get current position	92	**?P**	/0 OK(<position)	**0xD0**	-	<position>
Get current velocity	92	**?V**	/0 OK(<velocity>)	**0xD6**	-	<velocity>
Get current acceleration	93	**?A**	/0 OK(<acceleration>)	**0xC1**	-	<acceleration>
Get current torque	93	**?T**	/0 OK(<torque>)	**0xD4**	-	<torque>
Get current set-point	91	**?SP**	/0 OK(<type>, <value>)	**-**	-	-
Get UserSteps per revolution	70	**?Res**	/0 OK(<UserSteps>)	**0xE2**	-	<UserSteps>
Get the encoder resolution	71	**?Enc**	/0 OK(<EncRes>)	**0xE4**	-	<EncRes>
Get the motor step size	72	**?STEP**	/0 OK(<9|18|36|72>)	**0xE3**	-	<9|18|36|72>
Get the PID Gains	73	**?PID**	/0 OK(<P>, <I>, <D>)	**0xE8**	-	<P>
				0xE9	-	<I>
				0xEA	-	<D>
Get the resonant damping state	73	**?DAMP**	/0 OK(1|0)	**0xED**	-	<1|0>
Get the error band, time, and mode	74	**?Col**	/0 OK(<Error>, <Time>, <Mode>)	**0xEB**	-	<Error>
				0xEC	-	<Time>
Get the soft limits	74	**?Limits**	/0 OK(<Low>, <High>)	**0xF7**	-	<LowLimit>
				0xF8	-	<HighLimit>

		ASCII (Terminal Mode)		Binary (OEM Mode)		
Description	**Page**	**Command**	**Returns**	**Command**	**Data Sent**	**Data Returned**
Get the current Profile values.	82	**?Prof**	/0 OK(<Accel>, <Vel>, <Decel>)	**0xF0** **0xF1** **0xF2**	- - -	<Vel> <Accel> <Decel>
Get the over-current threshold	78	**?IX**	/0 OK(<mA>)	**0xF5**	-	<mA>
Get the maximum holding current	79	**?IH**	/0 OK(<mA>)	**0xF4**	-	<mA>
Get the maximum drive current	78	**?ID**	/0 OK(<mA>)	**0xF3**	-	<mA>
Get Channel 2 Protocol	69	**?PROT**	/0 OK(<1|2|3>)	**0xE5**	-	<1|2|3>
Get SPI Clock Polarity	70	**?CPOL**	/0 OK(<0|1>)	**0xFE**	-	<0|1>
Get SPI Clock Phase	70	**?CPHA**	/0 OK(<0|1>)	**0xFF**	-	<0|1>
Get serial number	94	**?Ser**	/0 OK(<serial>)	**0x84**	-	<serial>
Get firmware version	94	**?Ver**	/0 OK(<version>)	**0x85**	-	<version>
Get unit address	69	**?ADDR**	/0 OK(<address>)	**0xE1**	-	<address>
Get BAUD rate	69	**?BAUD**(<1|2>)	/0 OK(<baud>)	**0xE7**	<1|2>	<baud>

I/O Commands						
Open Digital Output	96	**DOP**<output>	-	**0x5B**	<output>	Status only
Close Digital Output	96	**DCL**<output>	-	**0x5C**	<output>	Status only

		ASCII (Terminal Mode)		Binary (OEM Mode)		
Description	**Page**	**Command**	**Returns**	**Command**	**Data Sent**	**Data Returned**
Set All Digital Outputs	97	**DO**<mask>	-	**0x5A**	<mask>	Status only
Read digital outputs	97	**?DO**	/0 OK(<outputs>)	**0xDA**	-	<outputs>
Read digital inputs	96	**?DI**	/0 OK(<inputs>)	**0xDB**	-	<inputs>
Set the Status Output	67	**DStat**(<OP>, <Type>)	/0 OK()	**0x5D**	(<OP>, <Type>)	Status only

Wait Commands						
Wait for <time> milliseconds	87	**WT**<time>	-	**-**	-	-
Wait for the motor to stop	88	**WS**	-	**-**	-	-
Wait for absolute position <pos>	89	**WP**<pos>	-	**-**	-	-
Wait for a negative edge on input <input>	89	**WNI**<input>	-	**-**	-	-
Wait for a positive edge on input <input>	90	**WPI**<input>	-	**-**	-	-

Triggered Program Commands						
Set trigger program <id> to <cmd>	100	**Prog**(<id>,<cmd>)	/0 OK()	**0x95**	(<id>,<cmd>)	Status only
Read back triggered program	101	**?Prog**(<id>)	/0 OK(<cmd>)	**0x97**	(<id>)	Status only
Manual trigger of program	101	**Run**(<id>)	-	**0x96**	(<id>)	Status only
Save triggered programs.	75	**Save()**	/0 OK()	**0x7F**	-	Status only

		ASCII (Terminal Mode)		Binary (OEM Mode)		
Description	**Page**	**Command**	**Returns**	**Command**	**Data Sent**	**Data Returned**
Configuration Commands						
Set the unit address	57	**Address**(<address>)	/0 OK()	**0x61**	<address>	Status only
Set the BAUD rate	58	**Baud**(<1|2>,<baud>)	/0 OK()	**-**	-	-
Set Channel 2 Protocol	58	**PROT**(<1|2|3>)	/0 OK()	**0x65**	<1|0>	Status only
Set SPI Clock Polarity	59	**CPOL**(<0|1>)	/0 OK()	**0x6E**	<0|1>	Status only
Set SPI Clock Phase	60	**CPHA**(<0|1>)	/0 OK()	**0x6F**	<0|1>	Status only
Set the system resolution	58	**Res**(<UserSteps>)	/0 OK()	**0x62**	<UserSteps>	Status only
Set the encoder resolution	62	**Enc**(<EncRes>)	/0 OK()	**0x64**	<EncRes>	Status only
Set the number of motor steps pre rev	63	**Step**(<9|18|36|72>)	/0 OK()	**0x63**	<9|18|36|72>	Status only
Set the PID Gains	63	**PID**(<P>,<I>,<D>)	/0 OK()	**0x68**	<P>	Status only
				0x69	<I>	Status only
				0x6A	<D>	Status only
Enable resonant damping	64	**DAMP**(<1|0>)	/0 OK()	**0x6D**	<1|0>	Status only
Set the error band and time	65	**Col**(<Error>, <Time>, [Mode])	/0 OK()	**0x6B**	<Error>	Status only
				0x6C	<Time>	Status only
Set the input polling time	66	**Poll**(<PollTime>)	/0 OK()	**0x60**	<PollTime>	Status only
Set the soft limits	66	**Limits**(<Low>, <High>)	/0 OK()	**0x77**	<LowLimit>	Status only
				0x78	<HighLimit>	Status only
Save the Configuration	75	**Save()**	/0 OK()	**0x7F**	-	Status only

		ASCII (Terminal Mode)		Binary (OEM Mode)		
Description	**Page**	**Command**	**Returns**	**Command**	**Data Sent**	**Data Returned**
Other Commands						
Zero current position	103	**Z**	-	**0x92**	<mA>	Status only
Scale current position	103	**ZS**<Pos>	-	**0x99**	<Pos>	Status only
Emergency Stop	104	**X**	/0 OK()	**0x94**	<mA>	Status only
Enable / Disable vector drive	104	**En**<1|0>	-	**0x93**	<1|0>	Status only
Reset the EvoDrive	105	**Reset**()	-	**0x91**	-	-
Calibrate motor	105	**Cal()**	-	**0x98**	-	Status only
Read configuration variable	106	**Read(**<name>**)**	/0 OK(<value>)	**-**	-	-
Write configuration variable	106	**Write(**<name>**,** <value>**)**	/0 OK()	**-**	-	-

Binary Command Reference

Write Command	Query Command	Parameter Description	Equivalent ASCII Command	Page
-	0x80	Get current status	?	91
-	0x84	Get serial number	?Ser	94
-	0x85	Get firmware version	?Ver	94
0x41	0xC1	Acceleration set-point	A <accel>	85
0x50	0xD0	Absolute position set-point	P	83
0x52	0xD2	Relative position set-point	R	84
0x54	0xD4	Torque set-point	T <torque>	86
0x56	0xD6	Velocity set-point	V <velocity>	84
0x5A	0xDA	Set Digital Outputs	DO <mask>	97
0x5B	-	Open Digital Output	DOP <output>	96
0x5C	-	Close Digital Output	DCL <output>	96
0x5D	-	Digital Status Output	DStat(<OP>, <Type>)	67
-	0xDB	Read digital inputs	?DI	96
0x60	0xE0	Input polling time	Poll(<PollTime>)>	66
0x61	0xE1	Unit address	Address(<address>)	57
0x62	0xE2	System resolution (UserRes)	Res(<UserSteps>)	58
0x63	0xE3	Motor step size	Step(<9|18|36|72>)	63
0x64	0xE4	Encoder resolution	Enc(<EncRes>)	62
0x65	0xE5	Channel 2 Protocol	PROT(<1|2|3>)	58
0x67	0xE7	BAUD Rate	Baud(<1|2>,<baud>)	58
0x68	0xE8	P Gain (PID)	PID(<P>,<I>,<D>)	63
0x69	0xE9	I Gain (PID)	PID(<P>,<I>,<D>)	63
0x6A	0xEA	D Gain (PID)	PID(<P>,<I>,<D>)	63
0x6B	0xEB	Collision Error Band	Col(<Error>, <Time>)	65
0x6C	0xEC	Collision Error Time	Col(<Error>, <Time>)	65
0x6D	0xED	Resonant damping	DAMP(<0|1>)	64
0x6E	0xEE	SPI Clock Polarity	CPOL(<0|1>)	59
0x6F	0xEF	SPI Clock Phase	CPHA(<0|1>)	60
0x70	0xF0	Velocity Profile	PV <vel>	81
0x71	0xF1	Acceleration Profile	PA <accel>	80
0x72	0xF2	Deceleration Profile	PD <decel>	80
0x73	0xF3	Max drive current	ID<mA>	77
0x74	0xF4	Max holding current	IH<mA>	77
0x75	0xF5	Over-current alarm level	IX<mA>	76
0x76	0xF6	Collision Error Mode	Col(<Error>, <Time>, [Mode])	65
0x77	0xF7	Set Soft Limit (Low)	LIMIT(<Low>,<High>)	66

Write Command	Query Command	Parameter Description	Equivalent ASCII Command	Page
0x78	**0xF8**	Set Soft Limit (High)	**LIMIT(<Low>,<High>)**	66
0x7F	**-**	Save the Configuration	**Save()**	75
0x91	**-**	Reset the EvoDrive	**Reset()**	105
0x92	**-**	Zero current position	**Z**	103
0x93	**-**	Enable / Disable vector drive	**En**<1|0>	104
0x94	**-**	Emergency Stop	**X**	104
0x95	**-**	Set Triggered Program	**Prog**(<id>,<cmd>)	100
0x96	**-**	Run Triggered Program	**Run**(<id>)	101
0x97	**-**	Read Triggered Program	**?Prog**(<id>)	101
0x98	**-**	Calibrate Motor	**CAL()**	105
0x99	**-**	Scale to current position	**ZS**	103