mks

XPS-RL

Universal High-Performance Motion Controller/Driver

ESP COMPATIBLE

RoHS Compliant

Newport®

User's Manual

V1.0.x

Warranty

Newport Corporation warrants that this product will be free from defects in material and workmanship and will comply with Newport's published specifications at the time of sale for a period of one year from date of shipment. If found to be defective during the warranty period, the product will either be repaired or replaced at Newport's option.

To exercise this warranty, write or call your local Newport office or representative, or contact Newport headquarters in Irvine, California. You will be given prompt assistance and return instructions. Send the product, freight prepaid, to the indicated service facility. Repairs will be made and the instrument returned freight prepaid. Repaired products are warranted for the remainder of the original warranty period or 90 days, whichever comes first.

Limitation of Warranty

The above warranties do not apply to products which have been repaired or modified without Newport's written approval, or products subjected to unusual physical, thermal or electrical stress, improper installation, misuse, abuse, accident or negligence in use, storage, transportation or handling. This warranty also does not apply to fuses, batteries, or damage from battery leakage.

THIS WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR USE. NEWPORT CORPORATION SHALL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM THE PURCHASE OR USE OF ITS PRODUCTS.

Original instructions.

Table of Contents

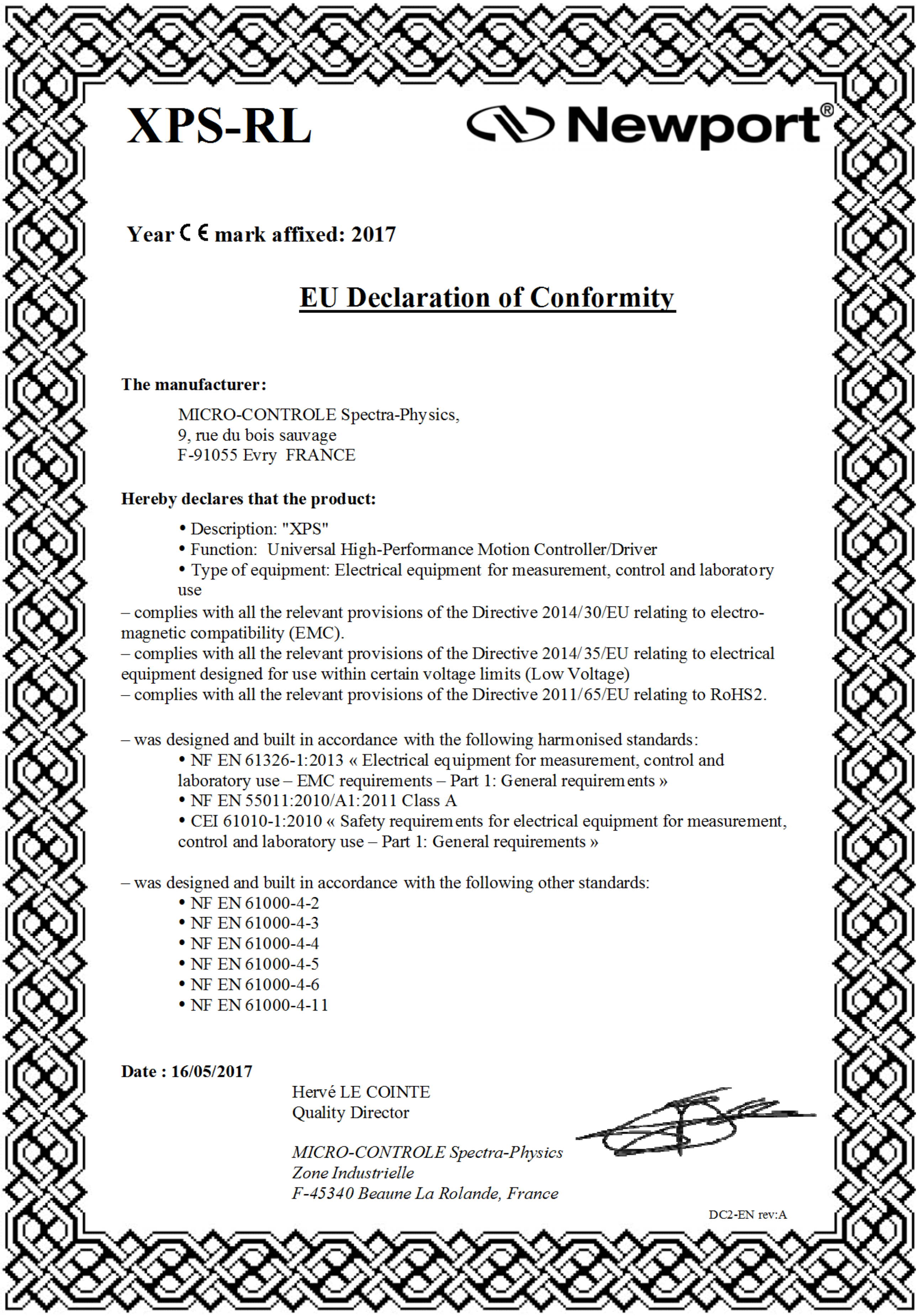
User's Manual

Software Tools

Motion Tutorial

Appendix

EU Declaration of Conformity



XPS-RL Newport®

Year CE mark affixed: 2017

EU Declaration of Conformity

The manufacturer:

MICRO-CONTROLE Spectra-Physics,
9, rue du bois sauvage
F-91055 Evry FRANCE

Hereby declares that the product:

- Description: "XPS"
- Function: Universal High-Performance Motion Controller/Driver
- Type of equipment: Electrical equipment for measurement, control and laboratory use

– complies with all the relevant provisions of the Directive 2014/30/EU relating to electro-magnetic compatibility (EMC).
– complies with all the relevant provisions of the Directive 2014/35/EU relating to electrical equipment designed for use within certain voltage limits (Low Voltage)
– complies with all the relevant provisions of the Directive 2011/65/EU relating to RoHS2.

– was designed and built in accordance with the following harmonised standards:

- NF EN 61326-1:2013 « Electrical equipment for measurement, control and laboratory use – EMC requirements – Part 1: General requirements »
- NF EN 55011:2010/A1:2011 Class A
- CEI 61010-1:2010 « Safety requirements for electrical equipment for measurement, control and laboratory use – Part 1: General requirements »

– was designed and built in accordance with the following other standards:

- NF EN 61000-4-2
- NF EN 61000-4-3
- NF EN 61000-4-4
- NF EN 61000-4-5
- NF EN 61000-4-6
- NF EN 61000-4-11

Date : 16/05/2017

Hervé LE COINTE
Quality Director

MICRO-CONTROLE Spectra-Physics
Zone Industrielle
F-45340 Beaune La Rolande, France

DC2-EN rev:A

Preface

Confidentiality & Proprietary Rights

Reservation of Title

The Newport Programs and all materials furnished or produced in connection with them (“Related Materials”) contain trade secrets of Newport and are for use only in the manner expressly permitted. Newport claims and reserves all rights and benefits afforded under law in the Programs provided by Newport Corporation.

Newport shall retain full ownership of Intellectual Property Rights in and to all development, process, align or assembly technologies developed and other derivative work that may be developed by Newport. Customer shall not challenge, or cause any third party to challenge, the rights of Newport.

Preservation of Secrecy and Confidentiality and Restrictions to Access

Customer shall protect the Newport Programs and Related Materials as trade secrets of Newport, and shall devote its best efforts to ensure that all its personnel protect the Newport Programs as trade secrets of Newport Corporation. Customer shall not at any time disclose Newport's trade secrets to any other person, firm, organization, or employee that does not need (consistent with Customer's right of use hereunder) to obtain access to the Newport Programs and Related Materials. These restrictions shall not apply to information (1) generally known to the public or obtainable from public sources; (2) readily apparent from the keyboard operations, visual display, or output reports of the Programs; (3) previously in the possession of Customer or subsequently developed or acquired without reliance on the Newport Programs; or (4) approved by Newport for release without restriction.

Sales, Tech Support & Service

North America & Asia
Newport Corporation
1791 Deere Ave.
Irvine, CA 92606, USA

Sales
Tel.: (877) 835-9620
e-mail: sales@newport.com

Technical Support
Tel.: (800) 222-6440
e-mail: tech@newport.com

Service, RMAs & Returns
Tel.: (800) 222-6440
e-mail: service@newport.com

Europe
MICRO-CONTROLE Spectra-Physics S.A.S
9, rue du Bois Sauvage
91055 Évry CEDEX
France

Sales France
Tel.: +33 (0)1.60.91.68.68
e-mail: france@newport.com

Sales Germany
Tel.: +49 (0) 61 51 / 708 – 0
e-mail: germany@newport.com

Sales UK
Tel.: +44 (0)1635.521757
e-mail: uk@newport.com

Technical Support
e-mail: tech_europe@newport.com

Service & Returns
Tel.: +33 (0)2.38.40.51.55

Service Information

The user should not attempt any maintenance or service of the XPS Series Controller/Driver system beyond the procedures outlined in this manual. Any problem that cannot be resolved should be referred to Newport Corporation. When calling Newport regarding a problem, please provide the Tech Support representative with the following information:

- Your contact information.
- System serial number or original order number.
- Description of problem.
- Environment in which the system is used.
- State of the system before the problem.
- Frequency and repeatability of problem.
- Can the product continue to operate with this problem?
- Can you identify anything that may have caused the problem?

Newport Corporation RMA Procedures

Any XPS Series Controller/Driver being returned to Newport must be assigned an RMA number by Newport. Assignment of the RMA requires the item’s serial number.

Packaging

XPS Series Controller/Driver being returned under an RMA must be securely packaged for shipment. If possible, re-use the original packaging.

User's Manual

1.0 Introduction

1.1 Scope of the Manual

The XPS is an extremely high-performance, easy to use, integrated motion controller/driver offering high-speed communication through 10/100/1000 Base-T Ethernet, outstanding trajectory accuracy and powerful programming functionality. It combines user-friendly web interfaces with advanced trajectory and synchronization features to precisely control from the most basic to the most complex motion sequences. Multiple digital and analog I/O's, triggers and supplemental encoder inputs provide users with additional data acquisition, synchronization and control features that can improve the most demanding motion applications.

To maximize the value of the XPS Controller/Driver system, it is important that users become thoroughly familiar with available documentation:

The **XPS Quick Start** is delivered as a hard copy with the controller.

The XPS-RL User's Manual, Programmer's and Software Drivers manuals are PDF files accessible from the controller disk which can be downloaded from the controller website under the tab Documentation.

.NET assemblies and corresponding sources are available from the controller disk which can be downloaded from the controller website under the tab Documentation -> Drivers & Examples.

LabVIEW VIs with examples are also available to download from the Newport website.

The first part of this manual serves as an introduction and also as a reference. It includes:

1. Introduction
2. System Overview
3. Getting Started Guide

The second part provides a detailed description of all software tools of the XPS controller. It also includes an introduction to FTP connections and some general guidelines for troubleshooting, maintenance and service:

4. Software Tools

5. Maintenance and Service

The third part provides an exhaustive description of the XPS architecture, its features and capabilities. Complementing the programmer's guide, this part is educational and is organized by features starting with the basics and getting to the more advanced features. It provides a complete list of descriptions of different features including:

6. XPS Architecture

7. Motion

8. Trajectories

9. Emergency Brake and Emergency Stop

10. Compensation

11. Event Triggers

12. Data Gathering

13. Output Triggers

14. Control Loops

15. Analog Encoder Calibration

16. Excitation Signal

17. 17. Pre-Corrector Excitation Signal

18. Introduction to XPS programming

1.2 Definitions and Symbols

The following terms and symbols are used in this documentation and also appear on the XPS Series Controller/Driver where safety-related issues occur.

1.2.1 General Warning or Caution

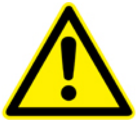


Figure 1: General warning or caution symbol.

The Exclamation Symbol in Figure 1 may appear in Warning and Caution tables in this document. This symbol designates an area where personal injury or damage to the equipment is possible.

1.2.2 Electric Shock

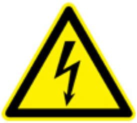


Figure 2: Electrical shock symbol.

The Electrical Shock Symbol in Figure 2 may appear on labels affixed to the XPS Series Controller/Driver. This symbol indicates a hazard arising from dangerous voltages. Any mishandling could result in damage to the equipment, personal injury, or even death.

1.2.3 European Union CE Mark

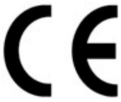


Figure 3: CE mark.

The presence of the CE Mark on Newport Corporation equipment means that it has been designed, tested and certified to comply with all current and applicable European Union (CE) regulations and recommendations.

1.2.4 "ON" Symbol

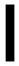


Figure 4: "ON" symbol.

The "ON" Symbol in Figure 4 appears on the power switch of the XPS Series Controller/Driver. This symbol represents the "Power On" condition.

1.2.5 "OFF" Symbol

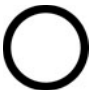


Figure 5: "OFF" symbol.

The "Off" Symbol in Figure 5 appears on the power switch of the XPS Series Controller/Driver. This symbol represents the "Power Off" condition.

1.3 Warnings and Cautions

The following are definitions of the Warnings, Cautions and Notes that may be used in this manual to call attention to important information regarding personal safety, safety and preservation of the equipment, or important tips.

⚠

WARNING

Situation has the potential to cause bodily harm or death.

⚠

CAUTION

Situation has the potential to cause damage to property or equipment.

⚠

WARNING

This product is equipped with a 3-wire grounding type plug. Any interruption of the grounding connection can create an electric shock hazard. If you are unable to insert the plug into your wall plug receptacle, contact an electrician to perform the necessary alterations to ensure that the green (green-yellow) wire is attached to earth ground.

System earthing must be of type earthed neutral (TN-) as defined by CEI60364.

NOTE

Additional information the user or operator should consider.

1.4 General Warnings and Cautions

The following general safety precautions must be observed during all phases of operation of this equipment.

Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and the intended use of the equipment.

- Heed all warnings on the unit and in the operating instructions.
- To prevent damage to the equipment, read the instructions in this manual for the selection of the proper input voltage.
- Only plug the Controller/Driver unit into a grounded power outlet.
- Ensure that the equipment is properly grounded to earth ground through the grounding lead of the AC power connector.
- Route power cords and cables where they are not likely to be damaged.
- **Use Proper Power Cord**
  Use only the power cord specified for this product and certified for the country of use.
- The system must be installed in such a way that the power switch and the power connector remain accessible to the user.
- Disconnect or do not plug-in the AC power cord under the following conditions:
  - If the AC power cord or any other attached cables are frayed or damaged.
  - If the power plug or receptacle is damaged.
  - If the unit is exposed to rain or excessive moisture, or liquids are spilled on it.

  - If the unit has been dropped or the case is damaged.
  - If the user suspects service or repair is required.
- Keep air vents free of dirt and dust and obstructions.
- Keep liquids away from unit.
- Do not expose equipment to excessive moisture (>85% humidity).
- Do not operate this equipment in an explosive atmosphere.
- Disconnect power before cleaning the Controller/Driver unit. Do not use liquid or aerosol cleaners.
- Do not open the XPS Controller/Driver stand alone motion controller. There are no user-serviceable parts inside the XPS Controller/Driver.
- Return equipment to Newport Corporation for service and repair.
- Dangerous voltages associated with the 100–240 VAC power supply are present inside Controller/Driver unit. To avoid injury, do not touch exposed connections or components while power is on.
- Follow precautions for static-sensitive devices when handling electronic circuits.

2.0 System Overview

2.1 Specifications

Number of Axes	• 1 to 4 axes of stepper, DC brush, DC brushless motors or piezo-electric stacks using internal drives • Other motion devices using external third-party drives
Communication Interfaces	• Internet protocol TCP/IP • One Ethernet 10/100/1000 Base-T (RJ45 connector) with fixed IP address and DHCP server for local communication • One Ethernet 10/100/1000 Base-T (RJ45 connector) for networking, dynamic addressing with DHCP and DNS • Typically 0.3 ms from sending a tell position command to receiving the answer
Firmware Features	• Powerful and intuitive, object oriented command language • Native user defined units (no need to program in encoder counts) • Real time execution of custom tasks using TCL scripts • Multi-user capability • Concept of sockets for parallel processes • Distance spaced trigger output pulses, max. 1.6 MHz rate (5 MHz for less than 4096 pulses), programmable filter • Array-based compensated position trigger output pulses, max 1.6 MHz (5 MHz for less than 4096 pulses) • Time spaced trigger output pulses, 0.05 Hz to 20 MHz, 5 ns accuracy • Trigger output on trajectories with 125 µs resolution • Data gathering at up to 8 kHz rate, up to 1,000,000 data entries • User-defined “actions at events” monitored by the controller autonomously at the servo rate • User-definable system referencing with hardware position latch of reference signal transition and “set current position to value” capability • Axis position or speed controlled by analog input • Axis position, speed or acceleration copied to analog output • Trajectory precheck function replying with travel requirement and max. possible speed • Auto-configuration, auto-tuning and auto-scaling
Motion	• Jogging mode including on-the fly changes of speed and acceleration • Synchronized point-to-point • Spindle motion (continuous motion with periodic position reset) • Line-arc mode (linear and circular interpolation incl. continuous path contouring) • Splines (Catmull-Rom type) • PVT (complex trajectory based on position, velocity and time coordinates) • Analog tracking (using analog input as position or velocity command) • Master-slave including single master-multiple slaves and custom gear ratio
Compensation	• Linear error, Backlash, positioner error mapping • XY and XYZ error mapping • All corrections are taken into account on the servo loop
Servo Rate	• Adjustable with default value 8 kHz

I/O	• Basic GPIO: – 8 TTL inputs and 8 TTL outputs (open collector) – 2 synch. analog inputs ±10 V, 12 bits – 2 synch. analog outputs ±10 V, 12 bits • Extended GPIO: – 40 TTL inputs and 40 TTL outputs (open collector) – 8 synch. analog inputs ±10 V, 16 bits – 8 synch. analog outputs ±5 V, ±10 V or ±12.288 V (configurable), 16 bits
Control Loop	• Open loop, PI position, PIDFF velocity, PIDFF acceleration, PIDDualFF voltage • Variable PID's (PID values depending on distance to target position) • Deadband threshold; Integration limit and integration time • Derivative cut-off filter; 2 user-defined notch filters
Trigger In	• Hardware latch of all positions and all analog I/O's; 8 kHz max. frequency • <50 ns latency on positions • <125 µs time jitter on analog I/O's
Trigger Out	• One high-speed position compare output only for axes 1 and 2 that can be either configured for position synchronized pulses or for time synchronized pulses : 5 ns accuracy, <700 ns latency (from real stage position to pulse generation), 5 MHz max frequency • Basic PCO: – interpolation x256 – not compensated • Extended PCO: – interpolation x65536 – compensated
Dedicated Inputs Per Axis	• RS-422 differential inputs for A, B and I, Max. 25 MHz, over-velocity and quadrature error detection • 1 Vpp analog encoder input up to x65536 interpolation used for servo; amplitude, phase and offset correction and synchronisation • Forward and reverse limit, home, error input
Dedicated Outputs Per Axis (when using external drives)	• 2 channel 16-bit, ±10 V D/A • Drive enable, error output
Drive Capability	• Analog voltage, analog velocity, and analog acceleration (used with XPS-DRV01 and XPS-DRV03 for DC brush motor control). • Analog position (used with XPS-DRV01 for stepper motor control or with the XPS-DRVP1 for piezo control) • Analog position (used with external drives for example 3rd party motors) • Analog acceleration, sine acceleration and dual sine acceleration (used with XPS-DRV02 for brushless motors control) • Step and direction and ± pulse mode for stepper motors (requires XPS-DRV00P and external stepper motor driver) • 300 W @ 230 VAC total available power
AC Power Requirements	• 100–240 VAC 60/50 Hz 7.5 A–3.8 A The controller should be connected to a power installation that incorporates appropriate protection devices. Refer to the installation requirements of your facility and local applicable Standards concerning the use of RCDs (residual current device).
Dimensions (W x D x H)	• 12.61 x 13.39 x 6.94 [4U] in. (320.4 x 340 x 176.3 mm). The chassis is compatible to an 19" rack with the purchase of additional hardware.
Weight	• 17.2 lb (7.8 kg) max.

Environmental Condition	• Internal use
Overvoltage Category	• Category II (2500 V max transient surges)
Operating Temperature	• +5°C to +40°C (Derate 1.5%/Degree above 40°C)
Humidity	• 85% R.H. at 40°C (Non-condensing)
Altitude	• Above 1000 m, derate at 1% per 100 m. Max. 2000 m
Storage Temperature	• -20°C to +55°C (Non-condensing)
Pollution	• Pollution Degree 2, exempt of conducting dust
Transport Temperature	• -20°C to +70°C (Non-condensing)

2.2 Drive Options

The XPS controller is capable of driving up to 4 axes of most Newport positioners using driver cards that slide through the back of the chassis. These factory-tested drives are powered by an internal 300 W power supply, which is independent of the controller power supply. When used with Newport ESP stages, the configuration of the driver cards is easy using the auto-configuration utility software. Advanced users can also manually develop their own configuration files optimized for specific applications.

The XPS-DRV01 is a software configurable PWM amplifier that is compatible with most of Newport's and other companies' DC brush and stepper motor positioners.

The XPS-DRV01 motor driver supplies a maximum current of 3 A and 48 V. It has the capability to drive bipolar stepper motors in microstep mode (sine/cosine commutation) and DC brush motors in velocity mode, for motors with tachometer, or voltage mode, for motors without tachometer. Programmable gains and a programmable PWM switching frequency up to 300 kHz allow a very fine adjustment of the driver to the motor. For added safety, a programmable over-current protection setting is also available.

The XPS-DRV02 is a software configurable PWM amplifier for 3-phase brushless motors. It has been optimized for performance with XM, ILS-LM, IMS-LM linear motor stages and RGV direct drive rotation stages. The XPS-DRV02 supplies a 100 kHz PWM output with a maximum output current of 5 A per phase and 44 Vpp. The XPS-DRV02 requires 1 Vpp analog encoder input signals used also for motor commutation. Motor initialization is done by a special routine measures the magnetic position without the need for Hall or other sensors.

The XPS-DRV03 is a fully digital, programmable PWM-amplifier that has been optimized for use with high-performance DC motors. The high switching frequency of 100 kHz and appropriate filter technologies minimize noise to enable ultra-precision positioning in the nm-range. The XPS-DRV03 supplies a maximum current of 5 A and 48 V. It is capable of driving DC motors in velocity mode (for motors with tachometer), in voltage mode (for motors without tachometer), and in current mode (for torque motors). All parameters are programmable in physical units (for instance the bandwidth of the velocity loop). Furthermore, the XPS-DRV03 features individual limits for the rms current and the peak current.

The XPS-DRVP1 is a programmable driver card for Newport's NanoPositioning line of piezoelectric stack stages. This driver card has a range of -10 to 150 VDC with 30 mA continuous. The drive features a 4 kHz update rate and resolution of 16 bits ADC and DAC. It also accepts strain gage position feedback.

The XPS-DRV00 and XPS-DRV00P pass-through module can be used to pass control signals to other external third-party amplifiers (drivers). By setting the controller's dual DAC output to either analog position, analog stepper position, analog velocity, analog voltage or analog acceleration (including sine commutation), the XPS is capable of controlling almost any motion device including 3rd party brushless motors and voice coils.

In addition to conventional digital AquadB feedback encoder interface, the XPS controller also features a high-performance analog encoder input (1 Vpp Heidenhain standard) on each axis. An ultra-high resolution, very low noise, encoder signal interpolator converts the sine-wave input to an exact position value with a signal subdivision up to 65536-fold. For example, when used with a scale with 4 µm signal period the resolution can be as fine as 0.061 nm. This interpolator can be used for accurate position feedback on the servo corrector of the system or for synchronization purposes with an accuracy of 5 ns and a latency inferior to 700 ns from real stage position to pulse generation. Unlike most high-resolution multiplication devices, the XPS interpolators do not compromise positioning speed. With a maximum input frequency of 2.4 MHz, the maximum speed of a stage with a 20 µm signal period scale can be up to 48 m/s.

2.3 Compatible Newport Positioners and Drive Power Consumption

The list of all compatible Newport positioners and the corresponding drive module needed is available from the Newport catalog or at www.newport.com

2.4 XPS Hardware Overview

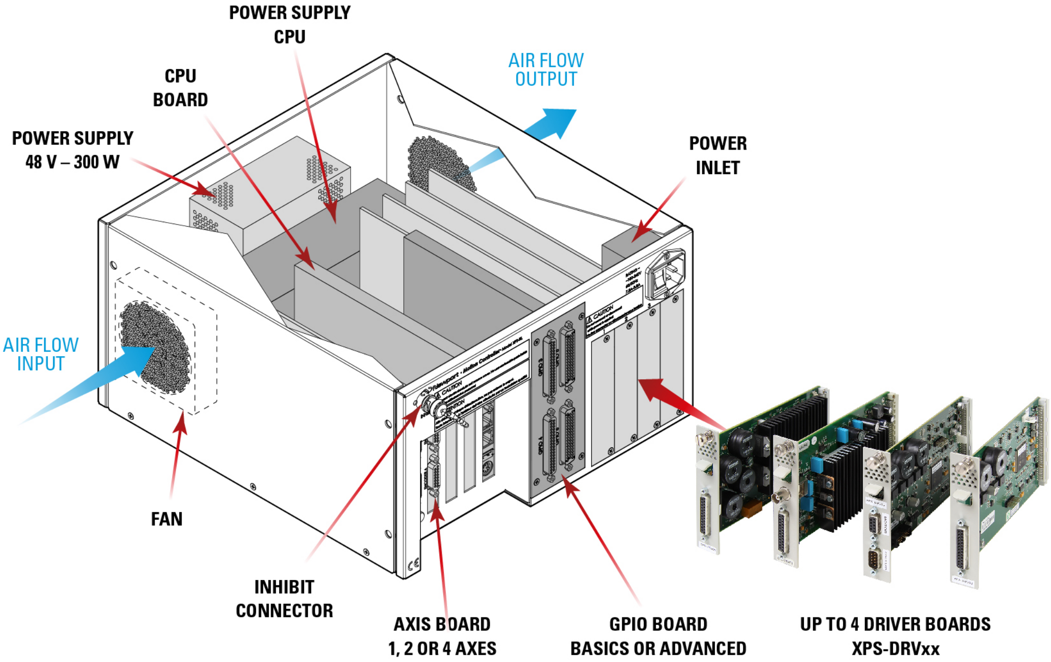


Figure 6: XPS hardware overview.

2.5 Rear Panel Description

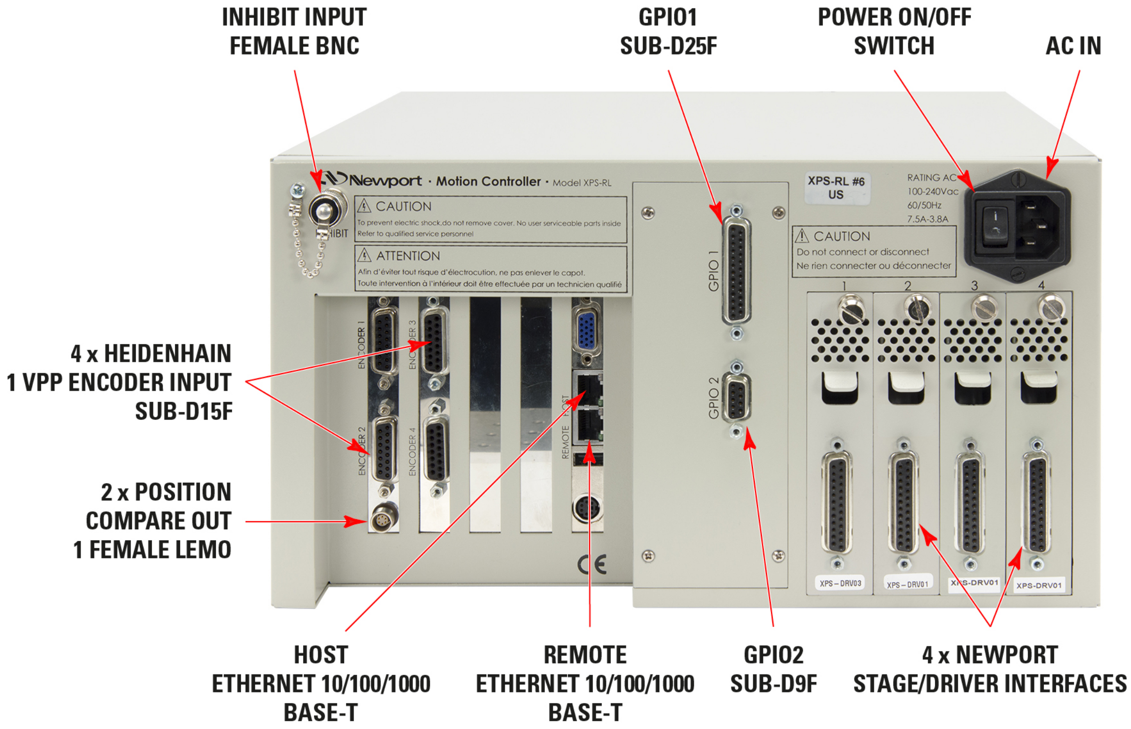


Figure 7: Rear panel of XPS Controller/Driver with basic GPIO.

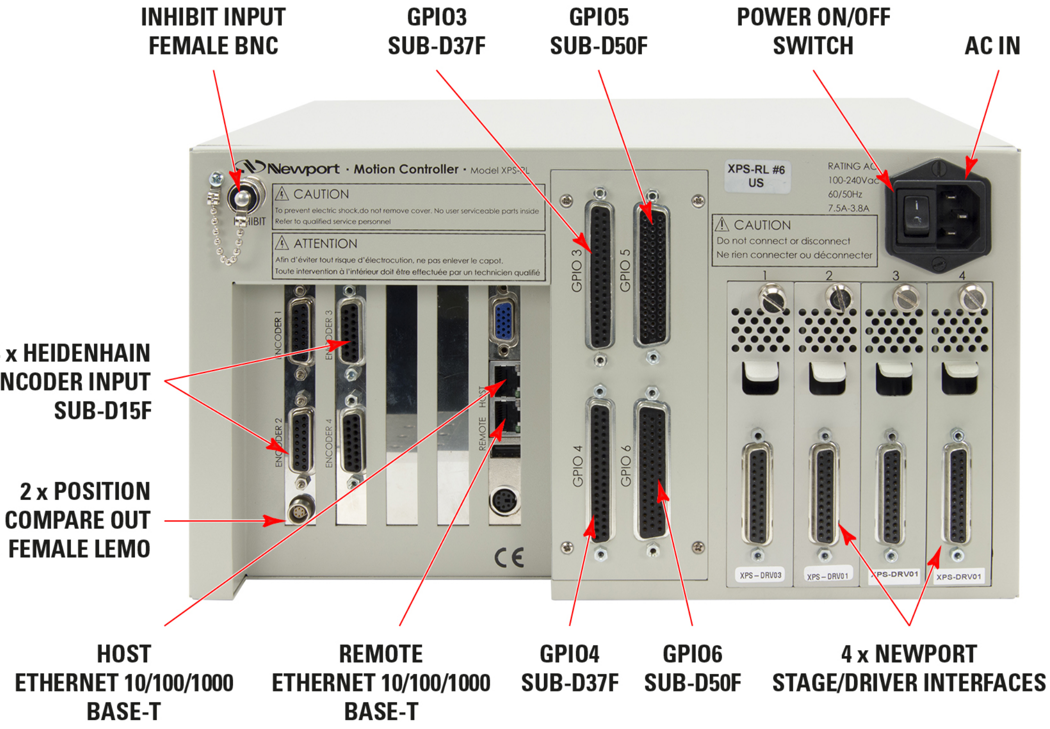


Figure 8: Rear panel of XPS Controller/Driver with extended GPIO.

NOTE

The XPS-RL USB plug is not activated.

NOTE

The Main Power ON/OFF Switch is located next to the inlet for the power cord. The switch and the inlet must be accessible to the user.

The AC power cable or the Inhibit Input are the product's main disconnect device and must be easily accessible at all times.

2.5.1 Axis Connectors (AXIS 1 – AXIS 4)

Each installed axis driver card features a connector to attach a cable (supplied with every Newport stage) between the controller and a motion device.

CAUTION

Carefully read the labels on the driver cards and make sure the specifications (motor type, voltage, current, etc.) match those of the motion devices you intend to connect. Severe damage could occur if a stage is connected to the wrong driver card.

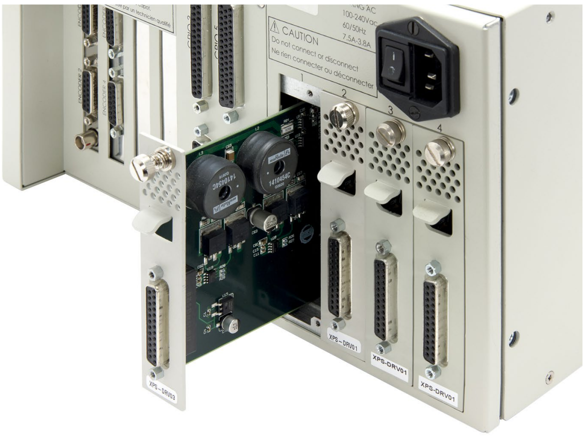


Figure 9: Axis driver card.

Please see the next section for installation instructions.

NOTE

Power Input : 100–240 V, 60/50 Hz, 7.5–3.8 A

2.6 Ethernet Configuration

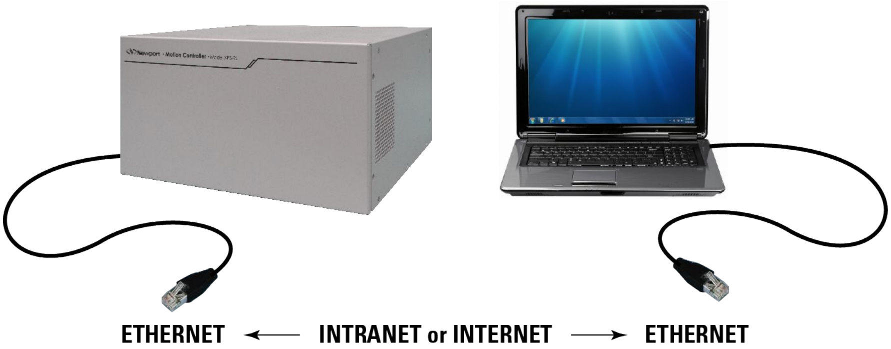


Figure 10: Ethernet configuration.

2.6.1 Communication Protocols

The Ethernet connection provides a local area network through which information is transferred in units known as packets. Communication protocols are necessary to dictate how these packets are sent and received. The XPS Controller/Driver supports the industry standard protocol TCP/IP.

TCP/IP is a “connection” protocol and in this protocol, the master must be connected to the slave in order to begin communication. Each packet sent is acknowledged when received. If no acknowledgment is received, the information is assumed lost and is resent.

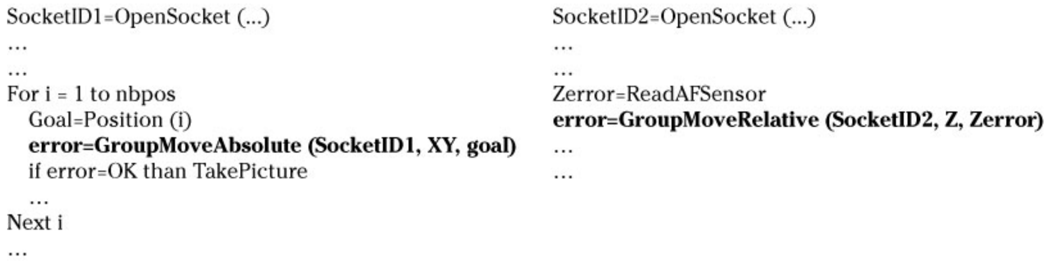
2.6.2 Addressing

There are two levels of addresses that define Ethernet devices. The first is the MAC address. This is a unique and permanent 6 byte number. No other device will have the same MAC address. The second level of addressing is the IP address. This is a 32-bit (or 4 byte) number. The IP address is constrained by each local network and must be assigned locally. Assigning an IP address to the controller can be done in a number of ways (see section 3.5: “Connecting to the XPS“).

2.7 Sockets, Multitasking and Multi-user Applications

Based on the TCP/IP Internet communication protocol, the XPS controller has a high number of virtual communication ports, known as sockets. To establish communication, the user must first request a socket ID from the XPS controller server (listening at a defined IP number and port number). When sending a function to a socket, the controller will always reply with a completion or error message to the socket that has requested the action.

The concept and application of sockets has many advantages. First, users can split their application into different segments that run independently on different threads or even on different computers. To illustrate this, see below:



```
SocketID1=OpenSocket (...)                            SocketID2=OpenSocket (...)
…                                                     …
…                                                     …
For i = 1 to nbpos                                    Zerror=ReadAFSensor
   Goal=Position (i)                                  error=GroupMoveRelative (SocketID2, Z, Zerror)
   error=GroupMoveAbsolute (SocketID1, XY, goal)      …
   if error=OK than TakePicture                       …
   …
Next i
…
```

In this example, a thread on socket 1 commands an XY stage to move to certain positions to take pictures while another thread on socket 2 independent of socket 1, concurrently manages an auto-focusing system. The second task could even be run on a different PC than the first task yet be simultaneously executed within the XPS. Alternatively, if the auto-focusing system is providing an analog feedback, this task could have been also implemented as a TCL script within the XPS (see the next topic).

Second, the concept of sockets has another practical advantage for many laboratory users since the use of threads allows them to share the same controller for different applications at the same time. With the XPS, it is possible that one group uses one axis of the XPS controller for an optical delay line, while another group simultaneously uses other axes for a totally different application. Both applications could run completely independent from different workstations without any delays or cross-talk.

The XPS controller uses TCP/IP blocking sockets, which means that the commands to the same socket are “blocked” until the XPS returns feedback about the completion of the currently executed command (either '0' if the command has been completed successfully, or an error code in case of an error). If customers want to run several processes in parallel, users should open as many 80 parallel sockets. Please refer to section 17.3: “Running Processes in Parallel” for further information about sockets and parallel processing.

2.8 Programming with TCL

TCL stands for Tool Command Language and is an open-source string based command language. With only a few fundamental constructs and relatively little syntax, it is very easy to learn, yet it can be as powerful and functional as traditional C language. TCL includes many different math expressions, control structures (if, for, foreach, switch, etc.), events, lists, arrays, time and date manipulation, subroutines, string manipulation, file management and much more. TCL is used worldwide with a user base approaching one million users. It is quickly becoming a standard and critical component in thousands of corporations. Consequently TCL is field proven, very well documented and has many tutorials, applications, tools and books publicly available (www.tcl.tk).

XPS users can use TCL to write complete application code and the XPS allows them to include any function in a TCL script. When developed, the TCL script can be executed in real time in the background of the motion controller processor and does not impact any processing requirements for servo updates or communication. The QNX hardware real time multiprocessing operating system used on the XPS controller assures precise management of the multiple processes with the highest reliability. Multiple TCL programs run in a time-sharing mode with the same priority and will get interrupted only by the servo, or communication tasks.

The advantage of executing application code within the controller over host run code is faster execution and better synchronization, in many cases without any time taken from the communication link. The complete communication link can be reserved for time critical process interaction from or to the process or host controller.

NOTE

It is important to note that the XPS gives communication requests priority over TCL script execution. When using TCL scripts for machine security or other time critical tasks, it is therefore important to limit the frequency of continuous communication requests from a host computer, which includes the XPS website, and to verify the execution speed of repetitive TCL scripts.

3.0 Getting Started

3.1 Unpacking and Handling

It is recommended that the XPS Controller/Driver be unpacked in your lab or work site rather than at the receiving dock. Unpack the system carefully; small parts and cables are included with the equipment. Inspect the box carefully for loose parts before disposing of the packaging. You are urged to save the packaging material in case you need to ship your equipment.

3.2 Inspection for Damage

XPS Controller/Driver has been carefully packaged at the factory to minimize the possibility of damage during shipping. Inspect the box for external signs of damage or mishandling. Inspect the contents for damage. If there is visible damage to the equipment upon receipt, inform the shipping company and Newport Corporation immediately.

WARNING

Do not attempt to operate this equipment if there is evidence of shipping damage or you suspect the unit is damaged. Damaged equipment may present additional personnel hazard. Contact Newport technical support for advice before attempting to plug in and operate damaged equipment.

3.3 Packing List

Included with each XPS controller are the following items:

- XPS-RL Quick Start.
- XPS controller.
- Cross-over cable, gray, 3 meters.
- Straight-through cable, black, 5 meters.
- Power cord.

If there are missing hardware or have questions about the hardware that were received, please contact Newport.

CAUTION

Before operating the XPS controller, please read chapter 1.0 very carefully.

3.4 System Setup

This section guides the user through the proper set-up of the motion control system. If not already done, carefully unpack and visually inspect the controller and stages for any damage. Place all components on a flat and clean surface.

CAUTION

No cables should be connected to the controller at this point!

First, the controller must be configured properly before stages can be connected.

3.4.1 Installing Driver cards

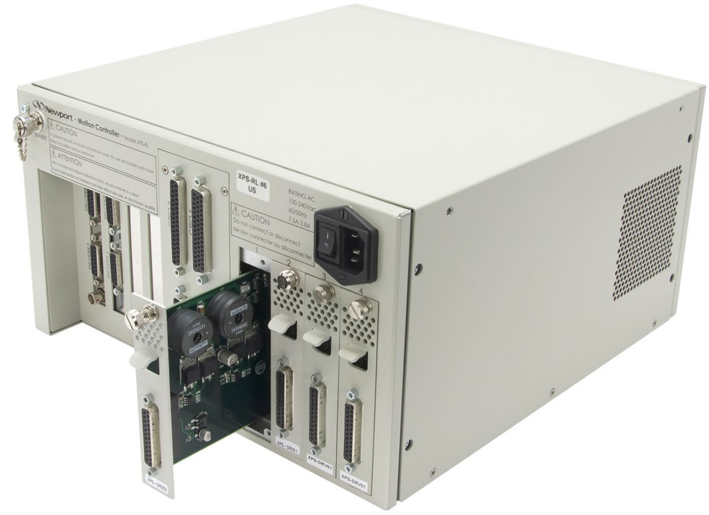


Figure 11: Installing driver cards.

Due to the high power of the XPS controller (180 W for the CPU and 300 W for the drives), ventilation is very important.

To ensure a good level of heat dissipation, the following rules must be followed:

1. It is strictly forbidden to use the XPS controller without the cover properly mounted on the chassis.
2. Driver boards must be inserted from left (driver 1) to right (driver 4) when looking at the rear of the controller.
3. If less than four are used, the remaining slots must be disabled with the appropriate slot covers that were delivered with the controller.
4. The surrounding ventilation holes at the sides and back of the XPS unit must be free from obstructions that prevent the free flow of air.

3.4.2 Power ON

- Plug the AC line cord supplied with the XPS into the AC power receptacle on the rear panel.
- Plug the AC line cord into the AC wall-outlet. Turn the Main Power Switch to ON (located on the Rear Panel).
- The system must be installed in such a way that power switch and power connector are accessible by the user.
- There is an initial beep after power on and a second beep when the controller has finished booting. If the controller boots properly, the second beep is happy-sounding, otherwise the sad-sounding beep is emitted. The time between the first and the second beeps can be 12–18 seconds.
- There is also an Inhibit switch with a BNC connector in the rear of the XPS. The Inhibit switch is directly linked by hardware to cut off motor power supply.

3.5 Connecting to the XPS

XPS-RL supports 10/100/1000 Mbps Ethernet networking:

1. Direct connection PC-to-XPS.

   The DHCP server active on the Ethernet plug identified "REMOTE" will automatically configure the connected computer to make it ready for communication with the XPS controller.

2. Network connection.

   The Ethernet plug identified "HOST" must be used to connect the XPS controller to a Network. Before connection, the controller IP setting must be set by the Network administrator.

Two cables are provided with the motion controller:

- Cross-over cable – used when connecting the XPS directly to a PC.
- Straight Ethernet cable – used when connecting the XPS through an intranet.

3.5.1 Straight through cables (black)

Standard Ethernet straight through cables are required when connecting the device to a standard network hub or switch.

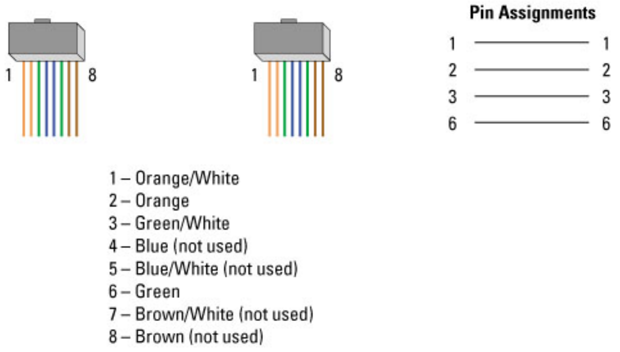


Figure 12: Straight through cables.

3.5.2 Cross-over cables (gray)

Standard Ethernet cross over cables are required when connecting the device directly to the Ethernet port of a PC.

NOTE

Cross over cables are typically labeled (cross over or XO) at one or both ends.

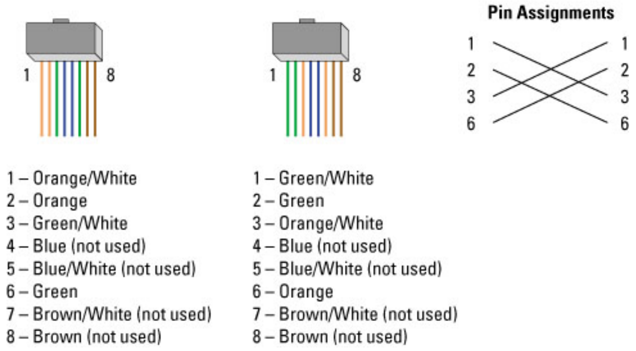


Figure 13: Ethernet cross over cables.

3.5.3 Direct Connection to the XPS Controller

For a direct connection between a PC and the XPS controller you need to use the crossover cable and the REMOTE connector at the back of the XPS.

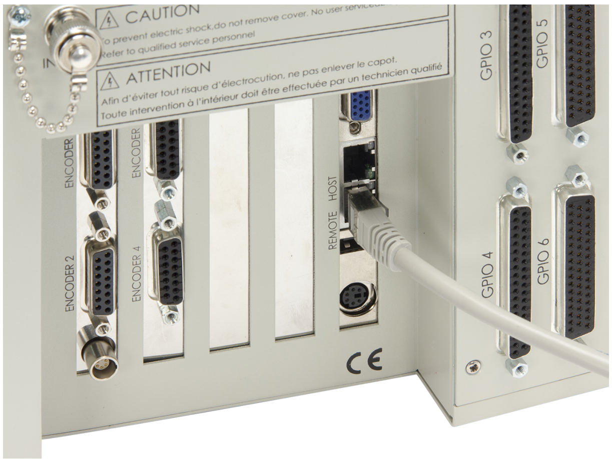
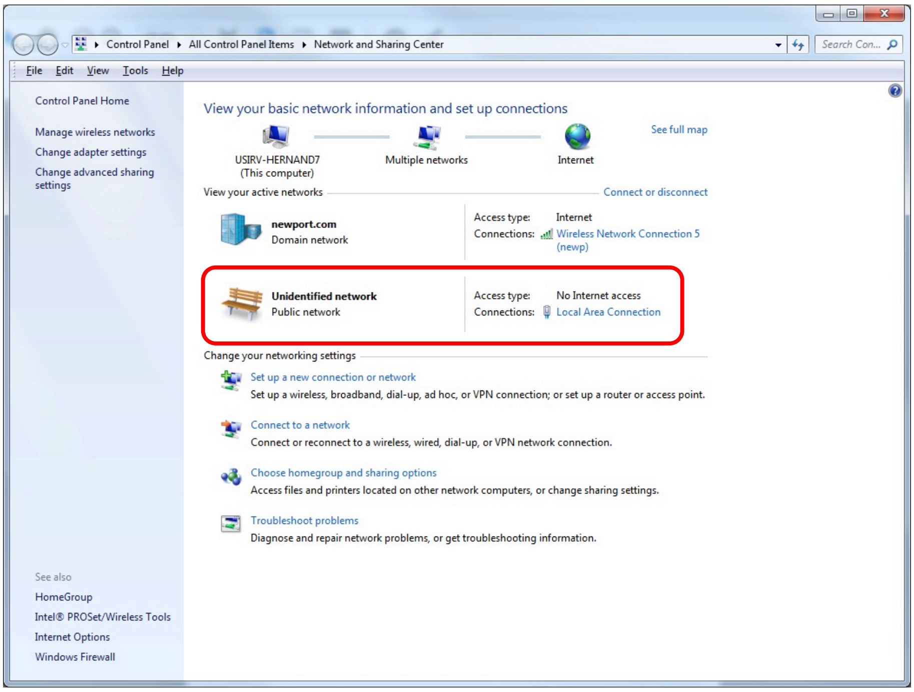

CAUTION
To prevent electric shock do not remove cover. No user service...
Refer to qualified service personnel
ATTENTION
Afin d'éviter tout risque d'électrocution, ne pas enlever le capot.
Toute intervention à l'intérieur doit être effectuée par un technicien qualifié
ENCODER
ENCODER 2
ENCODER
ENCODER 4
REMOTE
HOST
GPIO 3
GPIO 5
GPIO 4
GPIO 6
CE

Figure 14: Direct connection to the XPS using cross-over cable.

REMOTE Connection

The REMOTE plug has a DHCP server which automatically assigns an IP address on the PC's Ethernet card. Ensure the Local Area Connection is set to Obtain an IP address automatically. After connecting the REMOTE connector on the back of the XPS to the PC. Open an Internet Browser and connect to http://192.168.254.254.

When the PC is connected to the XPS, an Unidentified network will appear in your active networks found under Control Panel > Network and Sharing Center.


Control Panel ▸ All Control Panel Items ▸ Network and Sharing Center
Search Con...
File Edit View Tools Help
Control Panel Home
Manage wireless networks
Change adapter settings
Change advanced sharing settings

View your basic network information and set up connections

USIRV-HERNAND7 (This computer) — Multiple networks — Internet

View your active networks

newport.com
Domain network
Access type: Internet
Connections: Wireless Network Connection 5 (newp)

Unidentified network
Public network
Access type: No Internet access
Connections: Local Area Connection

Change your networking settings

Set up a new connection or network
Set up a wireless, broadband, dial-up, ad hoc, or VPN connection; or set up a router or access point.

Connect to a network
Connect or reconnect to a wireless, wired, dial-up, or VPN network connection.

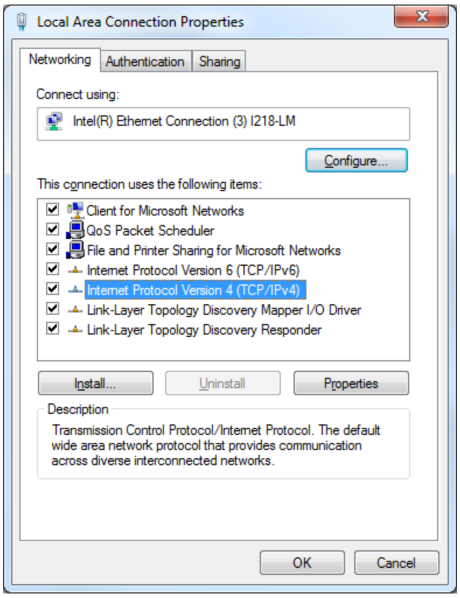
Choose homegroup and sharing options
Access files and printers located on other network computers, or change sharing settings.

Troubleshoot problems
Diagnose and repair network problems, or get troubleshooting information.

Following is the procedure to verify the Ethernet card address is set to Obtain an IP address automatically.

This procedure is for the Windows 7 operating system (almost similar process for Windows 8):

1. Start Button > Control Panel > Network and Sharing Center => Change adapter settings.
2. Right Click on Local Area Connection Icon and select Properties.



Local Area Connection Properties

Connect using:

Intel(R) Ethernet Connection (3) I218-LM

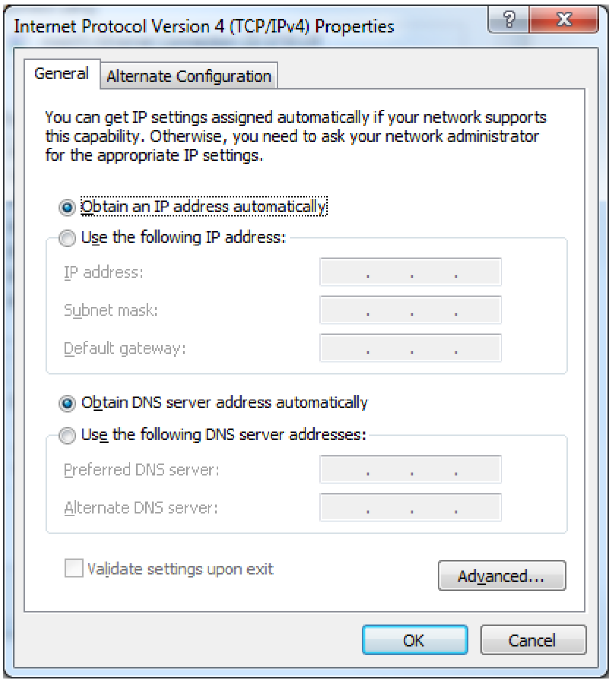
This connection uses the following items:

- ☑ Client for Microsoft Networks
- ☑ QoS Packet Scheduler
- ☑ File and Printer Sharing for Microsoft Networks
- ☑ Internet Protocol Version 6 (TCP/IPv6)
- ☑ Internet Protocol Version 4 (TCP/IPv4)
- ☑ Link-Layer Topology Discovery Mapper I/O Driver
- ☑ Link-Layer Topology Discovery Responder

Description

Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

3. Highlight Internet Protocol Version (TCP/IP, TCP/IP4) and click on Properties.



Internet Protocol Version 4 (TCP/IPv4) Properties

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

● Obtain an IP address automatically

○ Use the following IP address:

IP address: . . .

Subnet mask: . . .

Default gateway: . . .

● Obtain DNS server address automatically

○ Use the following DNS server addresses:

Preferred DNS server: . . .

Alternate DNS server: . . .

□ Validate settings upon exit

4. Verify Obtain an IP address automatically is selected and click "OK".

Following is the procedure for connecting to the controller:

5. Open an Internet Browser and connect to http://192.168.0.254 in case using HOST connector or connect to http://192.168.254.254 in case using REMOTE connector

Login:

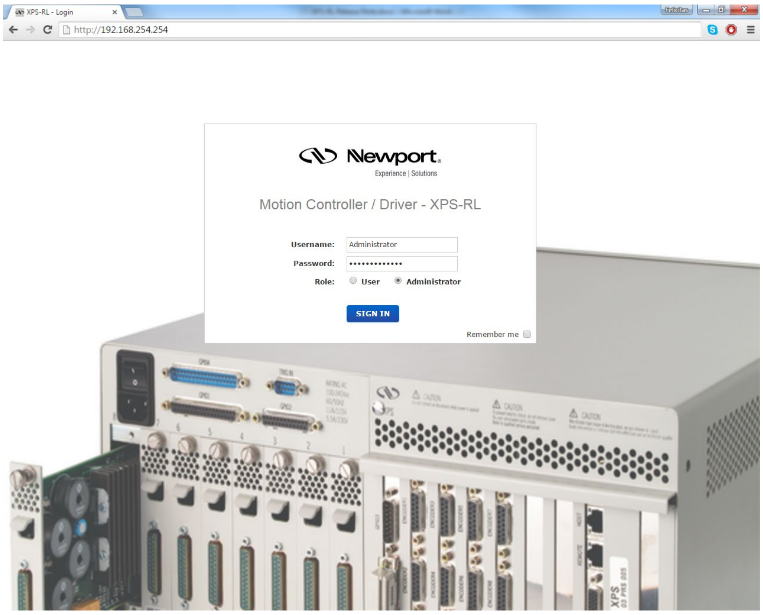
Name: **Administrator**

Password: **Administrator** (Please see the picture below).

Role: **Administrator**

NOTE

Please note that the login text is case sensitive.



XPS-RL - Login

http://192.168.254.254

Motion Controller / Driver - XPS-RL

Username: Administrator

Password: •••••••••••••

Role: ○ User ● Administrator

Once logged in, the XPS has established a direct connection to the local computer.

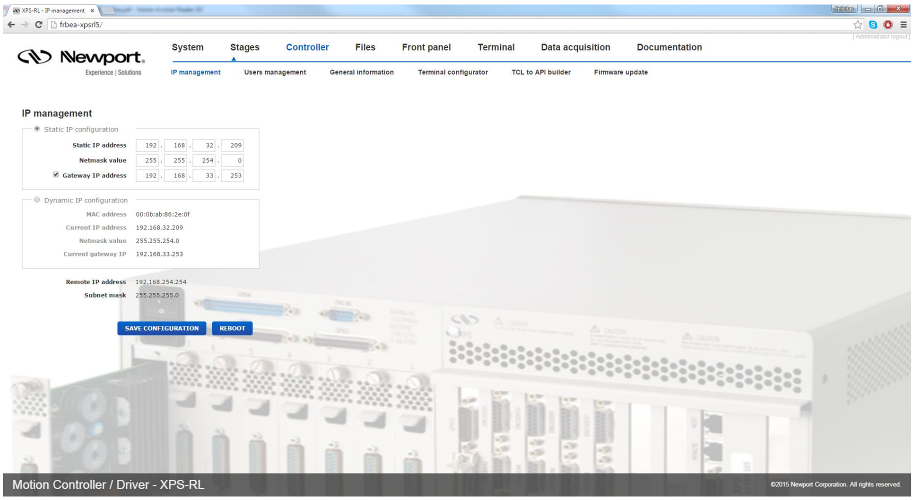
If you don't want to connect the XPS controller through a Corporate Network you may skip to section 3.7: "Connecting the Stages".

NOTE

If you want to change the IP address of the XPS controller, follow the explanation in the next section. It is necessary to keep using the gray cross-over Ethernet cable to connect the XPS controller directly to the PC.

3.5.4 Connecting the XPS to a Corporate Network using Static IP Configuration

Once you are logged in using the previously described steps for direct connection, you can change the IP configuration of the controller in order to connect the XPS over a Network. Select "CONTROLLER" of the web-site and select the sub-menu "IP Management".



IP management

● Static IP configuration

Static IP address 192 . 168 . 32 . 209
Netmask value 255 . 255 . 254 . 0
☑ Gateway IP address 192 . 168 . 33 . 253

○ Dynamic IP configuration

MAC address 00:0b:ab:86:2e:0f
Current IP address 192.168.32.209
Netmask value 255.255.254.0
Current gateway IP 192.168.33.253

Remote IP address 192.168.254.254
Subnet mask 255.255.255.0

The Static IP address, the Netmask value and the Gateway IP address must be provided by your Network Administrator to avoid network conflicts. Once you have these addresses, you can input them in the IP configuration window as shown above. The above shown addresses are only examples.

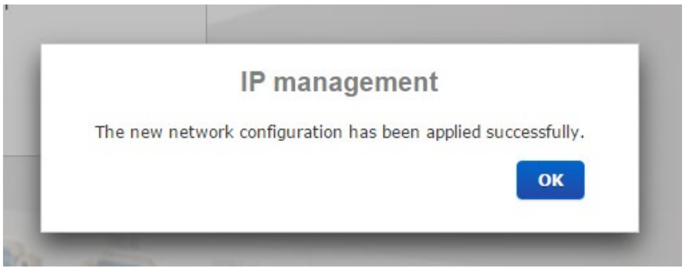
NOTE

To avoid conflict with the REMOTE Ethernet plug, the IP address must be different from 192.168.254.

NOTE

For the majority of Networks, the setting above for the Netmask value will work. However, for larger networks (200 computers or more), the Netmask value address must be verified with the IT department. In most cases and for larger networks, the Netmask value is set to 255.255.0.0.

Once the appropriate addresses for the Static IP configuration are set, click on "SAVE CONFIGURATION" and the following screen appears:



IP management

The new network configuration has been applied successfully.

Click "OK" and reboot the controller by clicking REBOOT.

A pop-up windows appears showing the "REBOOT IN PROGRESS". When the boot sequence is complete, the user is redirected to the login page. The time to reboot is about 50 seconds.

Connect the CAT-5 network cable (black) to the HOST connector of the XPS controller and to your network.

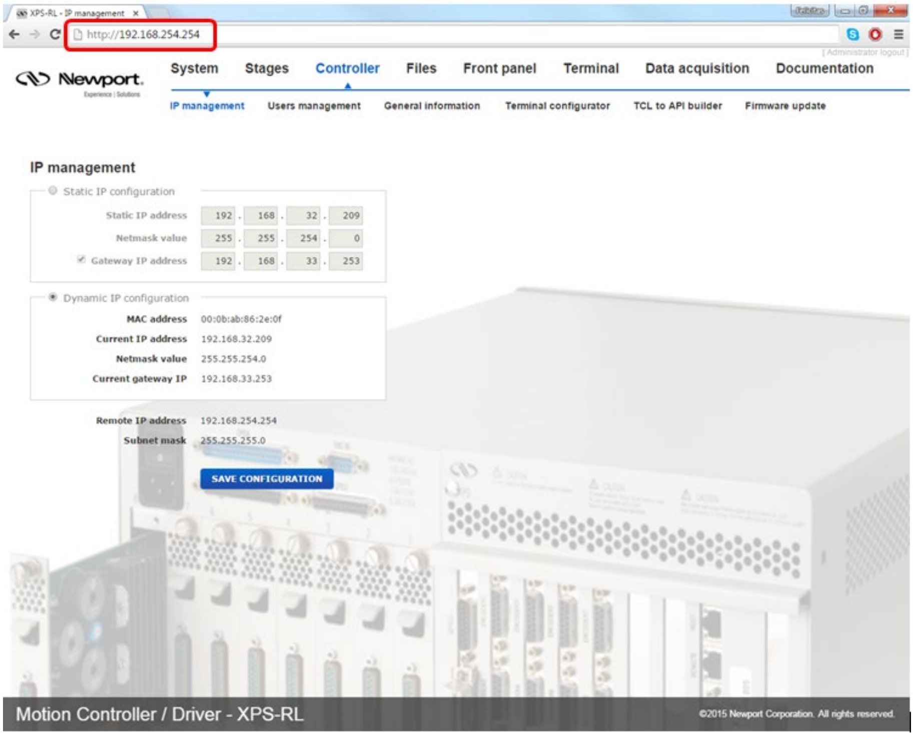
After restarting the controller, open the Internet browser and connect using your given Static IP address.

If you don't want to connect directly to the Corporate Network using the Dynamic IP Configuration, skip to section 3.7: "Connecting the Stages".

3.5.5 Configuring the XPS for Connection to a Corporate Network Using Dynamic IP Configuration

It is recommended to ask your IT department to configure the XPS to your network to avoid any issue with your network policies and rules.

- Connect to the XPS as described in section 3.5.3: "Direct Connection to the XPS Controller"
- Connect the HOST plug to your network using a direct cable.
- Get to **Controller → IP management** web page
- Select dynamic IP as shown below:



IP management

○ Static IP configuration

Static IP address 192 . 168 . 32 . 209

Netmask value 255 . 255 . 254 . 0

☑ Gateway IP address 192 . 168 . 33 . 253

● Dynamic IP configuration

MAC address 00:0b:ab:86:2e:0f

Current IP address 192.168.32.209

Netmask value 255.255.254.0

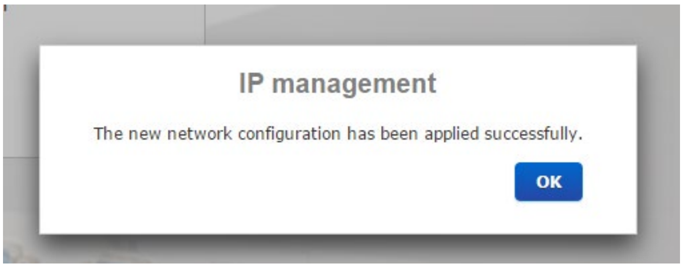
Current gateway IP 192.168.33.253

Remote IP address 192.168.254.254

Subnet mask 255.255.255.0

SAVE CONFIGURATION
Motion Controller / Driver - XPS-RL
©2015 Newport Corporation. All rights reserved.

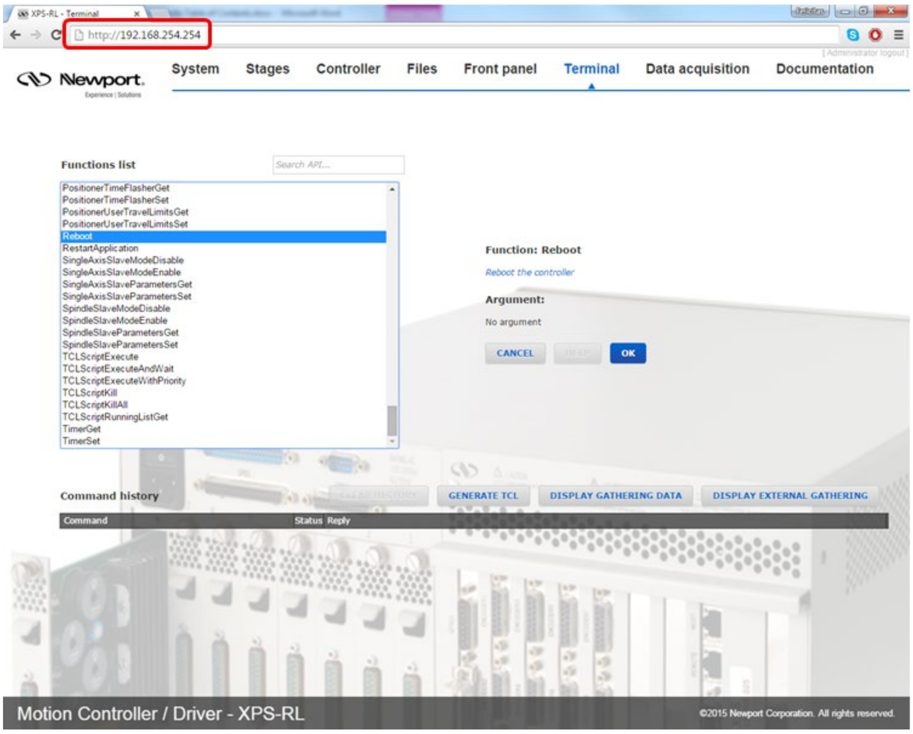
Click the "SAVE CONFIGURATION" button and the following screen appears:



IP management

The new network configuration has been applied successfully.

Go to the TERMINAL window, double click on the Reboot function, then press the Execute button:



Functions list

PositionerTimeFlasherGet
PositionerTimeFlasherSet
PositionerUserTravelLimitsGet
PositionerUserTravelLimitsSet
Reboot
RestartApplication
SingleAxisSlaveModeDisable
SingleAxisSlaveModeEnable
SingleAxisSlaveParametersGet
SingleAxisSlaveParametersSet
SpindleSlaveModeDisable
SpindleSlaveModeEnable
SpindleSlaveParametersGet
SpindleSlaveParametersSet
TCLScriptExecute
TCLScriptExecuteAndWait
TCLScriptExecuteWithPriority
TCLScriptKill
TCLScriptKillAll
TCLScriptRunningListGet
TimerGet
TimerSet

Function: Reboot

Reboot the controller

Argument:

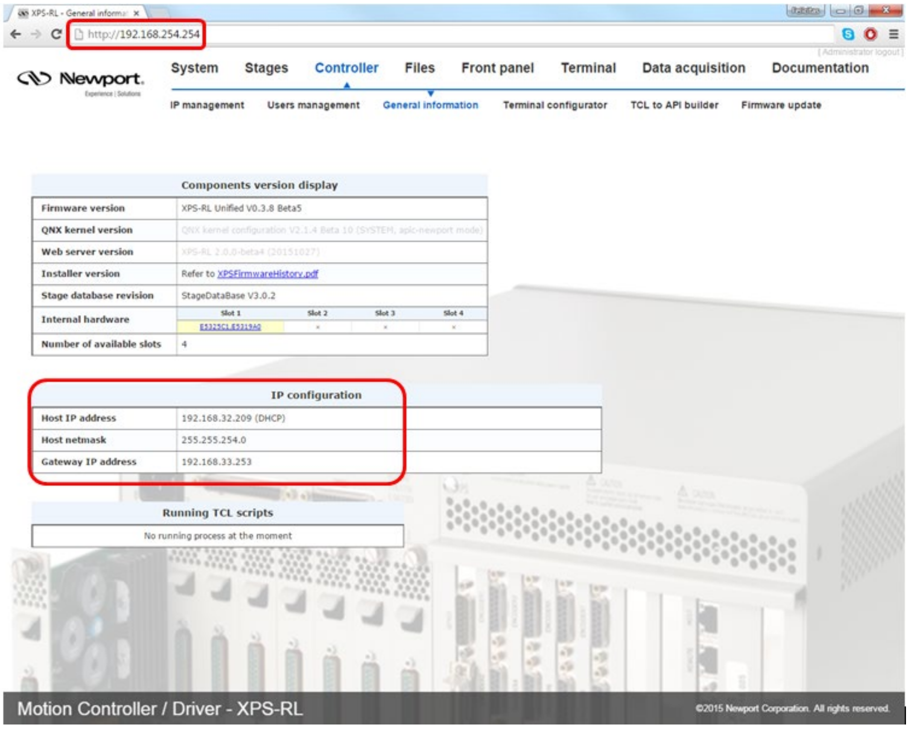
No argument

Command history

Command	Status	Reply

Wait for controller to reboot, open the internet browser and connect to REMOTE

You can see the dynamic IP address in **Controller → General**.



Components version display

Firmware version	XPS-RL Unified V0.3.8 Beta5			
QNX kernel version	QNX kernel configuration V2.1.4 Beta 10 (SYSTEM, apic-newport mode)			
Web server version	XPS-RL 2.0.0-beta4 (20151027)			
Installer version	Refer to XPSFirmwareHistory.pdf			
Stage database revision	StageDataBase V3.0.2			
Internal hardware	Slot 1	Slot 2	Slot 3	Slot 4
	E5325C1,E5319A0	×	×	×
Number of available slots	4			

IP configuration

Host IP address	192.168.32.209 (DHCP)
Host netmask	255.255.254.0
Gateway IP address	192.168.33.253

Running TCL scripts

No running process at the moment

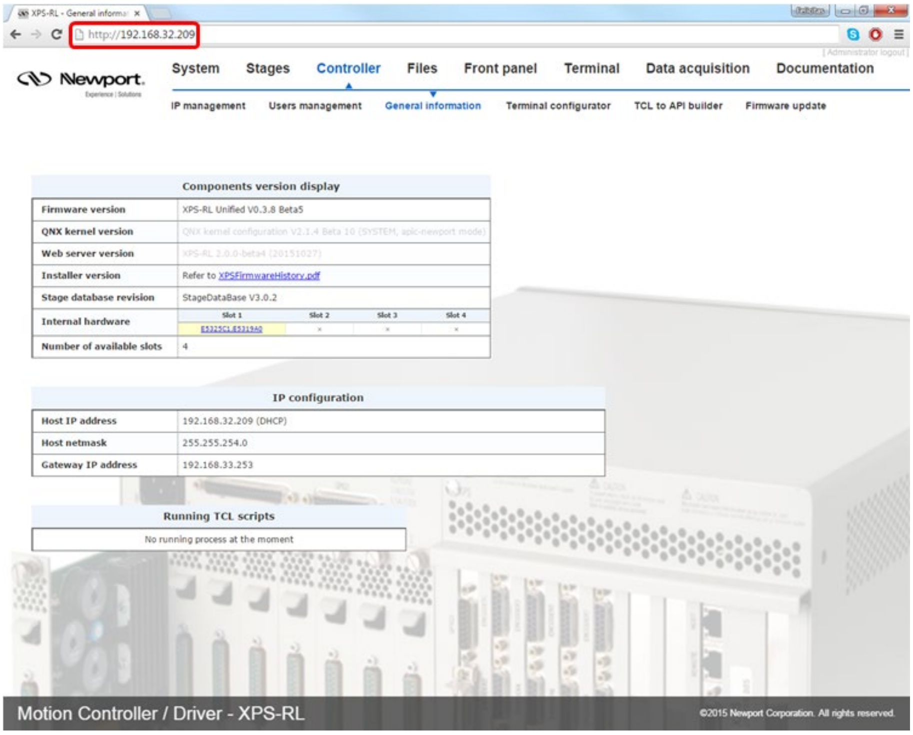
The IP address delivered by your DHCP is displayed above.

In case the XPS cannot negotiate an IP address from the DHCP the displayed address will be 0.0.0.0. In that case contact your IT department.

Remove the REMOTE cable and, if needed, configure your PC back to its original Ethernet configuration, you have saved before modification.

Make sure that the standard CAT-5 network cable (black) is connected to the HOST connector of the XPS controller and to your network.

Open your internet browser and use the dynamic IP address.



XPS-RL - General informa ×

http://192.168.32.209

Components version display				
Firmware version	XPS-RL Unified V0.3.8 Beta5			
QNX kernel version	QNX kernel configuration V2.1.4 Beta 10 (SYSTEM, apic-newport mode)			
Web server version	XPS-RL 2.0.0-beta4 (20151027)			
Installer version	Refer to XPSFirmwareHistory.pdf			
Stage database revision	StageDataBase V3.0.2			
Internal hardware	Slot 1	Slot 2	Slot 3	Slot 4
	E5325CL.E5319A0	×	×	×
Number of available slots	4			

IP configuration	
Host IP address	192.168.32.209 (DHCP)
Host netmask	255.255.254.0
Gateway IP address	192.168.33.253

Running TCL scripts
No running process at the moment

Motion Controller / Driver - XPS-RL
©2015 Newport Corporation. All rights reserved.

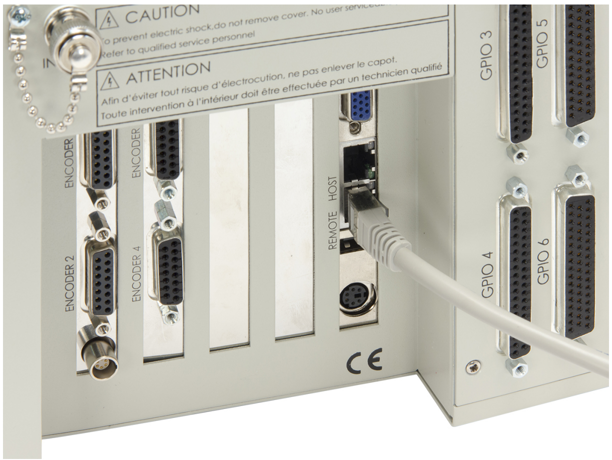
Check with your IT department that the lease time set at the DHCP is longer than the time you plan to leave the XPS switched off otherwise you will lose your dynamic address and will need to connect to the REMOTE to know the new assigned one by the DHCP.

NOTE

Do not use Dynamic IP configuration if your DHCP server uses Windows NT 4.0.

3.5.6 Recovering a lost IP configuration

If you want to recover a lost IP configuration, you need to connect the PC directly to the REMOTE connector at the back of the XPS with the gray cross-over cable.



CAUTION
To prevent electric shock,do not remove cover. No user service...
Refer to qualified service personnel
ATTENTION
Afin d'éviter tout risque d'électrocution, ne pas enlever le capot.
Toute intervention à l'intérieur doit être effectuée par un technicien qualifié
IN
ENCODER
ENCODER
ENCODER 2
ENCODER 4
REMOTE
HOST
GPIO 3
GPIO 5
GPIO 4
GPIO 6
CE

Figure 15: Direct connection to the XPS using a cross-over cable and the REMOTE connector.

First, the IP address on the PC's Ethernet card must be set to Obtain IP Address Automatically.

1. Open Internet Browser and connect to **http://192.168.254.254**

Login:

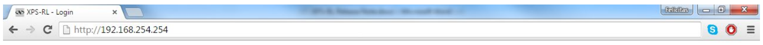
Name: **Administrator**

Password: **Administrator** (Please see the picture below).

Rights: **Administrator**

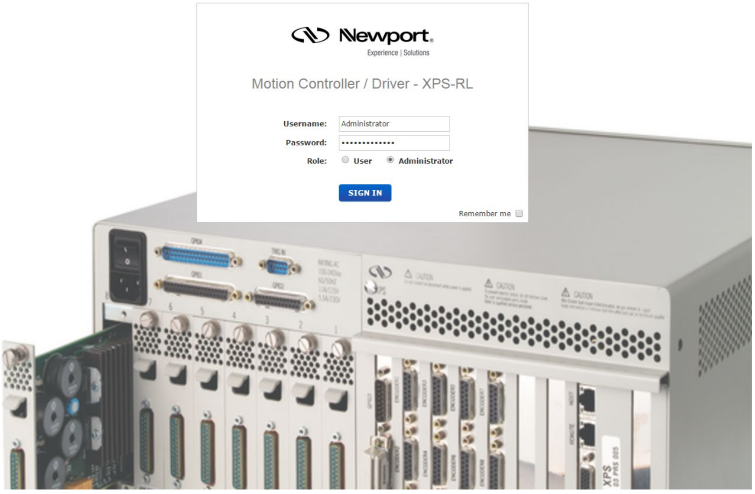
NOTE

Please note that the login text is case sensitive.



XPS-RL - Login

http://192.168.254.254



Newport. Experience | Solutions

Motion Controller / Driver - XPS-RL

Username: Administrator

Password: •••••••••••••

Role: ○ User ● Administrator

SIGN IN

Remember me □

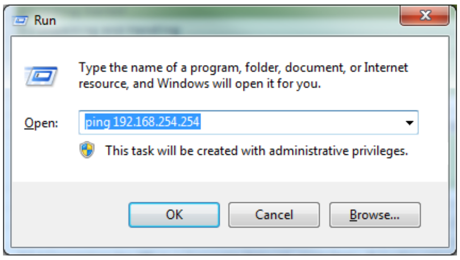
Once you are logged in, you can change the IP configuration by following the steps described in section 0 or 3.5.5 depending on your configuration.

NOTE

If you want to reset the IP address to the default factory setting, follow the section 3.5.4: "Connecting the XPS to a Corporate Network using Static IP Configuration" to set the IP address back to 192.168.0.254.

3.6 Testing your XPS-PC Connection and Communication

To check if the XPS communicates with to the host computer, send a ping message from the computer to the XPS. This is done through the Windows menu: Start->Run->, then type: ping + IP address of the XPS. See the example below for the IP address 192.168.254.254:



Run

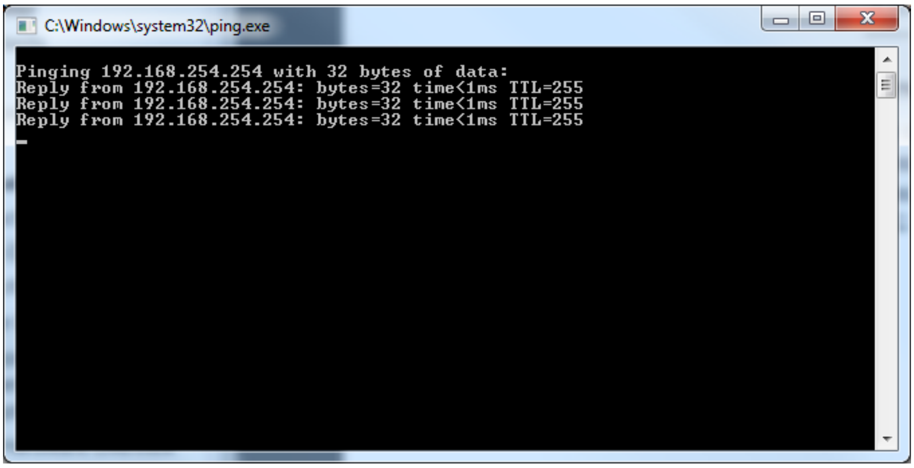
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: ping 192.168.254.254

This task will be created with administrative privileges.

OK Cancel Browse...

If the XPS is connected and communicates properly, it replies in the terminal window that appears after clicking on the OK button:



```
Pinging 192.168.254.254 with 32 bytes of data:
Reply from 192.168.254.254: bytes=32 time<1ms TTL=255
Reply from 192.168.254.254: bytes=32 time<1ms TTL=255
Reply from 192.168.254.254: bytes=32 time<1ms TTL=255
```

If the XPS controller is not communicating, the window displays that the time delay of the request is exceeded. Ensure that the correct cable and IP addresses are set properly.

3.7 Connecting the Stages

CAUTION

Never connect/disconnect stages while the XPS controller is powered on.

CAUTION

Mount the stage(s) on a flat, stable surface before connecting to the XPS controller.

With the power off, carefully connect the supplied cables to the stage and to the appropriate axis connector at the rear of the controller. Secure both connections with the locking thumbscrews.

When using stages with an analog encoder interface, a separate encoder cable must be connected to the corresponding axis connector of the control board labeled "Encoder 1" to "Encoder 4".

Please note that the XPS controller will not detect cross-connection errors between the motor of one stage and the encoder of another stage. Make sure that motor, encoder and other cables are plugged to the appropriate axis driver card and encoder connectors.

CAUTION

It is strongly recommended that the user read section 3.4: "System Setup" before attempting to turn the controller on. Serious damage could occur if the system is not properly configured.

All Newport ESP-compatible stages are electrically and physically compatible with the XPS controller. ESP-compatible stages are visually identified with a blue "ESP Compatible" sticker on the stage. If an ESP-compatible motion system was purchased, all necessary hardware to connect the stage with the XPS controller is included. The stage connects to the XPS via a shielded custom cable that carries all the power and control signals (encoder, limits, and home signals). The cable is terminated with a standard 25-pin D-Sub connector.

Dummy Stages

"Dummy stages" can be used to simulate a stage. This feature allows users to configure and test the system's behavior without having real stages or driver card connected.

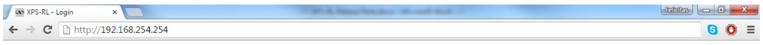
To configure your system with a number of dummy stages use Manual configuration. For more information about Manual configuration see section 4.13: “System – Manual Configuration”.

Dummy stage configuration file can be found in the StageDataBase.txt file with name [DUMMY@DUMMY_STAGE@NO_DRIVER].

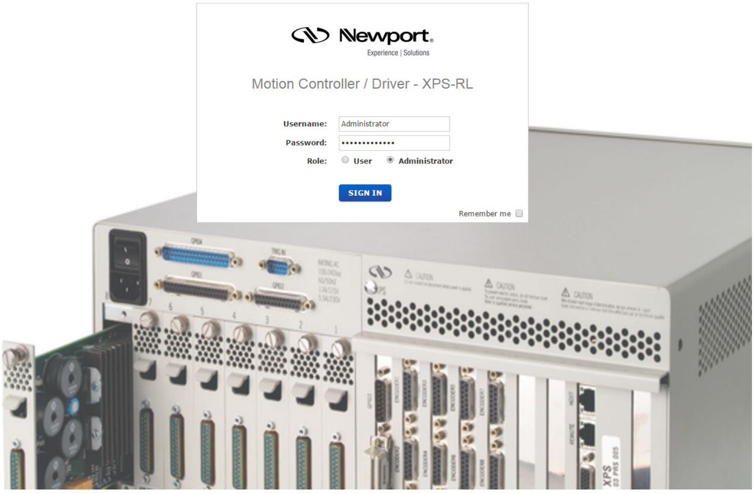
3.8 Configuring the Controller

When the driver boards are installed and the IP address is configured, the controller can be configured for the stages:

- Switch off the XPS controller.
- Connect the stages or motion devices.
- Switch on the XPS controller and wait for the end of the boot sequence (approximately 50 seconds).
- Open an internet browser and connect to http://<your fixed IP address>


XPS-RL - Login

http://192.168.254.254


Motion Controller / Driver - XPS-RL

Username: Administrator

Password: •••••••••••••

Role: ○ User ● Administrator

SIGN IN

Remember me □

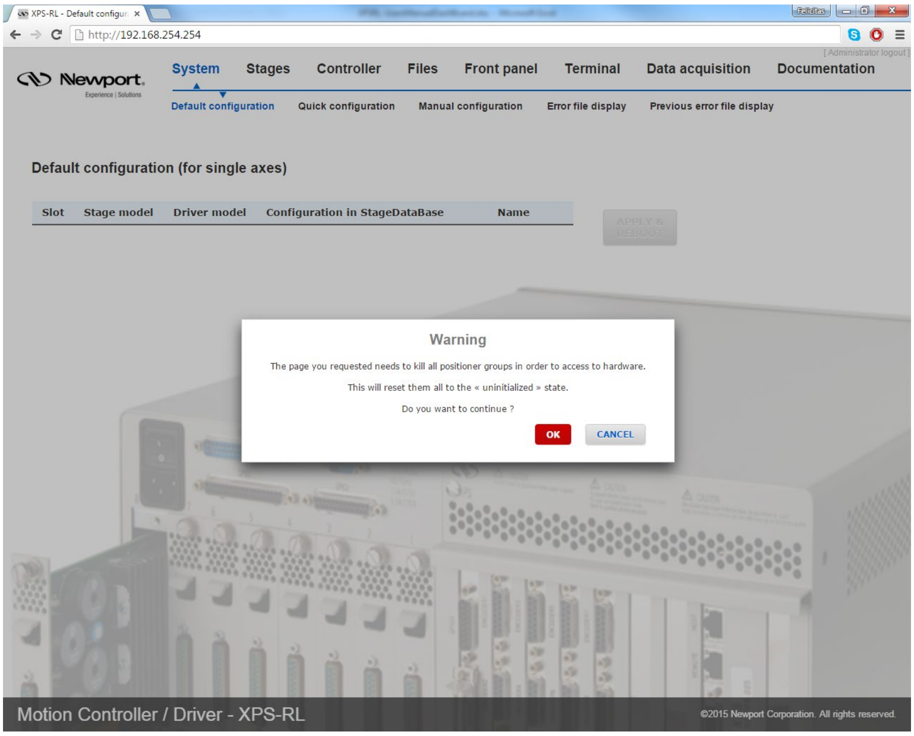
Login:	**Administrator**
Password:	**Administrator**
Rights:	**Administrator**

There are three possibilities to configure the controller: Default configuration, quick configuration and manual configuration. Default configuration is the simplest method to configure the controller, but has some limitations:

- Default configuration works only with Newport ESP compatible positioners.
- Default configuration configures all detected positioners as single axis groups. However, single axis groups provide limited functionality (no synchronized motion, no trajectories, no XY or XYZ compensation). To take full benefit of the capabilities of the XPS controller, a manual configuration is needed.
- For non-Newport stages or very old Newport stages, manual configuration is required. See chapter 23.0: “Appendix F: Configuration Wizard”.
- Manual configuration is also required for some vacuum compatible stages (no ESP chip) and for stages with adjustable home position (-1, 0, +1), if the home position is changed from the standard position 0 to -1 or +1. The positions +1 and -1 require different settings in the stage data base, as the home switch position is not recognized by the ESP chip.

3.8.1 Default Configuration

When logged in as Administrator, select **SYSTEM**, then **“Default configuration”**. The following screen appears:



Default configuration (for single axes)

Slot	Stage model	Driver model	Configuration in StageDataBase	Name

Warning

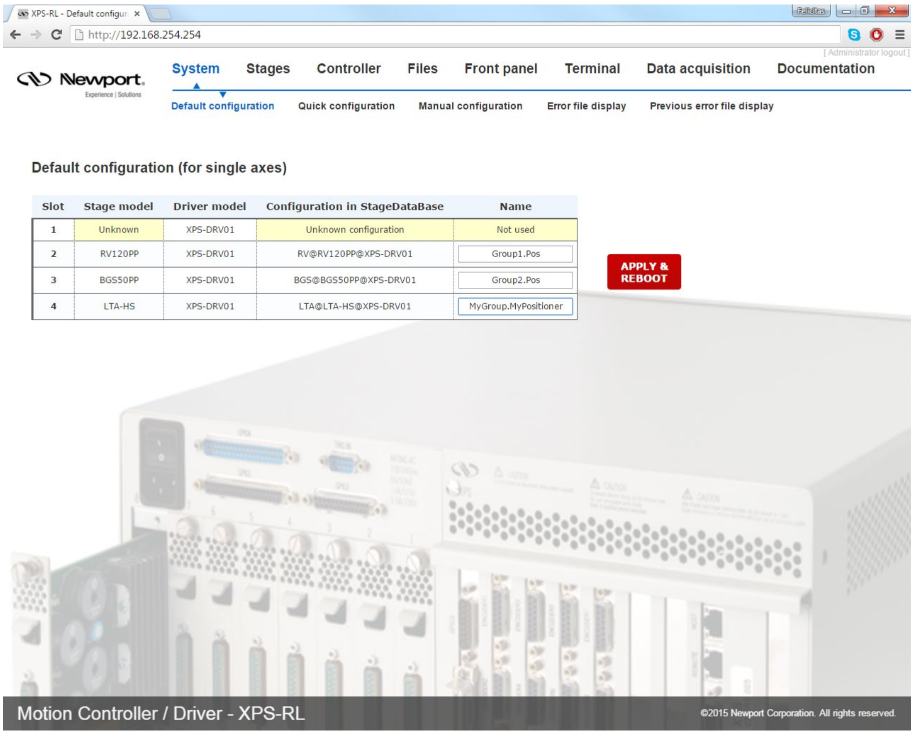
The page you requested needs to kill all positioner groups in order to access to hardware.

This will reset them all to the « uninitialized » state.

Do you want to continue ?

OK CANCEL

If you want to continue, click the "OK" button and the following page appears:

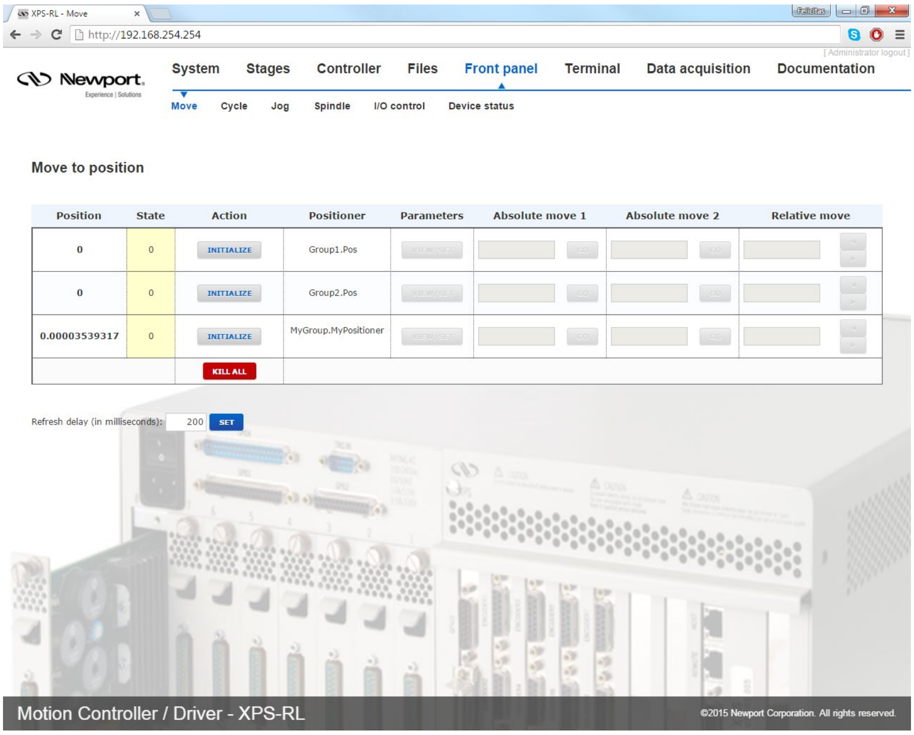


Default configuration (for single axes)

Slot	Stage model	Driver model	Configuration in StageDataBase	Name
1	Unknown	XPS-DRV01	Unknown configuration	Not used
2	RV120PP	XPS-DRV01	RV@RV120PP@XPS-DRV01	Group1.Pos
3	BGS50PP	XPS-DRV01	BGS@BGS50PP@XPS-DRV01	Group2.Pos
4	LTA-HS	XPS-DRV01	LTA@LTA-HS@XPS-DRV01	MyGroup.MyPositioner

Check, if all connected stages are recognized by the system. To change the Group name type in desired name under the Name column. If all connected stages are recognized, click “APPLY & BOOT”.

When the controller has finished booting login, select **FRONT PANEL**, and then select **“Move”**. The following screen appears:



Move to position

Position	State	Action	Positioner	Parameters	Absolute move 1	Absolute move 2	Relative move
0	0	INITIALIZE	Group1.Pos	VIEW/SET	GO	GO	
0	0	INITIALIZE	Group2.Pos	VIEW/SET	GO	GO	
0.00003539317	0	INITIALIZE	MyGroup.MyPositioner	VIEW/SET	GO	GO	
		KILL ALL					

Refresh delay (in milliseconds): 200 SET

Click “Initialize”. The State number changes from 0 to 42 and the Action button changes from “Initialize” to “Home”. Click “Home”. The stage starts moving to find its reference position. When done, the state number is 11 and the action button changes to disable. Enter an allowed position value in the “Abs move 1” field and click “Go”. The stage moves to this absolute position.

Your system is now ready to use. For more advanced functions, please read the rest of this manual.

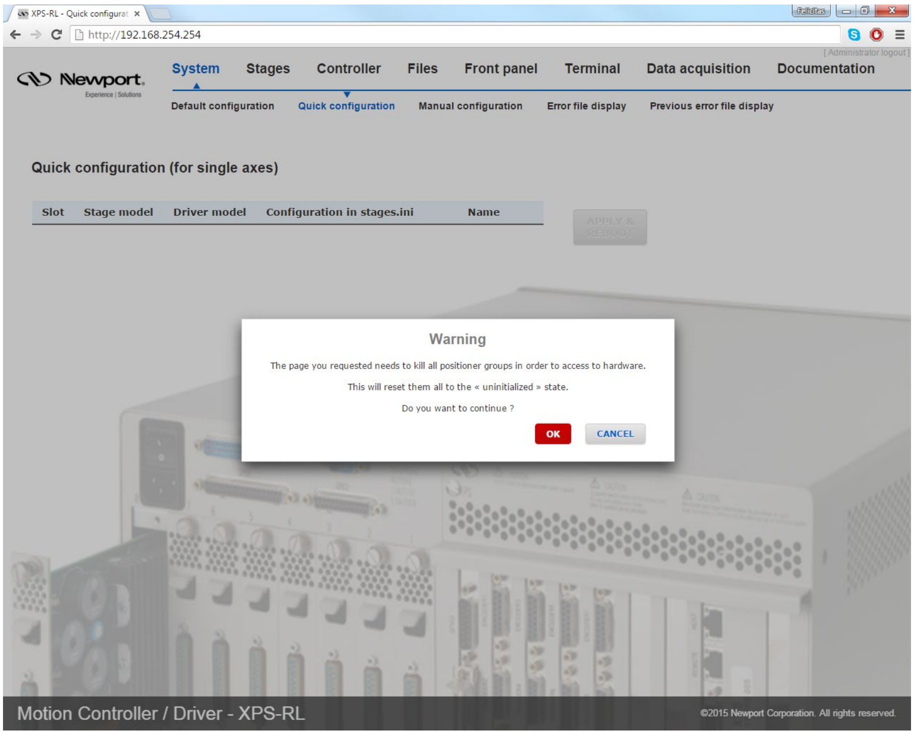
NOTE

In “DEFAULT-CONFIGURATION” the default group type is set as SingleAxis. To set the positioners to a different group type, use manual configuration.

3.8.2 Quick Configuration for Newport Positioners

Before using Quick Configuration, it is needed to populate the stages.ini file with the needed configurations, using **Add, remove or edit** stages under the main tab **Stages** (see section 3.8.3).

When logged in as Administrator, select **SYSTEM**, then **Quick configuration**. The following screen appears:



Quick configuration (for single axes)

Slot	Stage model	Driver model	Configuration in stages.ini	Name

Warning

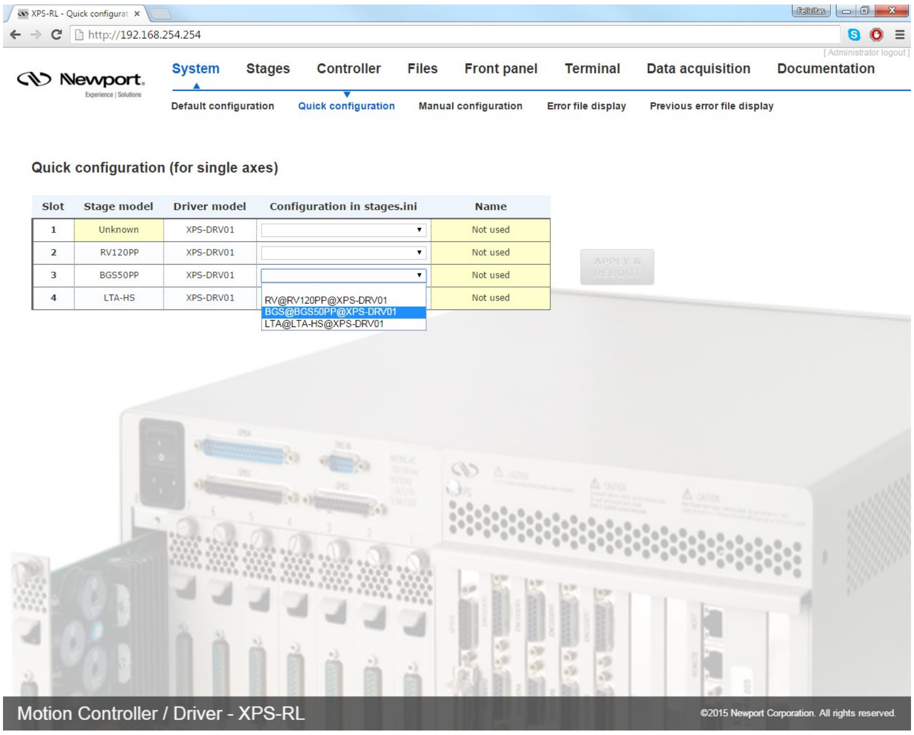
The page you requested needs to kill all positioner groups in order to access to hardware.

This will reset them all to the « uninitialized » state.

Do you want to continue ?

OK CANCEL

If you want to continue, click the "OK" button and the following page appears:



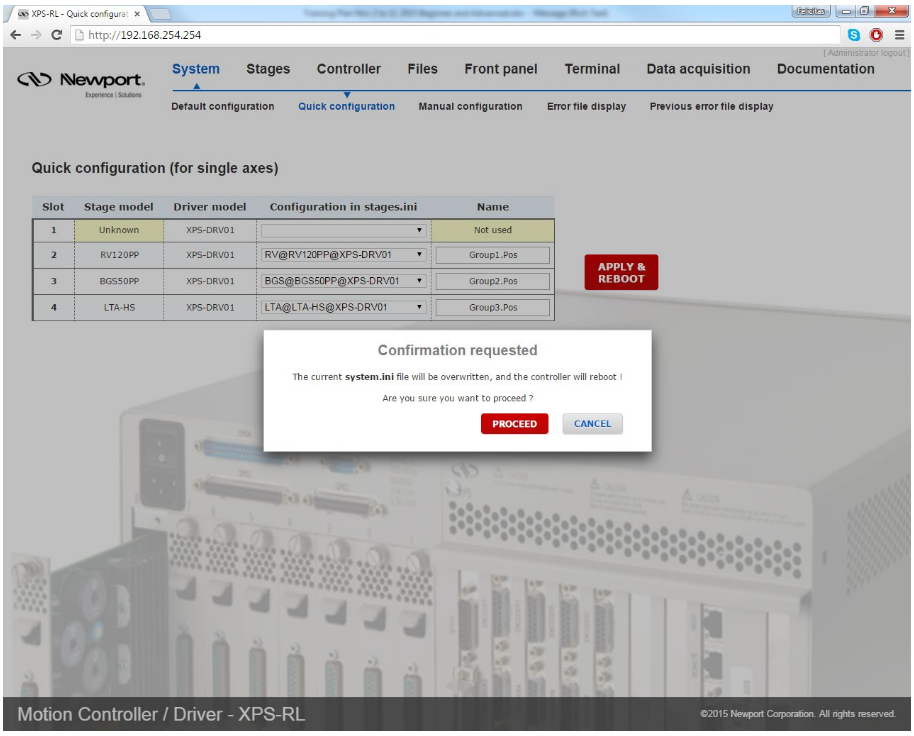
Quick configuration (for single axes)

Slot	Stage model	Driver model	Configuration in stages.ini	Name
1	Unknown	XPS-DRV01		Not used
2	RV120PP	XPS-DRV01		Not used
3	BGS50PP	XPS-DRV01		Not used
4	LTA-HS	XPS-DRV01		Not used

RV@RV120PP@XPS-DRV01
BGS@BGS50PP@XPS-DRV01
LTA@LTA-HS@XPS-DRV01

Check, if all connected stages are recognized by the system under Stage model. Use the drop down menu to select the stage configuration for the selected axis. The drop down menu lists stage configuration(s) available from the stages.ini file stored on the XPS controller. To add stage parameters to the stages.ini file click on **Stages** then click **Add, remove or edit stages**.

If all stages are recognized and after selecting the stage parameters, click **"APPLY & BOOT"** and the following page appears:



Quick configuration (for single axes)

Slot	Stage model	Driver model	Configuration in stages.ini	Name
1	Unknown	XPS-DRV01		Not used
2	RV120PP	XPS-DRV01	RV@RV120PP@XPS-DRV01	Group1.Pos
3	BGS50PP	XPS-DRV01	BGS@BGS50PP@XPS-DRV01	Group2.Pos
4	LTA-HS	XPS-DRV01	LTA@LTA-HS@XPS-DRV01	Group3.Pos

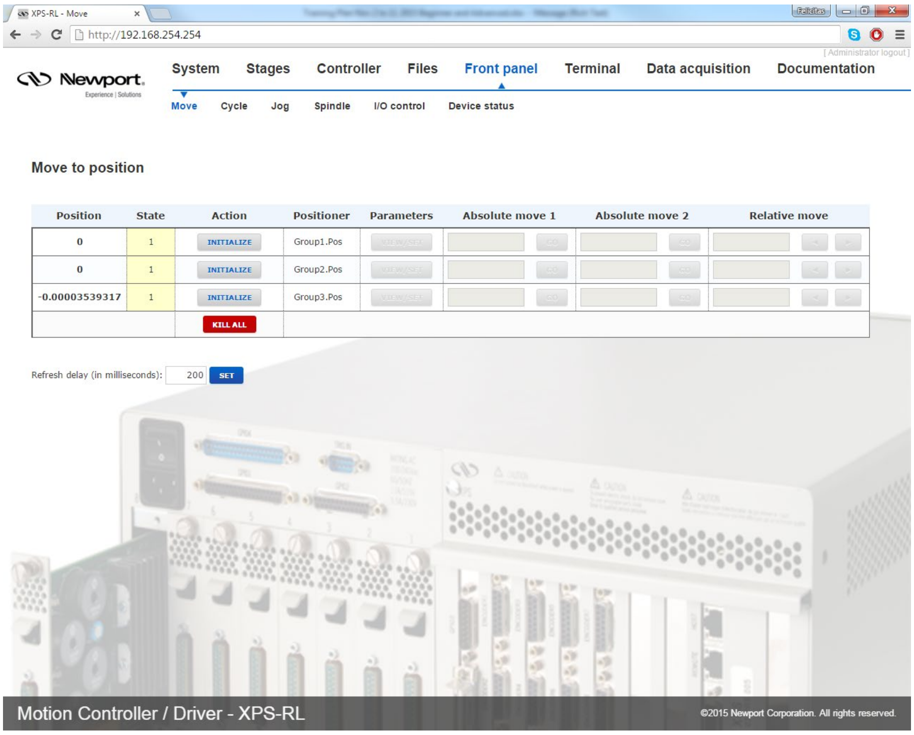
Confirmation requested

The current **system.ini** file will be overwritten, and the controller will reboot !

Are you sure you want to proceed ?

To configure the XPS click "PROCEED." The controller reboots and the Login screen appears (this may take up to 16 seconds).

When the controller has finished booting (a second beep after 12-18 seconds), Login, select **FRONT PANEL**, and then select **Move.** The following screen appears:



Move to position

Position	State	Action	Positioner	Parameters	Absolute move 1	Absolute move 2	Relative move
0	1	INITIALIZE	Group1.Pos	VIEW/SET	GO	GO	< >
0	1	INITIALIZE	Group2.Pos	VIEW/SET	GO	GO	< >
-0.00003539317	1	INITIALIZE	Group3.Pos	VIEW/SET	GO	GO	< >
		KILL ALL					

Refresh delay (in milliseconds): 200 SET

Motion Controller / Driver - XPS-RL
©2015 Newport Corporation. All rights reserved.

Click "INITIALIZE". The State number changes from 0 to 42 and the Action button changes from "INITIALIZE" to "HOME". Click "HOME". The stage starts moving to find its reference position. When done, the state number is 11 and the action button changes to disable. Enter an allowed position value in the "Absolute move 1" field and click "GO." The stage moves to this absolute position.

Your system is now ready to use. For more advanced functions, please read the rest of this manual.

NOTE

In "Quick-configuration" the default group type is SingleAxis. To set the positioners to a different group type, use "Manual configuration".

3.8.3 Manual Configuration for Newport Positioners

Manual configuration provides users access to all capabilities of the XPS controller.

For manual configuration, users first need to build the stage data base using the web tool "**Add, remove or edit stages**" under the main tab **Stages**. When adding a new stage from this web tool, the controller copies the parameters from its internal database (which contains parameters for all Newport stages) and stores these parameters in a file called stages.ini. Hence, the stages.ini file contains the parameters for only a subset of stages as defined by the user. Users can assign any name for their stages. The default name is the Newport part number, but in some cases it makes sense to use a different name. This way, for instance, it is possible to add the same set of parameters several times in the stage data base under different stage names. Later, you can modify certain parameters, like travel ranges or PID settings, to optimize the stage for different applications.

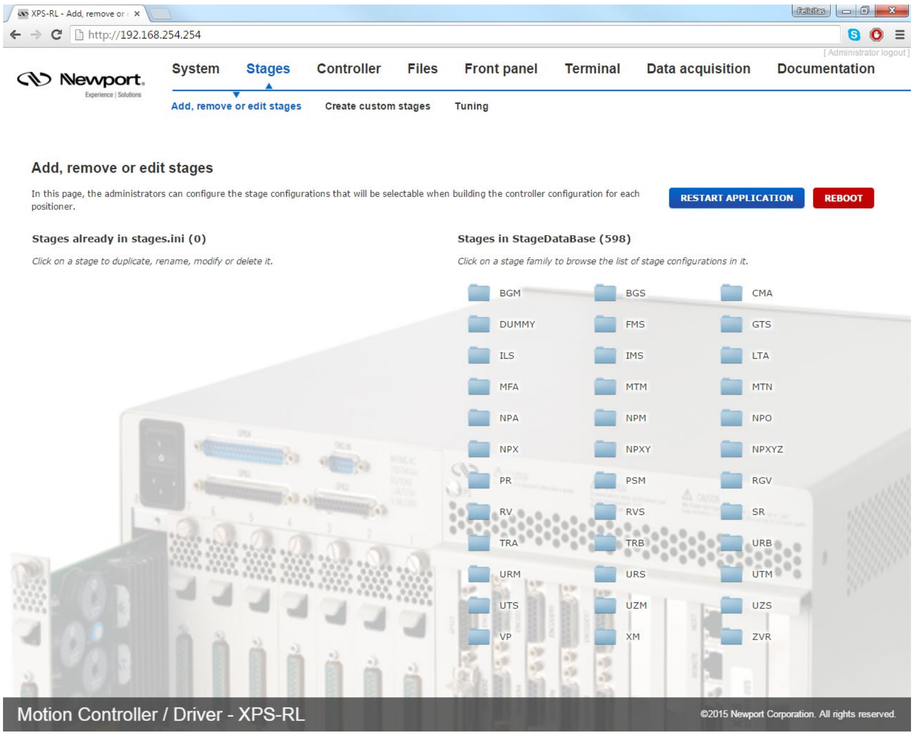
All stage parameters can be modified using the Web Tool **"Add, remove or edit stages"** under the main tab **STAGE**. Click on a stage to duplicate, rename, modify or delete it. Another Web Tool for modifying stage parameter can be found under Files → Configuration files using the text editor (see section 4.32 for details). Alternatively, the stage parameters can be modified directly in the stages.ini file using a text editor. The stages.ini file is located in the Config folder of the XPS controller. This folder is accessible via ftp, see section 4.32 for details.

When all stages are added to the stages.ini file, build the system using the web tool "**Manual Configuration**" under the main tab **SYSTEM**. In this tool, the stages get assigned to positioners and the positioners get assigned to motion groups. Please refer to section 6.3 for details on the different motion groups and their specific features. The group name and positioner name can be any user given name. Once the system has been built, all system information is stored in a file called system.ini. Also, the system.ini file is located in the Config folder of the XPS controllerand can be viewed or edited from the Web Tool text editor under Files → Configuration files (see section 4.32 for details).

The following describes the different steps needed to add a stage, to modify the stage parameters and to build a manual configuration. Chapter 4.0 provides further information about some of the steps described here.

Once you are logged in as Administrator, click on **Stages** and then click on **"Add, remove or edit stages"**.

1. The following screen appears:



Add, remove or edit stages

In this page, the administrators can configure the stage configurations that will be selectable when building the controller configuration for each positioner.

Stages already in stages.ini (0)

Click on a stage to duplicate, rename, modify or delete it.

Stages in StageDataBase (598)

Click on a stage family to browse the list of stage configurations in it.

BGM	BGS	CMA
DUMMY	FMS	GTS
ILS	IMS	LTA
MFA	MTM	MTN
NPA	NPM	NPO
NPX	NPXY	NPXYZ
PR	PSM	RGV
RV	RVS	SR
TRA	TRB	URB
URM	URS	UTM
UTS	UZM	UZS
VP	XM	ZVR

Motion Controller / Driver - XPS-RL

©2015 Newport Corporation. All rights reserved.

2. Click on the family name from the Stages in StageDataBase list and a pop up window appears.

3. Click the part number corresponding to your hardware.

4. Select the driver (corresponding to your hardware) and configuration.

   For all continuous rotation stages, you can choose between a "regular" stage configuration and a "Spindle" configuration. A Spindle is a specific rotary device (no indexing) with a periodic position reset at 360° (by default), meaning 360° equals 0°. When defining the stage as Spindle in the stages.ini, you must assign this stage also to a Spindle group in the system configuration and vice versa. For details about Spindles, please refer to section 6.3.

   For some stages, you can choose between "regular" initialization or LMI (Large Move Initialization). The LMI method produces a larger movement of the stage for commutation and could be used if "regular" initialization fails.

5. The box "Use ESP Compatibility for Hardware detection" is checked by default. If your stage has an ESP chip inside (see the ESP-compatible sticker on the stage) this

box should remain checked. Otherwise, with vacuum compatible stages or with old Newport stages, or with non-Newport stages, uncheck this box.

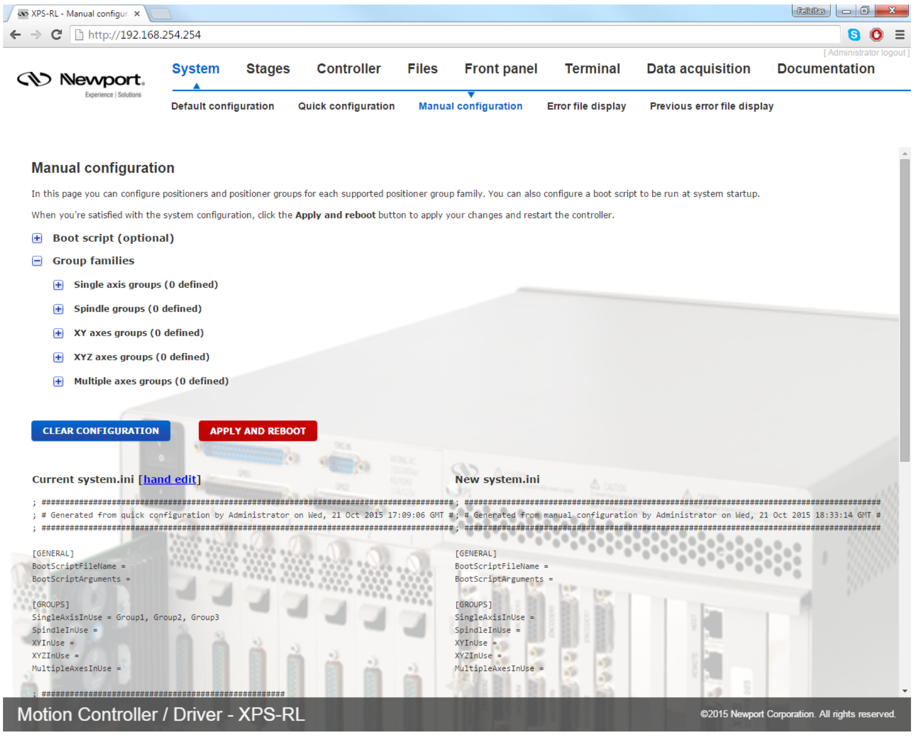
6. Click on **"ADD"** to add the stage to the stages.ini file.

   Once all stages have been added to the stages.ini file, you can review or modify these parameters from the screen **"Add, remove or edit stages"** under the main tab **Stages** by clicking on the stage name icon. A pop up windows appears which allows the user to make changes.

NOTE

From this screen, you have access to all stage parameters. Only experienced users should modify these parameters. For the exact meaning of the different parameters, please refer to chapter 23.0: "Appendix F: Configuration Wizard".

7. When all stages have been added to the stages.ini file, click on **"Manual Configuration"** under **SYSTEM**. The following screen appears:


XPS-RL - Manual configur ×
http://192.168.254.254
[Administrator logout]
Newport. Experience | Solutions
System
Stages
Controller
Files
Front panel
Terminal
Data acquisition
Documentation
Default configuration
Quick configuration
Manual configuration
Error file display
Previous error file display
Boot script (optional)
Group families
Single axis groups (0 defined)
Spindle groups (0 defined)
XY axes groups (0 defined)
XYZ axes groups (0 defined)
Multiple axes groups (0 defined)
CLEAR CONFIGURATION
APPLY AND REBOOT
Motion Controller / Driver - XPS-RL
©2015 Newport Corporation. All rights reserved.

Manual configuration

In this page you can configure positioners and positioner groups for each supported positioner group family. You can also configure a boot script to be run at system startup.

When you're satisfied with the system configuration, click the **Apply and reboot** button to apply your changes and restart the controller.

Current system.ini [hand edit]

```
; ##################################################################################
; # Generated from quick configuration by Administrator on Wed, 21 Oct 2015 17:09:06 GMT #
; ##################################################################################

[GENERAL]
BootScriptFileName =
BootScriptArguments =

[GROUPS]
SingleAxisInUse = Group1, Group2, Group3
SpindleInUse =
XYInUse =
XYZInUse =
MultipleAxesInUse =

; ##################################################
```

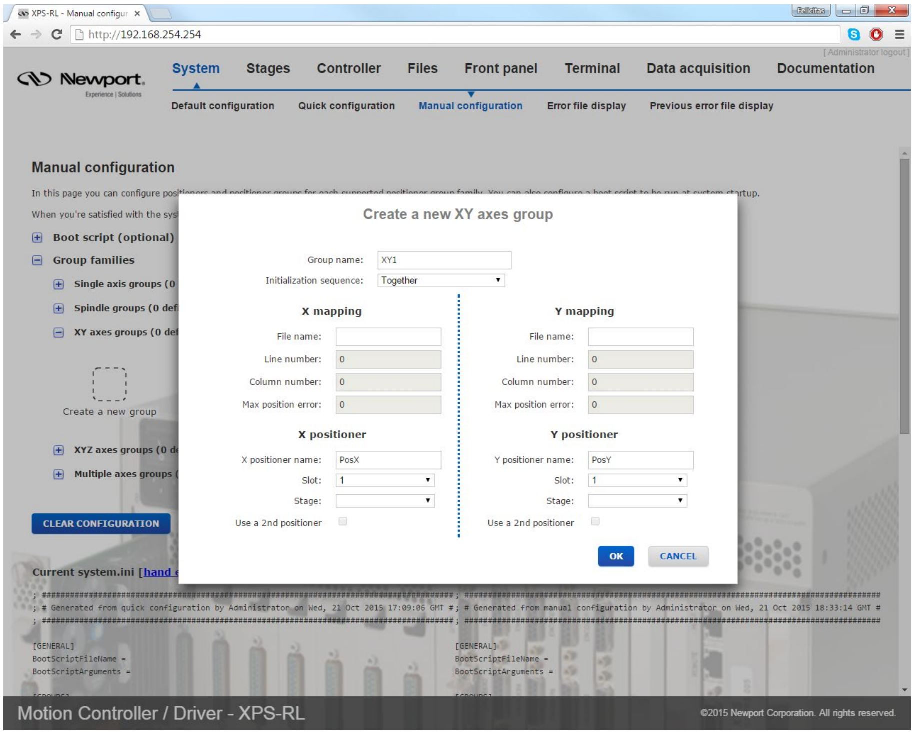
New system.ini

```
; ###################################################################################
; # Generated from manual configuration by Administrator on Wed, 21 Oct 2015 18:33:14 GMT #
; ###################################################################################

[GENERAL]
BootScriptFileName =
BootScriptArguments =

[GROUPS]
SingleAxisInUse =
SpindleInUse =
XYInUse =
XYZInUse =
MultipleAxesInUse =
```

8. Click on a Group type.

   For example, if you are setting up two ILS stages, you can set them up as two Single Axis groups, one XY group or one or two MulipleAxis groups.

9. Click on **"Create a new group"** and the following pop up window appears:



Manual configuration

In this page you can configure pos… …rtup.

When you're satisfied with the sys…

Create a new XY axes group

Group name: XY1

Initialization sequence: Together

X mapping		Y mapping	
File name:		File name:	
Line number:	0	Line number:	0
Column number:	0	Column number:	0
Max position error:	0	Max position error:	0

X positioner		Y positioner	
X positioner name:	PosX	Y positioner name:	PosY
Slot:	1	Slot:	1
Stage:		Stage:	
Use a 2nd positioner	□	Use a 2nd positioner	□

Current system.ini [hand …

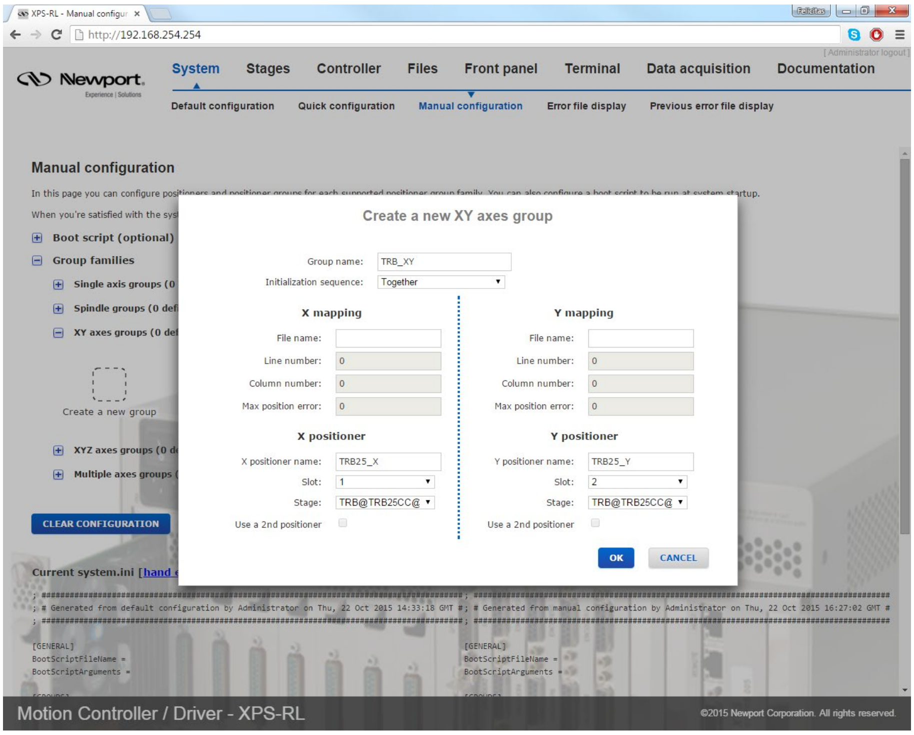
```
; ##########################################################################; ##########################################################################
; # Generated from quick configuration by Administrator on Wed, 21 Oct 2015 17:09:06 GMT #; # Generated from manual configuration by Administrator on Wed, 21 Oct 2015 18:33:14 GMT #
; ##########################################################################; ##########################################################################

[GENERAL]                                   [GENERAL]
BootScriptFileName =                        BootScriptFileName =
BootScriptArguments =                       BootScriptArguments =
```

10. Enter the group name as well as the positioner names.

Any group name and any positioner name can be used. In this example the group name is TRB_XY and the X positioner name is TRB25_X. The home sequence can be either “Together” or “X then Y”.

The other fields refer to the error compensation (mapping) of the XPS controller, see chapter 10.0 for details. For the first configuration, don’t enter anything in these fields.

11. Enter the appropriate Slot number. The Slot number is the axis number where the stage is physically connected to the XPS controller. Looking at the rear of the controller, plug number 1 is the first plug on the left and the number increases to the right.



Manual configuration

In this page you can configure pos… …rtup.

When you're satisfied with the sys…

Create a new XY axes group

Group name: TRB_XY

Initialization sequence: Together

X mapping		Y mapping	
File name:		File name:	
Line number:	0	Line number:	0
Column number:	0	Column number:	0
Max position error:	0	Max position error:	0

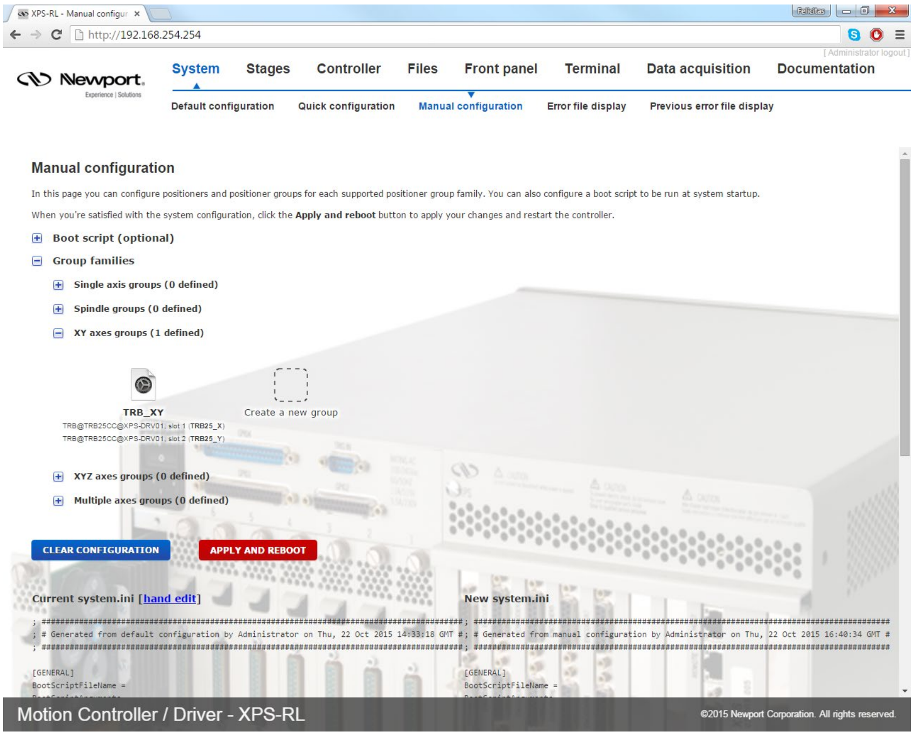
X positioner		Y positioner	
X positioner name:	TRB25_X	Y positioner name:	TRB25_Y
Slot:	1	Slot:	2
Stage:	TRB@TRB25CC@	Stage:	TRB@TRB25CC@
Use a 2nd positioner	□	Use a 2nd positioner	□

Current system.ini [hand …

```
; ##########################################################################; ##########################################################################
; # Generated from default configuration by Administrator on Thu, 22 Oct 2015 14:33:18 GMT #; # Generated from manual configuration by Administrator on Thu, 22 Oct 2015 16:27:02 GMT #
; ##########################################################################; ##########################################################################

[GENERAL]                                   [GENERAL]
BootScriptFileName =                        BootScriptFileName =
BootScriptArguments =                       BootScriptArguments =
```

12. Select the StageName from the list of stages. These stage names refer to the stages defined with the Web Tool "Stage Management".

13. Click on **"OK"** to return to the initial screen.



Manual configuration

In this page you can configure positioners and positioner groups for each supported positioner group family. You can also configure a boot script to be run at system startup.

When you're satisfied with the system configuration, click the **Apply and reboot** button to apply your changes and restart the controller.

TRB@TRB25CC@XPS-DRV01, slot 1 (TRB25_X)
TRB@TRB25CC@XPS-DRV01, slot 2 (TRB25_Y)

Current system.ini [hand edit]

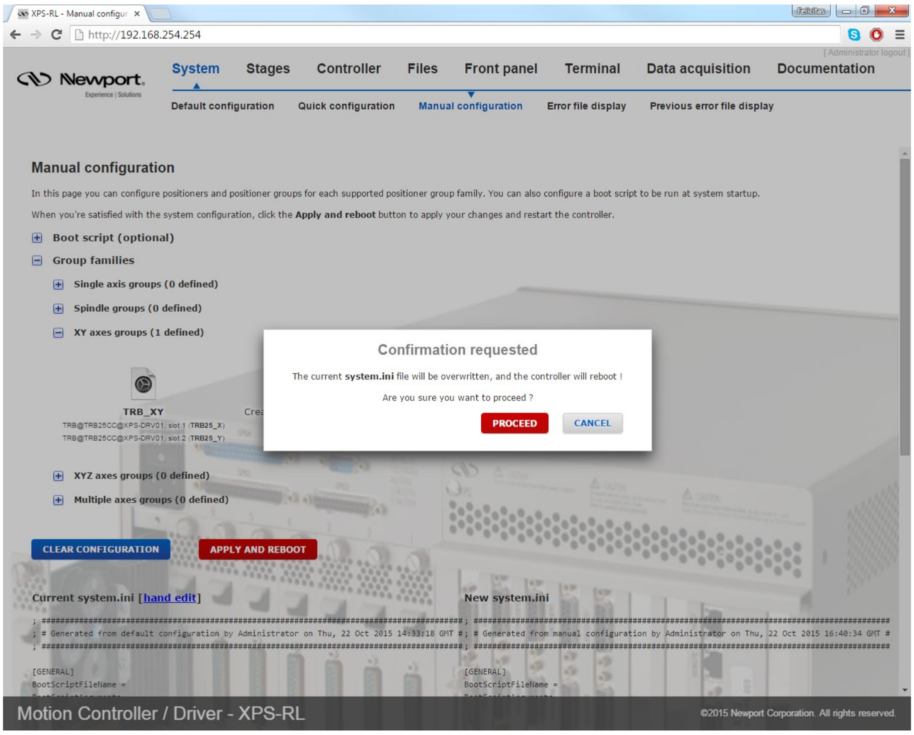
```
; ###########################################################################
; # Generated from default configuration by Administrator on Thu, 22 Oct 2015 14:33:18 GMT #
; ###########################################################################

[GENERAL]
BootScriptFileName =
```

New system.ini

```
; ###########################################################################
; # Generated from manual configuration by Administrator on Thu, 22 Oct 2015 16:40:34 GMT #
; ###########################################################################

[GENERAL]
BootScriptFileName =
```

14. Continue the same way with the other motion groups.

15. When done, click on **"APPLY AND REBOOT"** to complete the System configuration. The controller re-boots and the following message appears:



Manual configuration

In this page you can configure positioners and positioner groups for each supported positioner group family. You can also configure a boot script to be run at system startup.

When you're satisfied with the system configuration, click the **Apply and reboot** button to apply your changes and restart the controller.

TRB@TRB25CC@XPS-DRV01, slot 1 (TRB25_X)
TRB@TRB25CC@XPS-DRV01, slot 2 (TRB25_Y)

Current system.ini [hand edit]

New system.ini

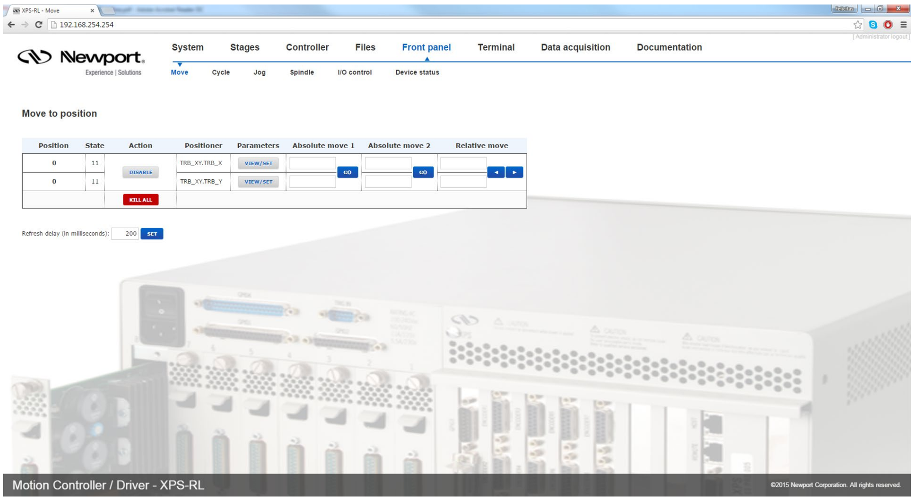
Confirmation requested

The current **system.ini** file will be overwritten, and the controller will reboot !

Are you sure you want to proceed ?

Click on **"PROCEED".**

When the controller has finished booting (a second beep after 12-18 seconds), you are redirected to the login page. Login then select **FRONT PANEL**, then select **"Move"**. The following screen appears (Group names will be different according to your definition):



Move to position

Position	State	Action	Positioner	Parameters	Absolute move 1	Absolute move 2	Relative move
0	11	DISABLE	TRB_XY.TRB_X	VIEW/SET	GO	GO	◄ ►
0	11		TRB_XY.TRB_Y	VIEW/SET			
		KILL ALL					

Refresh delay (in milliseconds): 200 SET

Click "Initialize". The State number changes from 0 to 42 and the Action button changes from "Initialize" to "Home". Click "Home". The stage starts moving to find its reference position. When done, the state number is 11 and the action button is "Disable". Enter an allowed position value in the "Abs move 1" field and click "Go". The stage moves to this absolute position.

Your system is now ready to use. For more advanced functions, please read the rest of this manual.

3.8.4 Manual Configuration for non Newport stages

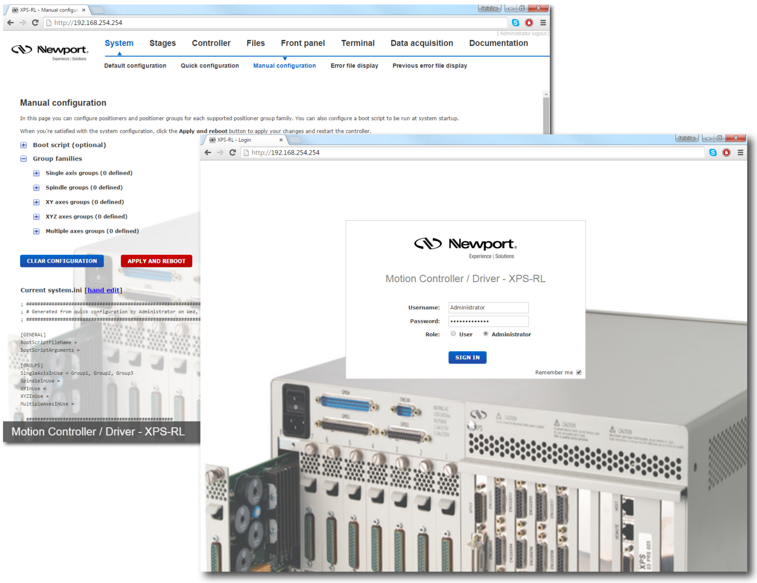
To configure the XPS controller to stages or positioning devices not made by Newport, use the tool **"Create custom stages"** under the main tab **STAGE**. For detailed information about this tool, please refer to chapter 23.0: "Appendix F: Configuration Wizard"

3.9 System Shut-Down

To shut down the system entirely, perform the following procedure:

Wait for the stage(s) to complete their moves and come to a stop.

Turn off the power using the power switch located above the power cord at the back of the controller.



Manual configuration

In this page you can configure positioners and positioner groups for each supported positioner group family. You can also configure a boot script to be run at system startup.

When you're satisfied with the system configuration, click the **Apply and reboot** button to apply your changes and restart the controller.

⊞ Boot script (optional)

⊟ Group families

- ⊞ Single axis groups (0 defined)
- ⊞ Spindle groups (0 defined)
- ⊞ XY axes groups (0 defined)
- ⊞ XYZ axes groups (0 defined)
- ⊞ Multiple axes groups (0 defined)

Current system.ini [hand edit]

```
; ##########################################
; # Generated from quick configuration by Administrator on Wed,
; ##########################################

[GENERAL]
BootScriptFileName =
BootScriptArguments =

[GROUPS]
SingleAxisInUse = Group1, Group2, Group3
SpindleInUse =
XYInUse =
XYZInUse =
MultipleAxesInUse =
```

Motion Controller / Driver - XPS-RL

Username: Administrator

Password: ••••••••••••••

Role: ○ User ● Administrator

Remember me ☑

Software Tools

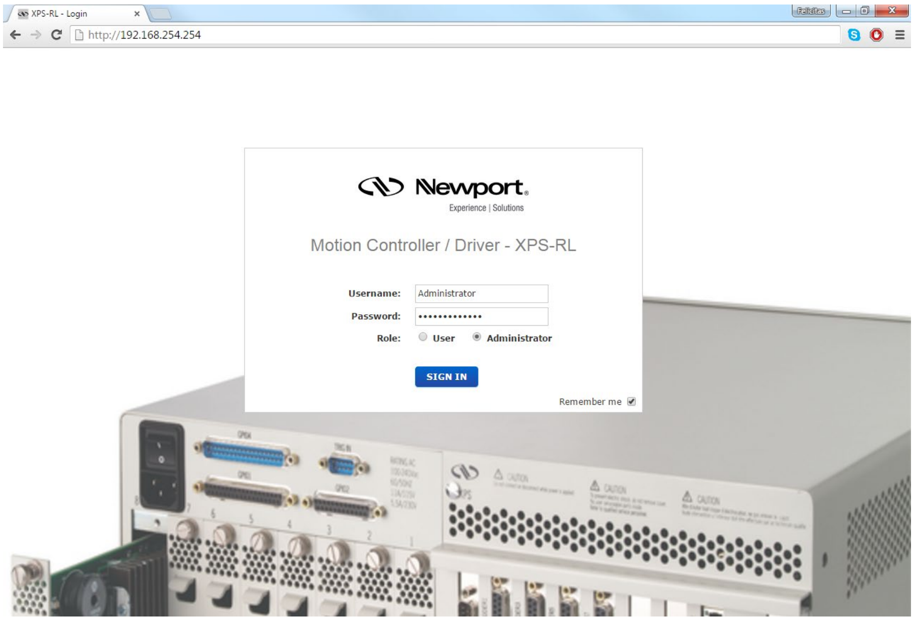
4.0 Software Tools

4.1 Software Tools Overview

The XPS software tools provide users a convenient access to the most common features and functions of the XPS controller. All software tools are implemented as a web interface. The advantage of a web interface is that it is independent from the user's operating system and doesn't require any specific software on the host PC.

There are two options to log-in to the XPS controller: as “User” or as “Administrator”. Users can log-in only with User rights. Administrators can log-in with User or with Administrator rights by selecting the respective Role in the login page. When logged-in with Administrator rights, you have an extended set of tools available.

The predefined user has the log-in name **Anonymous**, Password **Anonymous**. The predefined Administrator has the log-in name **Administrator**, Password **Administrator**. Both the log-in name and the password are case sensitive. Select **"Remember me"** to save the login credentials.



Motion Controller / Driver - XPS-RL

Username: Administrator

Password: ••••••••••••••

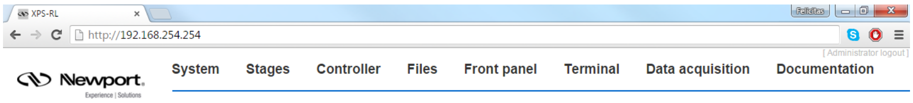
Role: ○ User ● Administrator

Remember me ☑

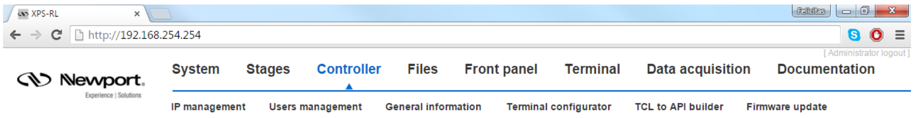
The main tab is displayed across the top of the XPS Motion Controller/Driver main program window, and lists each primary interface option. Each interface option has its own pull-down menu that allows the user to select various options by clicking the mouse's left button.

On the following pages, a brief description of all available tools is provided.

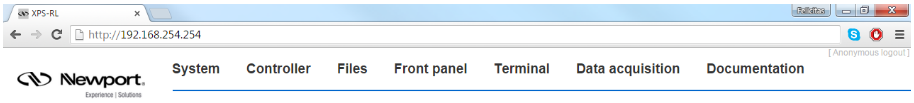
Administrator Menus (with Administrator Rights)



Sub-Menu for CONTROLLER (with Administrator Rights)



Restricted set of User Menus



4.2 Restart and Reboot

After making changes to the hardware or software parameters a Restart or a Reboot of the controller will be a necessary to apply the changes. Restart is a quicker processes than a Reboot.

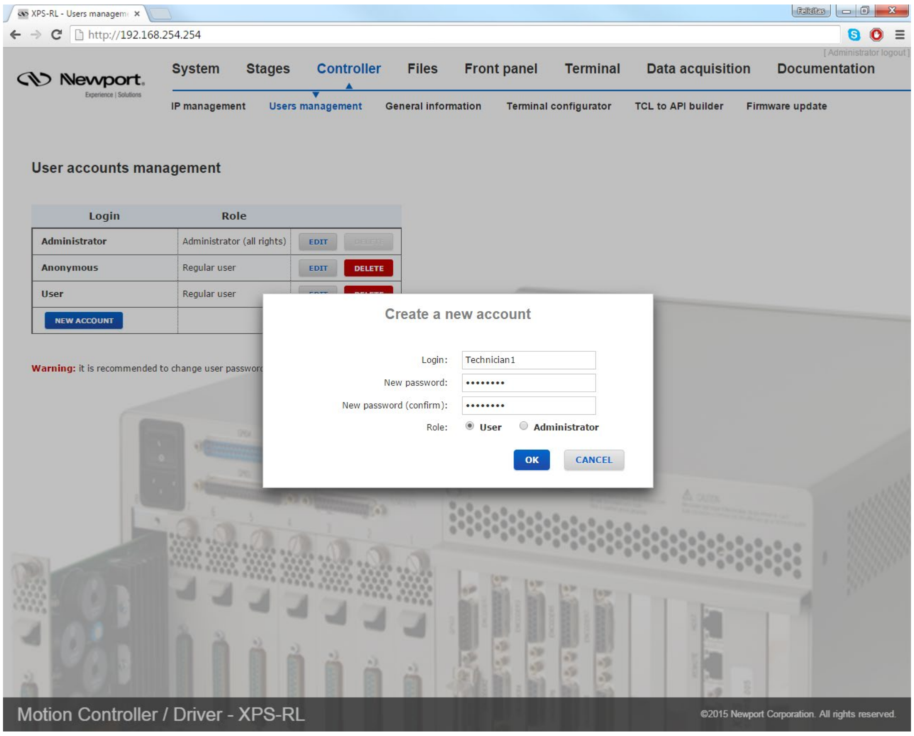
- Restart = reload all configuration files.
- Reboot = reload files + re-initialize hardware.

4.3 Controller – Users Management

This tool allows managing User accounts. There are two types of users: Administrators and Users. Administrators have configurations rights. Users have restricted rights to use the system.

The following steps are needed to create a new user:

1. Click on "New Account" and the following window appears:



User accounts management

Login	Role		
Administrator	Administrator (all rights)	EDIT	DELETE
Anonymous	Regular user	EDIT	DELETE
User	Regular user	EDIT	DELETE
NEW ACCOUNT			

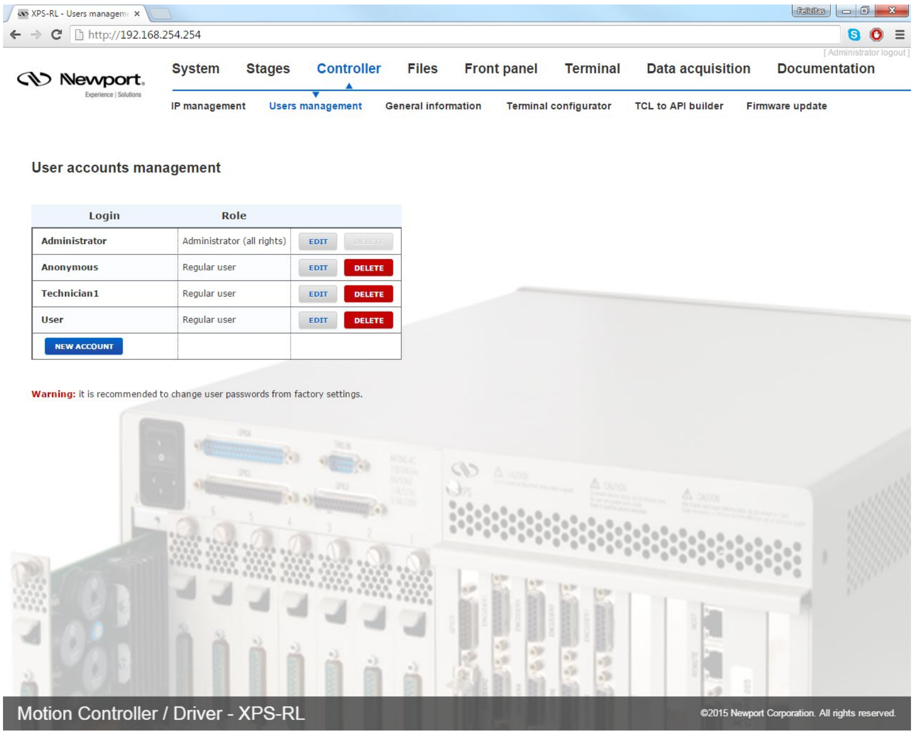
Warning: it is recommended to change user passwor

Create a new account

Login: Technician1
New password: ••••••••
New password (confirm): ••••••••
Role: ● User ○ Administrator

OK CANCEL

2. Type in a Login name, password, and role (User or Administrator).
3. Click **“OK”** to add the new access account.



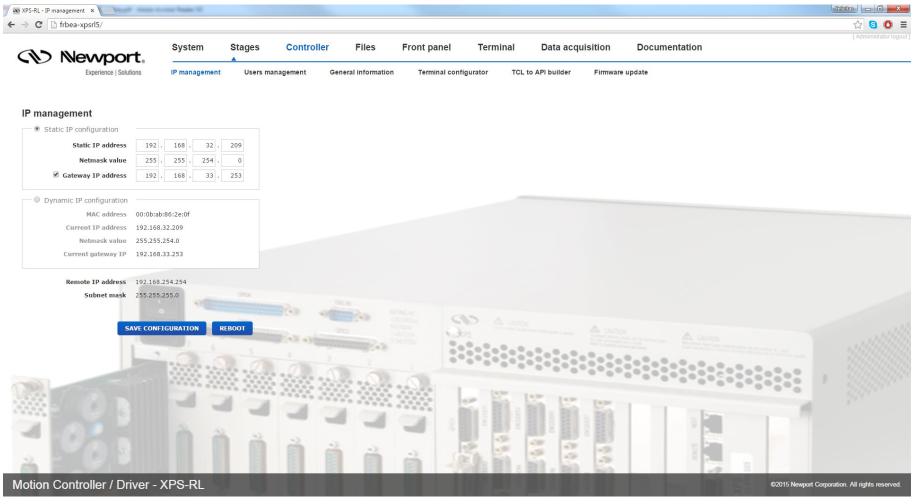
User accounts management

Login	Role		
Administrator	Administrator (all rights)	EDIT	DELETE
Anonymous	Regular user	EDIT	DELETE
Technician1	Regular user	EDIT	DELETE
User	Regular user	EDIT	DELETE
NEW ACCOUNT			

Warning: it is recommended to change user passwords from factory settings.

4.4 Controller – IP Management

To access this Web Tool, users must be logged in with Administrator rights (see section 3.5 for details).



IP management

● Static IP configuration

Static IP address	192 . 168 . 32 . 209
Netmask value	255 . 255 . 254 . 0
☑ Gateway IP address	192 . 168 . 33 . 253

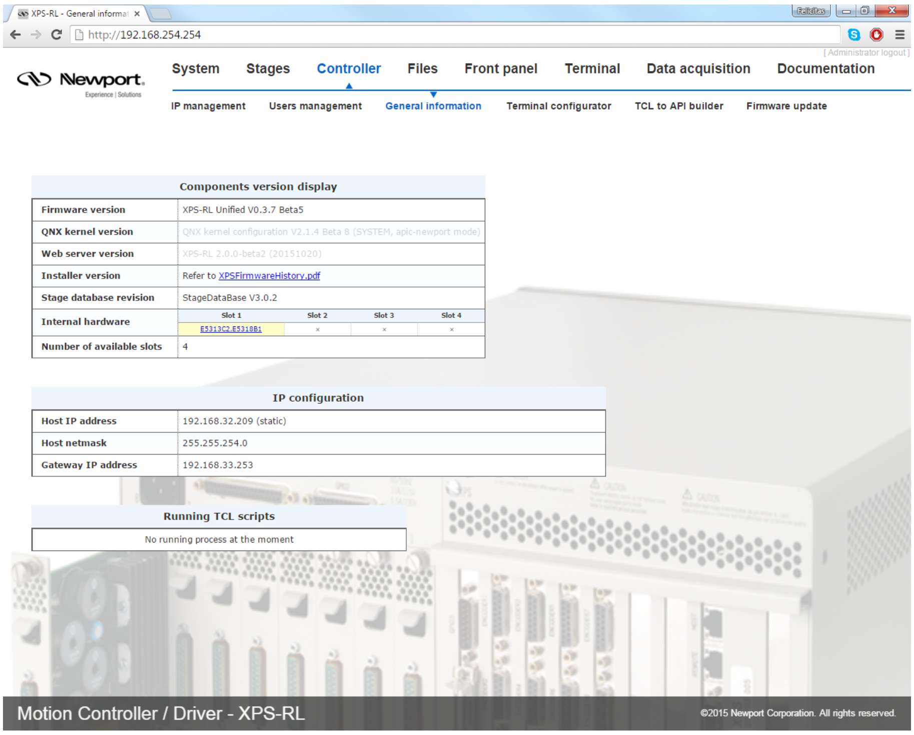
○ Dynamic IP configuration

MAC address	00:0b:ab:86:2e:0f
Current IP address	192.168.32.209
Netmask value	255.255.254.0
Current gateway IP	192.168.33.253

Remote IP address	192.168.254.254
Subnet mask	255.255.255.0

4.5 Controller – General

This screen provides valuable information about the firmware and the hardware of the controller. It is an important screen for troubleshooting the controller. This screen also displays information about the IP configuration as well as TCL scripts which are currently running.



Components version display				
Firmware version	XPS-RL Unified V0.3.7 Beta5			
QNX kernel version	QNX kernel configuration V2.1.4 Beta 8 (SYSTEM, apic-newport mode)			
Web server version	XPS-RL 2.0.0-beta2 (20151020)			
Installer version	Refer to XPSFirmwareHistory.pdf			
Stage database revision	StageDataBase V3.0.2			
Internal hardware	Slot 1	Slot 2	Slot 3	Slot 4
	E5313C2,E5318B1	×	×	×
Number of available slots	4			

IP configuration	
Host IP address	192.168.32.209 (static)
Host netmask	255.255.254.0
Gateway IP address	192.168.33.253

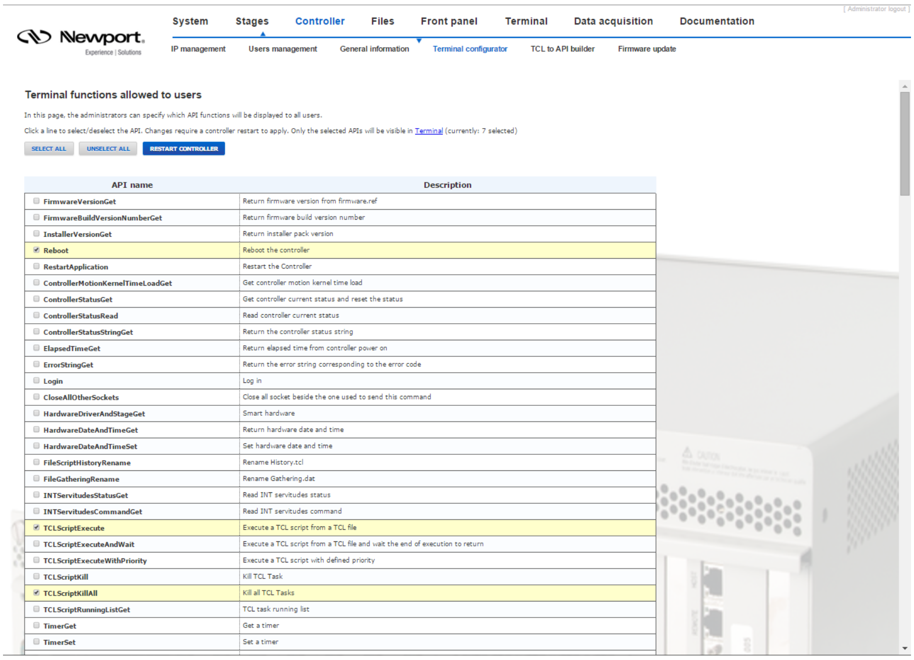
Running TCL scripts
No running process at the moment

4.6 Controller – Terminal Configurator

Under **Controller → Terminal configurator**, an Administrator user can specify which API functions will be displayed to all users in the **Terminal** webpage. Not all API functions may be useful to a given application especially after the application has been developed. For this purpose simplifying the Terminal display may be helpful.

1. Click a line to select/deselect the API function.
2. Once all desired API functions are selected, click **RESTART CONTROLLER**.

In the following example a total of 7 API functions were selected.

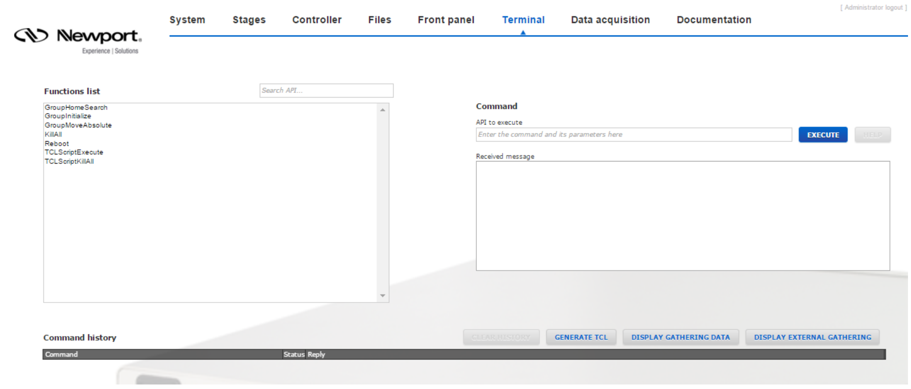


Terminal functions allowed to users

In this page, the administrators can specify which API functions will be displayed to all users.

Click a line to select/deselect the API. Changes require a controller restart to apply. Only the selected APIs will be visible in Terminal (currently: 7 selected)

API name	Description
□ FirmwareVersionGet	Return firmware version from firmware.ref
□ FirmwareBuildVersionNumberGet	Return firmware build version number
□ InstallerVersionGet	Return installer pack version
☑ Reboot	Reboot the controller
□ RestartApplication	Restart the Controller
□ ControllerMotionKernelTimeLoadGet	Get controller motion kernel time load
□ ControllerStatusGet	Get controller current status and reset the status
□ ControllerStatusRead	Read controller current status
□ ControllerStatusStringGet	Return the controller status string
□ ElapsedTimeGet	Return elapsed time from controller power on
□ ErrorStringGet	Return the error string corresponding to the error code
□ Login	Log in
□ CloseAllOtherSockets	Close all socket beside the one used to send this command
□ HardwareDriverAndStageGet	Smart hardware
□ HardwareDateAndTimeGet	Return hardware date and time
□ HardwareDateAndTimeSet	Set hardware date and time
□ FileScriptHistoryRename	Rename History.tcl
□ FileGatheringRename	Rename Gathering.dat
□ INTServitudesStatusGet	Read INT servitudes status
□ INTServitudesCommandGet	Read INT servitudes command
☑ TCLScriptExecute	Execute a TCL script from a TCL file
□ TCLScriptExecuteAndWait	Execute a TCL script from a TCL file and wait the end of execution to return
□ TCLScriptExecuteWithPriority	Execute a TCL script with defined priority
□ TCLScriptKill	Kill TCL Task
☑ TCLScriptKillAll	Kill all TCL Tasks
□ TCLScriptRunningListGet	TCL task running list
□ TimerGet	Get a timer
□ TimerSet	Set a timer



GroupHomeSearch
GroupInitialize
GroupMoveAbsolute
KillAll
Reboot
TCLScriptExecute
TCLScriptKillAll

4.7 Controller – TCL to API Builder

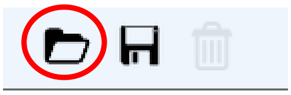
Under **Controller →TCL to API builder**, users can write custom API functions from a TCL script then add the custom API to the terminal and launch the custom API.

Example

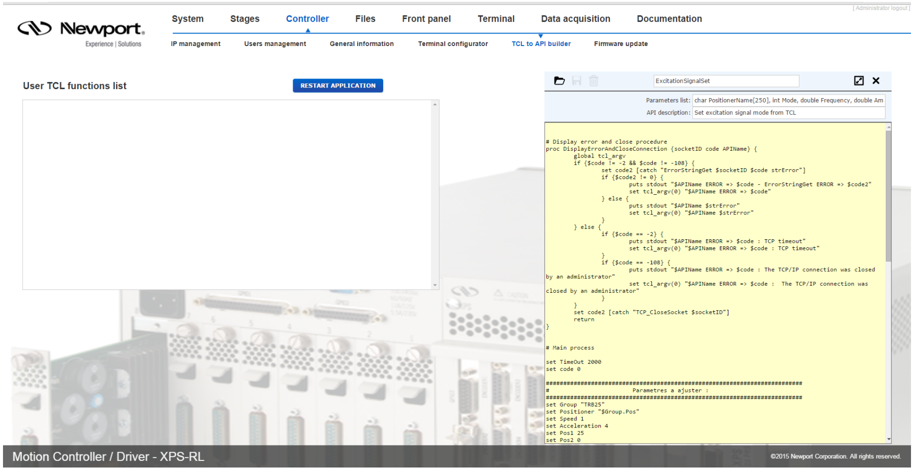
For this example, there must be a TCL file "ExcitationSignalSet.tcl" uploaded to the XPS controller.

Go to the XPS **Controller → TCL To API builder** webpage:

1. Load the existing TCL script named ExcitationSignalSet.tcl by clicking on the folder icon in the text editor.



2. In the Parameters list enter the arguments "char PositionerName[250], int Mode, double Frequency, double Amplitude, double Time".
3. Enter a description in the API description: "Set excitation signal mode from TCL".



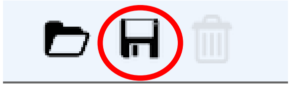
ExcitationSignalSet

Parameters list: char PositionerName[250], int Mode, double Frequency, double Am

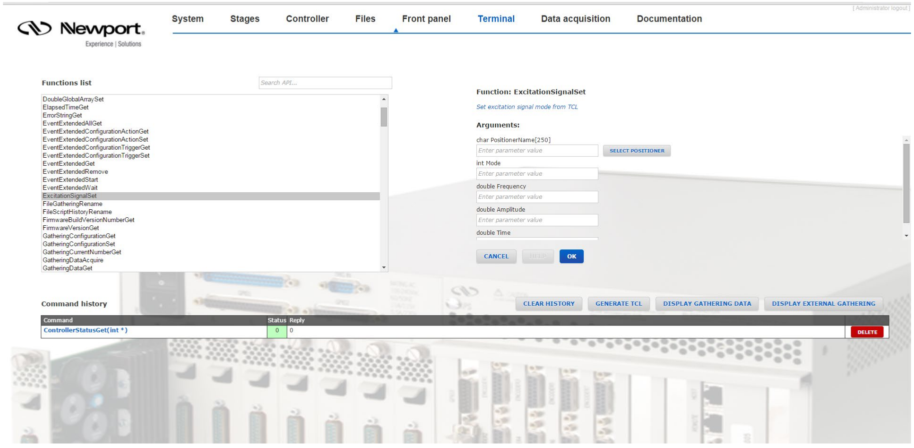
API description: Set excitation signal mode from TCL

```
# Display error and close procedure
proc DisplayErrorAndCloseConnection {socketID code APIName} {
        global tcl_argv
        if {$code != -2 && $code != -108} {
                set code2 [catch "ErrorStringGet $socketID $code strError"]
                if {$code2 != 0} {
                        puts stdout "$APIName ERROR => $code - ErrorStringGet ERROR => $code2"
                        set tcl_argv(0) "$APIName ERROR => $code"
                } else {
                        puts stdout "$APIName $strError"
                        set tcl_argv(0) "$APIName $strError"
                }
        } else {
                if {$code == -2} {
                        puts stdout "$APIName ERROR => $code : TCP timeout"
                        set tcl_argv(0) "$APIName ERROR => $code : TCP timeout"
                }
                if {$code == -108} {
                        puts stdout "$APIName ERROR => $code : The TCP/IP connection was closed
by an administrator"
                        set tcl_argv(0) "$APIName ERROR => $code :  The TCP/IP connection was
closed by an administrator"
                }
        }
        set code2 [catch "TCP_CloseSocket $socketID"]
        return
}


# Main process

set TimeOut 2000
set code 0

###########################################################################
#                         Parametres a ajuster :
###########################################################################
set Group "TRB25"
set Positioner "$Group.Pos"
set Speed 1
set Acceleration 4
set Pos1 25
set Pos2 0
```

4. Save the custom API by clicking in the floppy disk icon in the text editor.



5. In the TCL function list, "int ExcitationSignalSet(char PositionerName[250], int Mode, double Frequency, double Amplitude, double Time) // Set excitation signal mode from TCL" is added.
6. Click "RESTART APPLICATION" to take in account the changes.
7. Connect to the website, go to the "Terminal" page and search for the new API, "ExcitationSignalSet," in the **Function list**.



Functions list

DoubleGlobalArraySet
ElapsedTimeGet
ErrorStringGet
EventExtendedAllGet
EventExtendedConfigurationActionGet
EventExtendedConfigurationActionSet
EventExtendedConfigurationTriggerGet
EventExtendedConfigurationTriggerSet
EventExtendedGet
EventExtendedRemove
EventExtendedStart
EventExtendedWait
ExcitationSignalSet
FileGatheringRename
FileScriptHistoryRename
FirmwareBuildVersionNumberGet
FirmwareVersionGet
GatheringConfigurationGet
GatheringConfigurationSet
GatheringCurrentNumberGet
GatheringDataAcquire
GatheringDataGet

Function: ExcitationSignalSet

Set excitation signal mode from TCL

Arguments:

char PositionerName[250]

int Mode

double Frequency

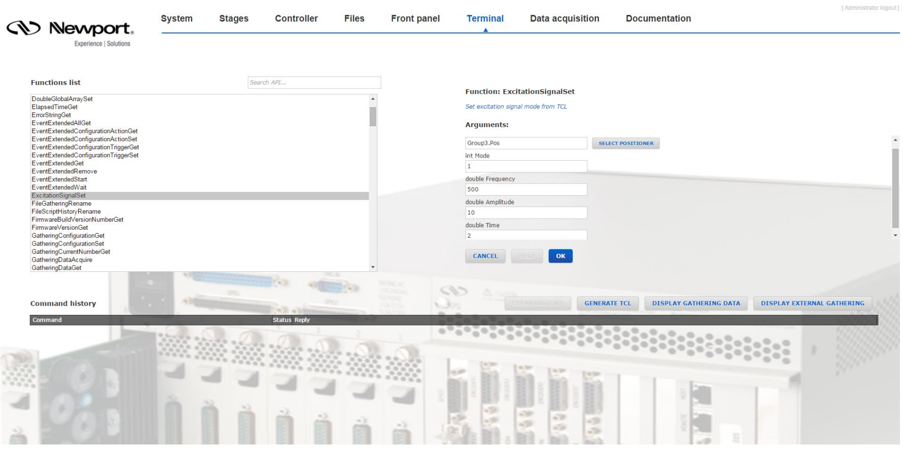
double Amplitude

double Time

Command history

Command	Status	Reply
ControllerStatusGet(int *)	0	0

8. Select new API and enter the parameters as usual.



Function: ExcitationSignalSet

Set excitation signal mode from TCL

Arguments:

Group3.Pos

int Mode: 1

double Frequency: 500

double Amplitude: 10

double Time: 2

…

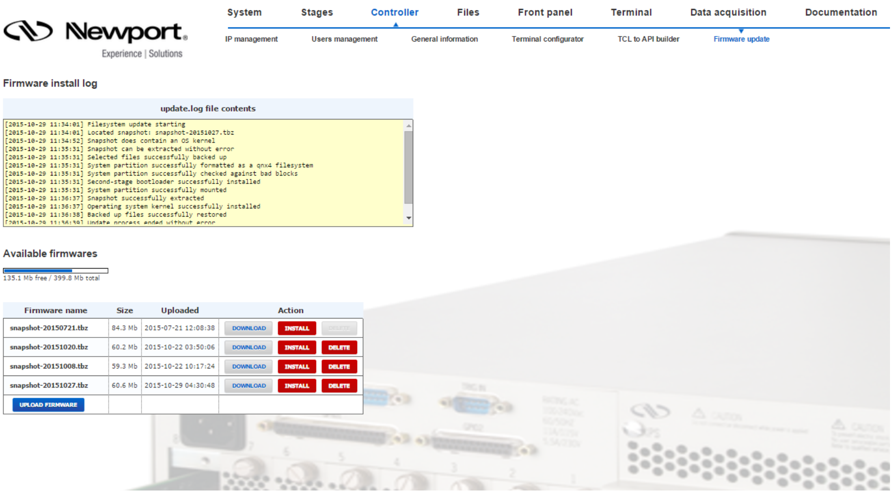
4.8 Controller – Firmware Update

Users can regularly update the controller with new firmware releases. Updating the firmware does overwrite the stages.ini or system.ini files if changes are required. The configuration will also be reset when upgrading the firmware hence the Configuration should be backed up prior to the firmware upgrade. Refer to the FirmwareHistory document which explains changes to the stages.ini and system.ini files, if any.

Refer to the XPS webpage at www.newport.com for more information including the FirmwareHistory document, the StageDataBase.txt file and the new firmware installer pack.

Updating the XPS Firmware

1. Download the firmware installer pack from the XPS webpage at www.newport.com.
2. Connect to the XPS controller. For more information see section 3.5: "Connecting to the XPS".
3. Login on to the XPS with Administrator rights.
4. Go to **Controller → Firmware update**.



Firmware install log

update.log file contents

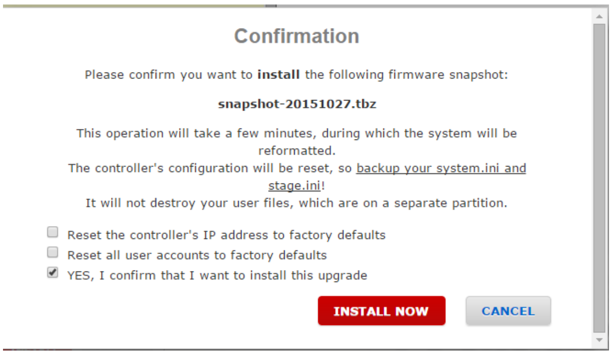
```
[2015-10-29 11:34:01] Filesystem update starting
[2015-10-29 11:34:01] Located snapshot: snapshot-20151027.tbz
[2015-10-29 11:34:52] Snapshot does contain an OS kernel
[2015-10-29 11:35:31] Snapshot can be extracted without error
[2015-10-29 11:35:31] Selected files successfully backed up
[2015-10-29 11:35:31] System partition successfully formatted as a qnx4 filesystem
[2015-10-29 11:35:31] System partition successfully checked against bad blocks
[2015-10-29 11:35:31] Second-stage bootloader successfully installed
[2015-10-29 11:35:31] System partition successfully mounted
[2015-10-29 11:36:37] Snapshot successfully extracted
[2015-10-29 11:36:37] Operating system kernel successfully installed
[2015-10-29 11:36:38] Backed up files successfully restored
[2015-10-29 11:36:39] Update process ended without error
```

Available firmwares

135.1 Mb free / 399.8 Mb total

Firmware name	Size	Uploaded	Action		
snapshot-20150721.tbz	84.3 Mb	2015-07-21 12:08:38	DOWNLOAD	INSTALL	DELETE
snapshot-20151020.tbz	60.2 Mb	2015-10-22 03:50:06	DOWNLOAD	INSTALL	DELETE
snapshot-20151008.tbz	59.3 Mb	2015-10-22 10:17:24	DOWNLOAD	INSTALL	DELETE
snapshot-20151027.tbz	60.6 Mb	2015-10-29 04:30:48	DOWNLOAD	INSTALL	DELETE
UPLOAD FIRMWARE					

5. Click on UPLOAD FIRMWARE and select the installer pack file saved on the PC.
6. Click INSTALL and the following Confirmation window appears:



Confirmation

Please confirm you want to **install** the following firmware snapshot:

snapshot-20151027.tbz

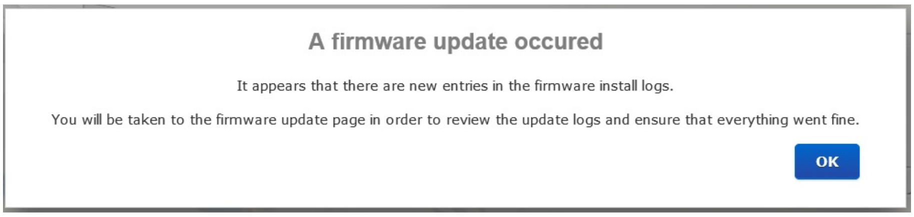
This operation will take a few minutes, during which the system will be reformatted.
The controller's configuration will be reset, so backup your system.ini and stage.ini!
It will not destroy your user files, which are on a separate partition.

□ Reset the controller's IP address to factory defaults
□ Reset all user accounts to factory defaults
☑ YES, I confirm that I want to install this upgrade

7. Select "Yes, I confirm that I want to install this upgrade." Resetting the controller IP address or user accounts is optional.
8. Click INSTALL NOW.

NOTE

This will reboot the controller and reset the controller configuration.



A firmware update occured

It appears that there are new entries in the firmware install logs.

You will be taken to the firmware update page in order to review the update logs and ensure that everything went fine.

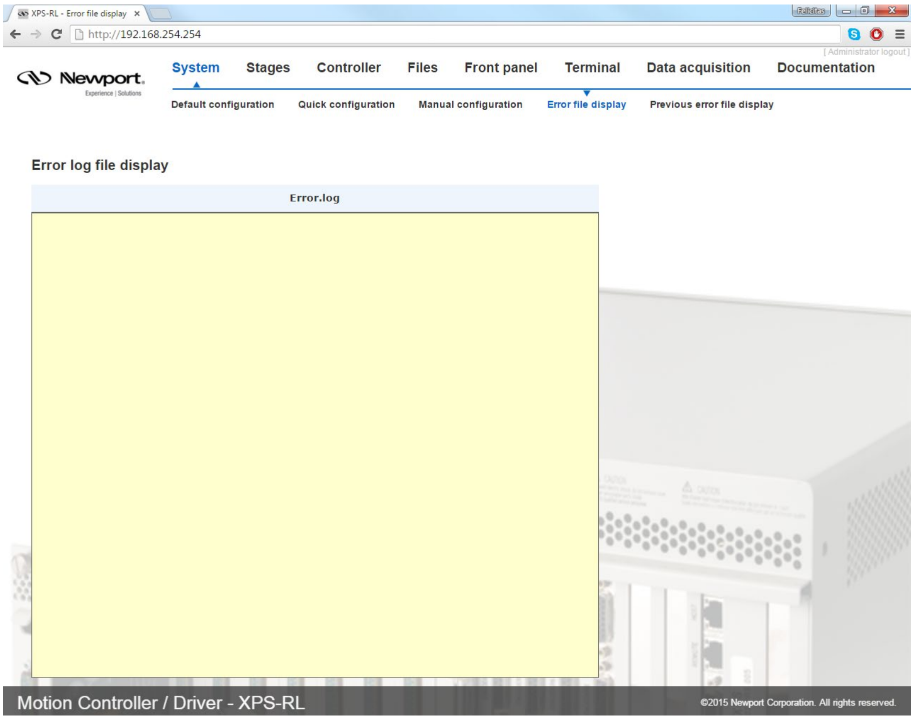
NOTE

Controller configuration files including stages.ini and system.ini files can be downloaded under Files→ Configuration files prior to updating the controller firmware. See chapter 4.30: “Files – Configuration Files” for more information.

4.9 System – Error file display

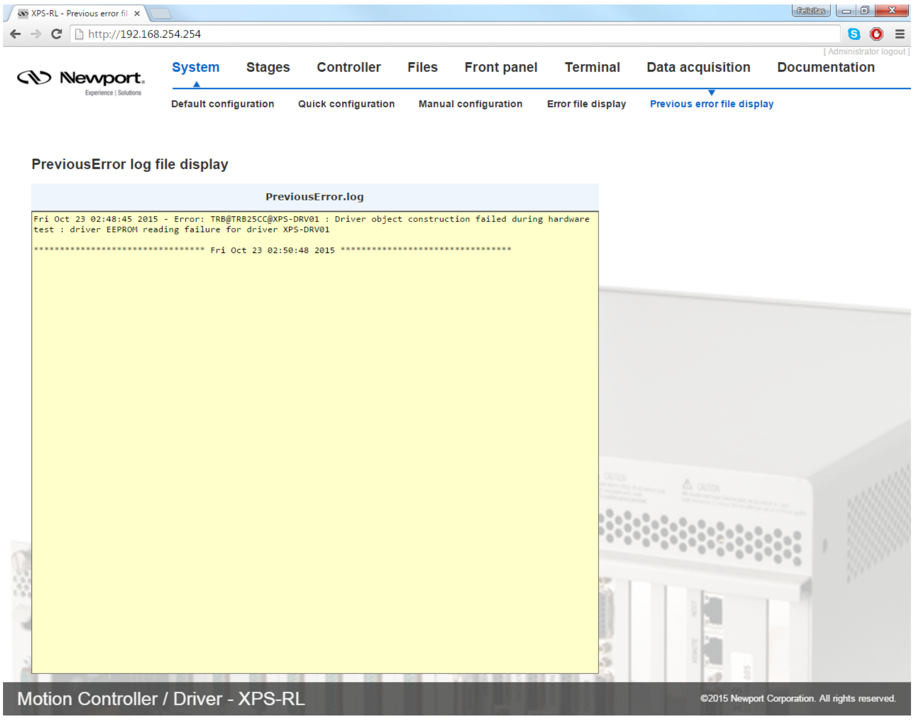
The Error File Display is another important screen for troubleshooting the XPS controller. When the XPS encounters any error during booting, for instance due to an error in the configuration files or because the configuration is not compatible with the connected hardware, there are entries in the error log file that guides you to correct the error.

When no error is detected during the system boot, this file is blank.



4.10 System – Last error file display

The Last error file display shows errors encountered in the last XPS boot. When no error is detected during the last system boot, this file is blank.



PreviousError log file display

PreviousError.log

```
Fri Oct 23 02:48:45 2015 - Error: TRB@TRB25CC@XPS-DRV01 : Driver object construction failed during hardware
test : driver EEPROM reading failure for driver XPS-DRV01

******************************** Fri Oct 23 02:50:48 2015 ********************************
```

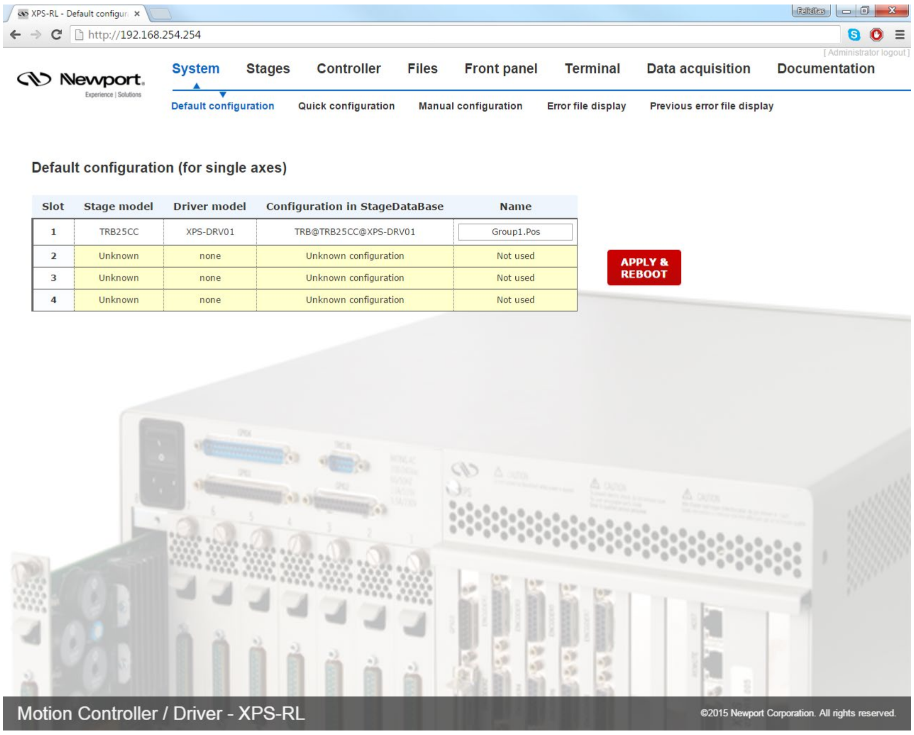
4.11 System – Default Configuration

With the help of this screen, a fast, basic configuration of the XPS controller can be done. When done, click **“APPLY & BOOT”**. The XPS controller reboots. After re-booting, you are able to use the XPS controller in this basic, SingleAxis group type configuration. For further information, refer to section 3.8.1: “Default Configuration”.

NOTE

“APPLY & BOOT” deletes your current system.ini configuration file. For troubleshooting a system, make sure to backup the original system.ini file for recovery.

The Default configuration (for single axes) lists all Newport ESP compatible stages and motor drivers under the respectively columns Stage Model and Driver model as seen by the XPS controller. As a result, this screen also provides valuable information for diagnosing or troubleshooting the system.



Default configuration (for single axes)

Slot	Stage model	Driver model	Configuration in StageDataBase	Name
1	TRB25CC	XPS-DRV01	TRB@TRB25CC@XPS-DRV01	Group1.Pos
2	Unknown	none	Unknown configuration	Not used
3	Unknown	none	Unknown configuration	Not used
4	Unknown	none	Unknown configuration	Not used

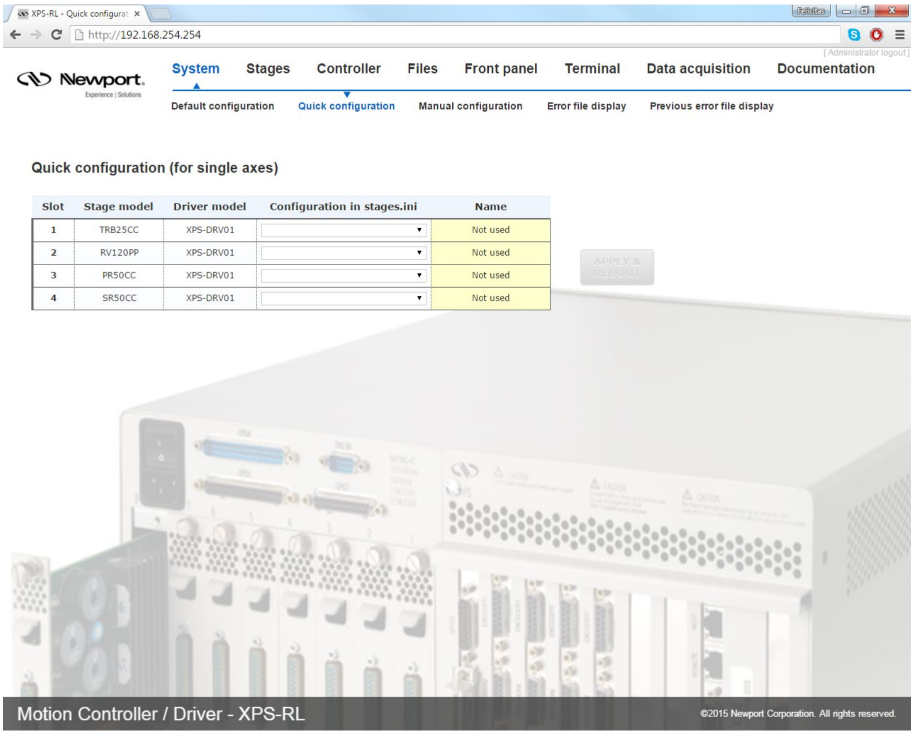
4.12 System – Quick Configuration

The Quick configuration is very similar to the Default configuration as it also lists all detected hardware including Newport ESP compatible stages and motor drivers under the respectively columns Stage Model and Driver model. The Quick configuration differs in the source for stage configuration. For the Quick configuration the stage configuration is taken from the stages.ini where as the Default configuration takes the stage configuration from the StageDataBase.

As a result, this screen also provides valuable information for diagnosing or troubleshooting the system.

NOTE

“APPLY and BOOT” deletes your current system.ini configuration files. For diagnosing or troubleshooting a system, make sure to first backup the original system.ini of the system.



Quick configuration (for single axes)

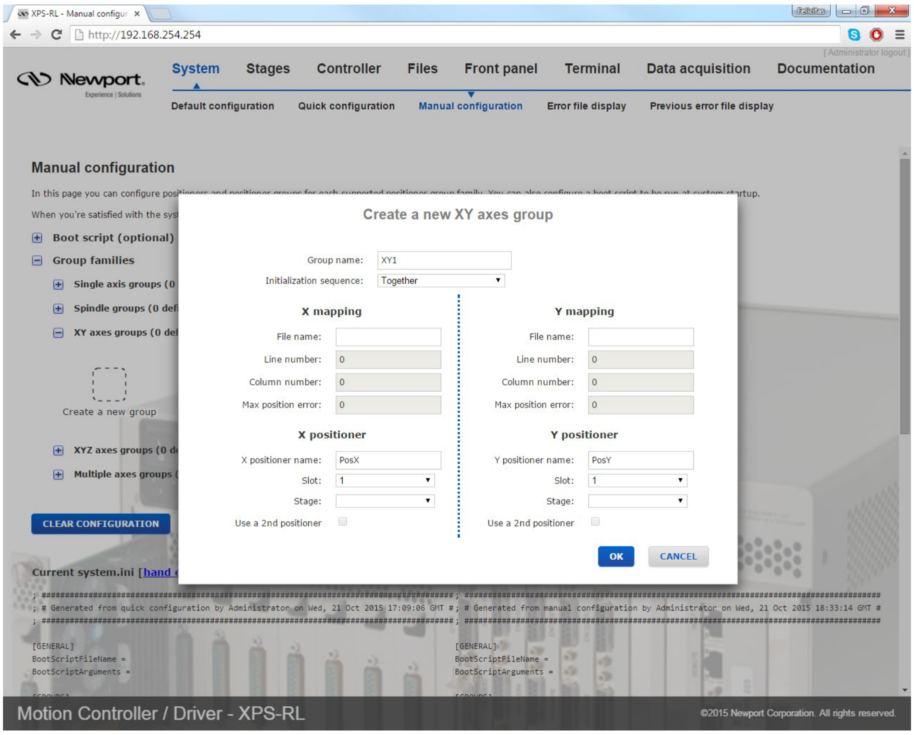
Slot	Stage model	Driver model	Configuration in stages.ini	Name
1	TRB25CC	XPS-DRV01	▾	Not used
2	RV120PP	XPS-DRV01	▾	Not used
3	PR50CC	XPS-DRV01	▾	Not used
4	SR50CC	XPS-DRV01	▾	Not used

4.13 System – Manual Configuration

Manual Configuration allows you to review the current system configuration or to define a new one. See also section 3.8.2: “Quick Configuration for Newport Positioners” for further information.

To create a new system configuration, define all motion groups that should belong to that system. To define a new motion group, do the following:

1. Select one Group Type (XY axes in this case): Single Axis, Spindle, XY axes, XYZ axes, or Multiple axes.
2. Click on **“Create a new group”** and the following window appears:



Manual configuration

In this page you can configure positioners and positioner groups for each supported positioner group family. You can also configure a boot script to be run at system startup.

When you're satisfied with the sys

Create a new XY axes group

Group name: XY1

Initialization sequence: Together

X mapping

File name:

Line number: 0

Column number: 0

Max position error: 0

Y mapping

File name:

Line number: 0

Column number: 0

Max position error: 0

X positioner

X positioner name: PosX

Slot: 1

Stage:

Use a 2nd positioner □

Y positioner

Y positioner name: PosY

Slot: 1

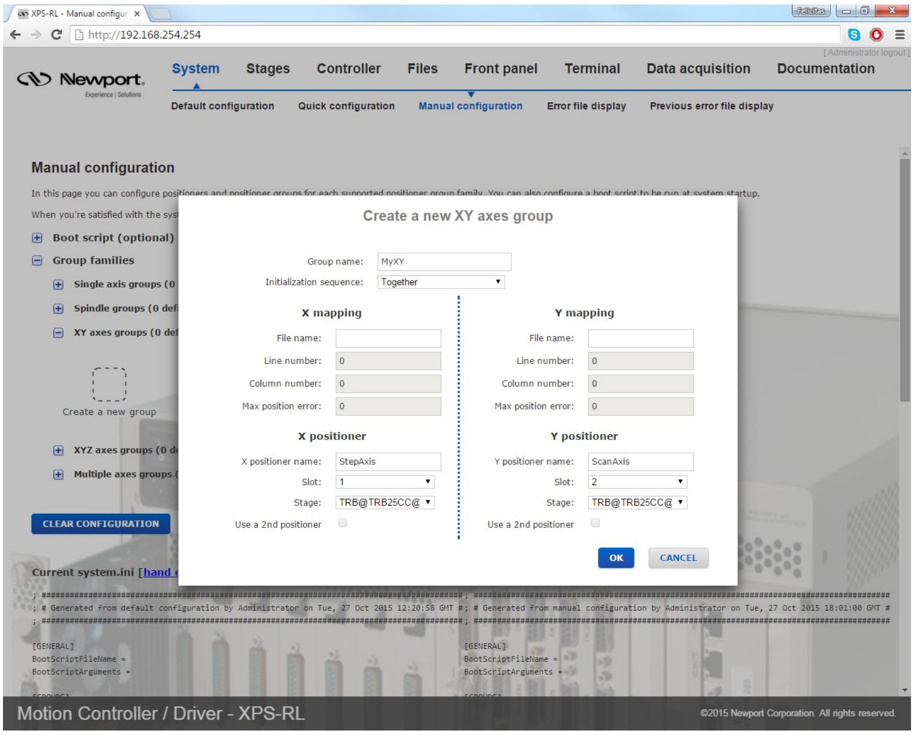
Stage:

Use a 2nd positioner □

Current system.ini [hand

```
; ##########################################################################; ##########################################################################
; # Generated from quick configuration by Administrator on Wed, 21 Oct 2015 17:09:06 GMT #; # Generated from manual configuration by Administrator on Wed, 21 Oct 2015 18:33:14 GMT #
; ##########################################################################; ##########################################################################

[GENERAL]                                                  [GENERAL]
BootScriptFileName =                                       BootScriptFileName =
BootScriptArguments =                                      BootScriptArguments =
```

3. Enter the group name in the example the group name is MyXY.
4. Define the home sequence (“Together” or “XThenY” or “YThenX”).
5. For error compensation, define the name and structure of the correction data, otherwise leave these fields blank. For details about error compensation, see chapter 10.0: “Compensation”.
6. Enter the positioner names in this example the positioner names are StepAxis and Scan Axis.
7. For each positioner select the Slot number. The Slot number is the axis number where the stage is physically connected to the XPS controller. Looking at the rear of the controller, plug number 1 is the left plug and the number increases to the right.
8. Select the stage parameters for each positioner. Before the stage parameters appear in the drop down menu, the stage configuration must be added to the stages.ini through the window Stages -> Add, remove or edit stages. See section 3.8.3: “Manual Configuration for Newport Positioners” for more information.
9. When done, click on **“OK”** to accept the configuration).



Manual configuration

In this page you can configure positioners and positioner groups for each supported positioner group family. You can also configure a boot script to be run at system startup.

When you're satisfied with the sys

Create a new XY axes group

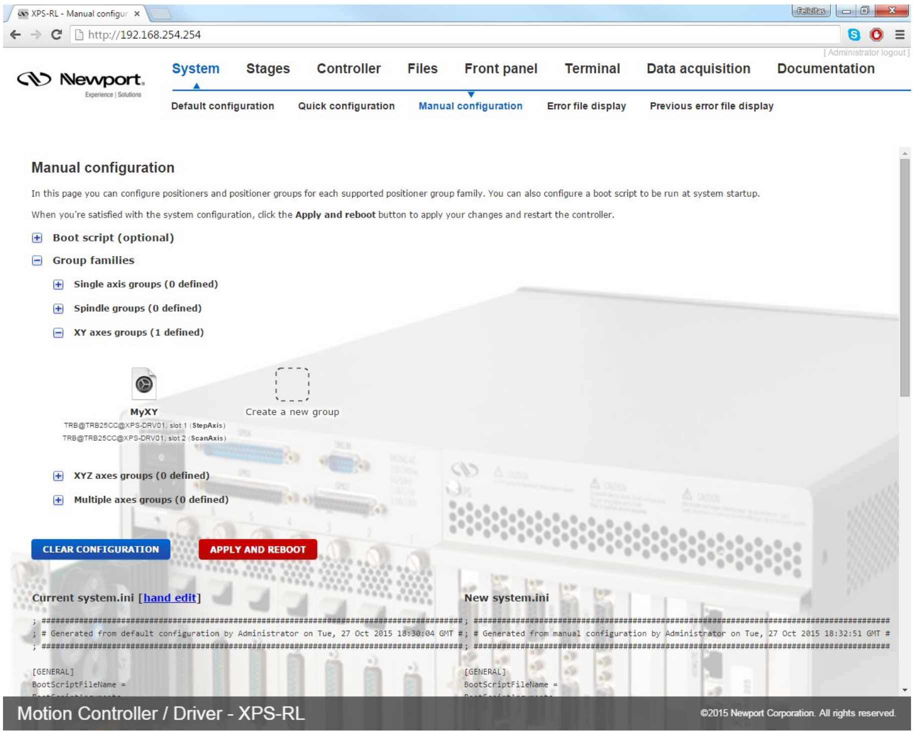
Group name: MyXY
Initialization sequence: Together

X mapping		Y mapping	
File name:		File name:	
Line number:	0	Line number:	0
Column number:	0	Column number:	0
Max position error:	0	Max position error:	0

X positioner		Y positioner	
X positioner name:	StepAxis	Y positioner name:	ScanAxis
Slot:	1	Slot:	2
Stage:	TRB@TRB25CC@	Stage:	TRB@TRB25CC@
Use a 2nd positioner	□	Use a 2nd positioner	□

```
; ##########################################################################; ##########################################################################
; # Generated from default configuration by Administrator on Tue, 27 Oct 2015 12:20:58 GMT #; # Generated from manual configuration by Administrator on Tue, 27 Oct 2015 18:01:00 GMT #
; ##########################################################################; ##########################################################################

[GENERAL]                                        [GENERAL]
BootScriptFileName =                             BootScriptFileName =
BootScriptArguments =                            BootScriptArguments =
```

10. When the configuration of each positioner is complete, the new group is listed under "New system.ini".



Manual configuration

In this page you can configure positioners and positioner groups for each supported positioner group family. You can also configure a boot script to be run at system startup.

When you're satisfied with the system configuration, click the **Apply and reboot** button to apply your changes and restart the controller.

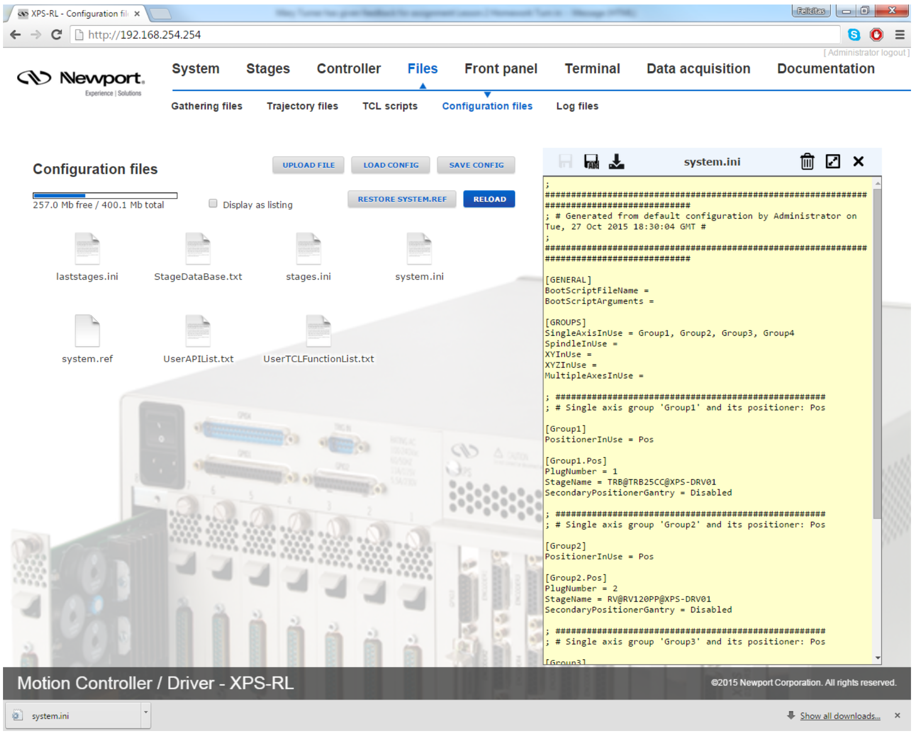
MyXY
TRB@TRB25CC@XPS-DRV01, slot 1 (StepAxis)
TRB@TRB25CC@XPS-DRV01, slot 2 (ScanAxis)

```
; ##########################################################################; ##########################################################################
; # Generated from default configuration by Administrator on Tue, 27 Oct 2015 18:30:04 GMT #; # Generated from manual configuration by Administrator on Tue, 27 Oct 2015 18:32:51 GMT #
; ##########################################################################; ##########################################################################

[GENERAL]                                        [GENERAL]
BootScriptFileName =                             BootScriptFileName =
```

11. Do the same for all other motion groups. When done, click on **"APPLY and REBOOT"** to apply the new configuration.

NOTE

"APPLY and REBOOT" deletes the current system.ini file. To create a copy of the current system.ini file, retrieve this from Files -> Configuration files. Click on the system.ini file and download from the text editor before clicking "APPLY and REBOOT".



Configuration files

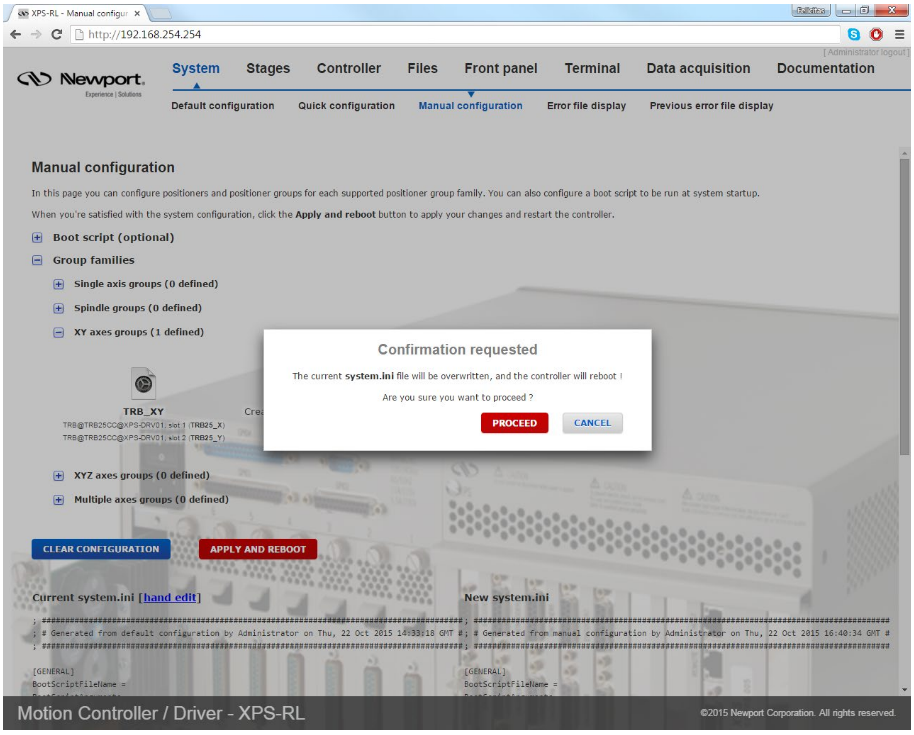
257.0 Mb free / 400.1 Mb total

□ Display as listing

system.ini

```
;
##############################################################################################
; # Generated from default configuration by Administrator on
Tue, 27 Oct 2015 18:30:04 GMT #
;
##############################################################################################

[GENERAL]
BootScriptFileName =
BootScriptArguments =

[GROUPS]
SingleAxisInUse = Group1, Group2, Group3, Group4
SpindleInUse =
XYInUse =
XYZInUse =
MultipleAxesInUse =

; ######################################################
; # Single axis group 'Group1' and its positioner: Pos

[Group1]
PositionerInUse = Pos

[Group1.Pos]
PlugNumber = 1
StageName = TRB@TRB25CC@XPS-DRV01
SecondaryPositionerGantry = Disabled

; ######################################################
; # Single axis group 'Group2' and its positioner: Pos

[Group2]
PositionerInUse = Pos

[Group2.Pos]
PlugNumber = 2
StageName = RV@RV120PP@XPS-DRV01
SecondaryPositionerGantry = Disabled

; ######################################################
; # Single axis group 'Group3' and its positioner: Pos

[Group3]
```

The following screen appears:



Manual configuration

In this page you can configure positioners and positioner groups for each supported positioner group family. You can also configure a boot script to be run at system startup.

When you're satisfied with the system configuration, click the **Apply and reboot** button to apply your changes and restart the controller.

⊞ Boot script (optional)

⊟ Group families

⊞ Single axis groups (0 defined)

⊞ Spindle groups (0 defined)

⊟ XY axes groups (1 defined)

TRB_XY
TRB@TRB25CC@XPS-DRV01, slot 1 (TRB25_X)
TRB@TRB25CC@XPS-DRV01, slot 2 (TRB25_Y)

⊞ XYZ axes groups (0 defined)

⊞ Multiple axes groups (0 defined)

Confirmation requested

The current **system.ini** file will be overwritten, and the controller will reboot !

Are you sure you want to proceed ?

Current system.ini [hand edit]

```
; ##########################################################################
; # Generated from default configuration by Administrator on Thu, 22 Oct 2015 14:33:18 GMT #
; ##########################################################################

[GENERAL]
BootScriptFileName =
```

New system.ini

```
; ##########################################################################
; # Generated from manual configuration by Administrator on Thu, 22 Oct 2015 16:40:34 GMT #
; ##########################################################################

[GENERAL]
BootScriptFileName =
```

Click on **“PROCEED”.**

12. When the controller has finished booting (one beep after 50 s–1 mn), select the **SYSTEM** tab, then **“Error File Display”**. When there is no entry in the error file, your system is configured correctly and ready to use. If not, this file provides some valuable information for troubleshooting; see also section 5.3.

This is an example of a system.ini file with one XY group and one Spindle group:

```
[GENERAL]
BootScriptFileName =
BootScriptArguments =
[GROUPS]
SingleAxisInUse =
SpindleInUse = Spin
XYInUse = My_XY_Group
XYZInUse =
MultipleAxesInUse =

 [My_XY_Group]
PositionerInUse = StepAxis,ScanAxis
InitializationAndHomeSearchSequence = Together
;--- Mapping XY
XMappingFileName =
YMappingFileName =

 [My_XY_Group.StepAxis]
PlugNumber = 3
StageName = VP-25XA-SECONDARY
[My_XY_Group.ScanAxis]
PlugNumber = 4
StageName = VP-25XA-PRIMARY

 [Spin]
PositionerInUse = Rot

 [Spin.Rot]
PlugNumber = 2
StageName = URS100CC_Spindle
```

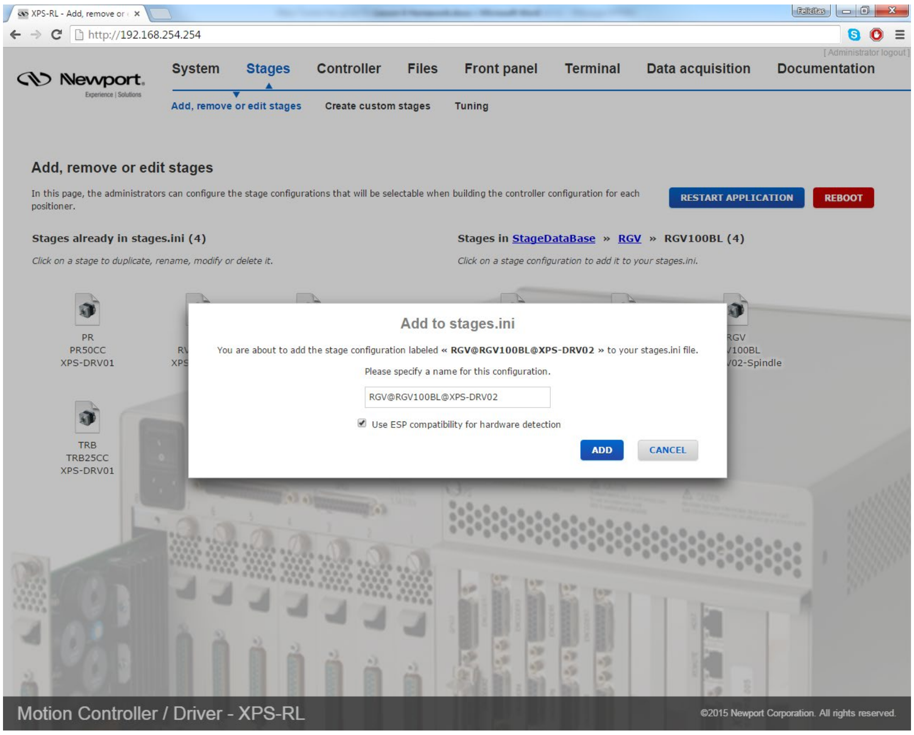
4.14 Stage – Add, Remove or Edit Stages

With the help of this screen, a stage from the Newport stage data base can be added to or removed from the personal stage data base, called stages.ini, as well as modified. On the left side of the screen, you can review the name of the stages that are already in stages.ini file. To add a new stage, do the following:

1. Click to select a family name from the list under **"Stages in StageDataBase"**.
2. Click to select the part number corresponding to your hardware.
3. Select the driver (corresponding to your hardware) and group configuration. A window appears and click **"ADD".**

   For all continuous rotation stages, you can choose between a "regular" stage configuration and a "Spindle" configuration. A Spindle is a specific rotary device with a periodic position reset at 360°, meaning 360° equals 0°. When defining the stage as Spindle in the stages.ini, you must assign this stage also to a Spindle group in the system configuration and vice versa. For details about Spindles, please refer also to section 6.3.

   For some stages, you can choose between "regular" initialization or LMI (Large Move Initialization). The LMI method produces a larger movement of the stage for commutation and could be used if "regular" initialization fails.
4. The box "Use ESP Compatibility for Hardware detection" is checked by default. If the stage has an ESP chip (a blue ESP-compatible sticker is on the stage) this box shall remain checked. Otherwise, with vacuum compatible stages or with old Newport stages, uncheck this box.



Add, remove or edit stages

In this page, the administrators can configure the stage configurations that will be selectable when building the controller configuration for each positioner.

Stages already in stages.ini (4)

Click on a stage to duplicate, rename, modify or delete it.

PR PR50CC XPS-DRV01

TRB TRB25CC XPS-DRV01

Stages in StageDataBase » RGV » RGV100BL (4)

Click on a stage configuration to add it to your stages.ini.

Add to stages.ini

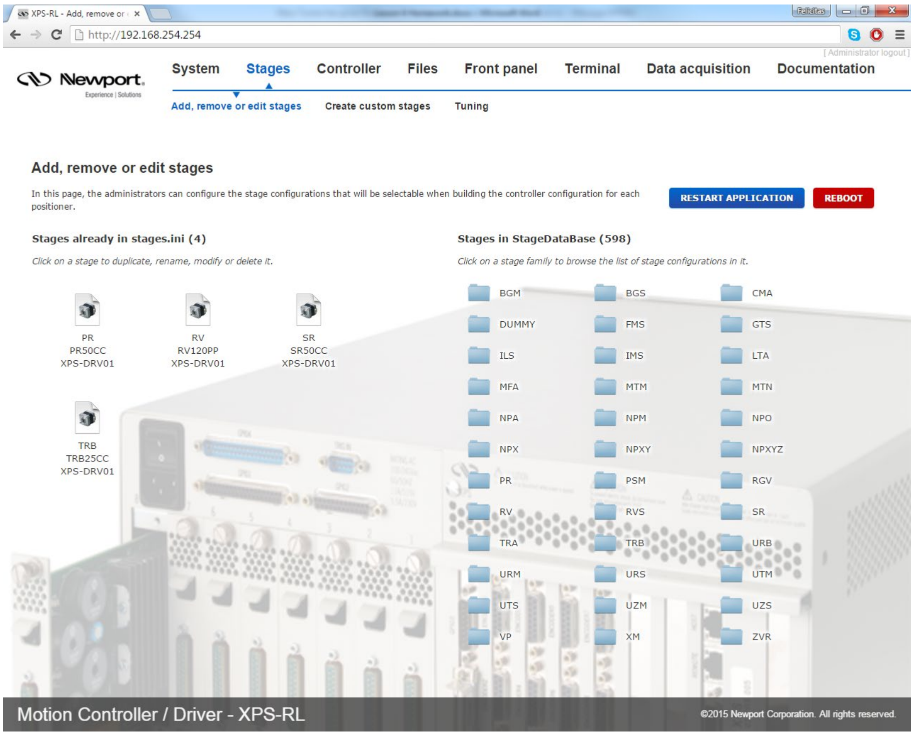
You are about to add the stage configuration labeled « **RGV@RGV100BL@XPS-DRV02** » to your stages.ini file.

Please specify a name for this configuration.

RGV@RGV100BL@XPS-DRV02

☑ Use ESP compatibility for hardware detection

5. Click **“ADD”** to add the stage to the stages.ini file.
6. Once the stage name appears under **“Stages already in stages.ini”**, it can be modified by clicking on the stage name. A window opens that allows the user to duplicate, modify or delete the stage from the stages.ini. The default name is the Newport part number, but in some cases it makes sense to use a different name. This way, for instance, it is possible to add the same set of parameters several times in the stage data base under different stage names. Later, modifying certain parameters, like travel ranges or PID settings, to optimize the stage for different applications becomes straightforward.



Add, remove or edit stages

In this page, the administrators can configure the stage configurations that will be selectable when building the controller configuration for each positioner.

Stages already in stages.ini (4)

Click on a stage to duplicate, rename, modify or delete it.

PR PR50CC XPS-DRV01

RV RV120PP XPS-DRV01

SR SR50CC XPS-DRV01

TRB TRB25CC XPS-DRV01

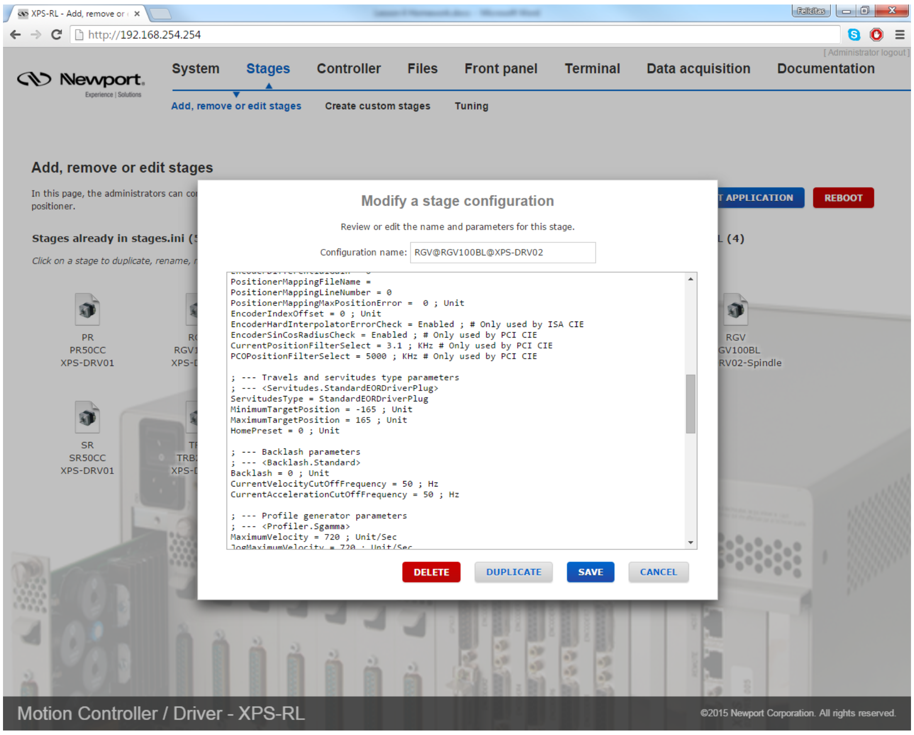
Stages in StageDataBase (598)

Click on a stage family to browse the list of stage configurations in it.

BGM	BGS	CMA
DUMMY	FMS	GTS
ILS	IMS	LTA
MFA	MTM	MTN
NPA	NPM	NPO
NPX	NPXY	NPXYZ
PR	PSM	RGV
RV	RVS	SR
TRA	TRB	URB
URM	URS	UTM
UTS	UZM	UZS
VP	XM	ZVR

To modify the parameters of a stage, do the following:

1. Click on a stage from the list under **“Stages already in stages.ini”.** A window appears which allows the user to modify the stage in the stages.ini file.
2. Scroll down to the section that contains the parameters that will be modified. Parameters commonly changed, are the minimum and the maximum target positions of a rotation stage. For example, to enable larger rotations of a rotation stage that is not configured as a Spindle, set the maximum target position to a very high value and the minimum target position to a very low value. In this case it is also required to disable the limit switches of the rotation stage, see stage manual for details.



Review or edit the name and parameters for this stage.

Configuration name: RGV@RGV100BL@XPS-DRV02

```
PositionerMappingFileName =
PositionerMappingLineNumber = 0
PositionerMappingMaxPositionError =  0 ; Unit
EncoderIndexOffset = 0 ; Unit
EncoderHardInterpolatorErrorCheck = Enabled ; # Only used by ISA CIE
EncoderSinCosRadiusCheck = Enabled ; # Only used by PCI CIE
CurrentPositionFilterSelect = 3.1 ; KHz # Only used by PCI CIE
PCOPositionFilterSelect = 5000 ; KHz # Only used by PCI CIE

; --- Travels and servitudes type parameters
; --- <Servitudes.StandardEORDriverPlug>
ServitudesType = StandardEORDriverPlug
MinimumTargetPosition = -165 ; Unit
MaximumTargetPosition = 165 ; Unit
HomePreset = 0 ; Unit

; --- Backlash parameters
; --- <Backlash.Standard>
Backlash = 0 ; Unit
CurrentVelocityCutOffFrequency = 50 ; Hz
CurrentAccelerationCutOffFrequency = 50 ; Hz

; --- Profile generator parameters
; --- <Profiler.Sgamma>
MaximumVelocity = 720 ; Unit/Sec
JogMaximumVelocity = 720 ; Unit/Sec
```

3. When done, click "Save" to apply the new values, or click “Cancel” if a mistake was made.
4. To take the new values into account, reboot the controller or use “Restart Application” button.

The same screen allows duplicating stages in the stages.ini (in most case some parameters are modified as a second step) or to delete stages from the stages.ini.

4.15 Stages – Create Custom Stages

This web page is used to build stage configuration files for stages not found in the controller's StageDataBase.txt such as non-Newport stages. Refer to chapter 23.0: "Appendix F: Configuration Wizard" for more information.

4.16 Stages – Tuning

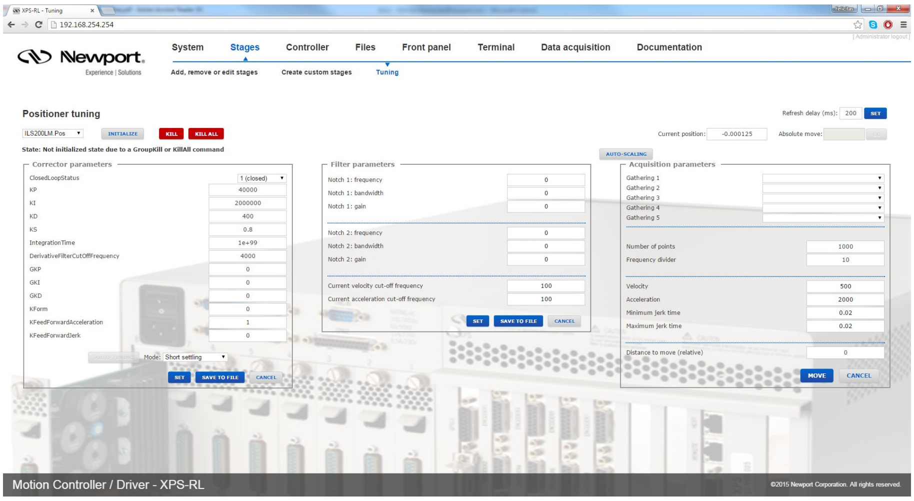
4.16.1 Tuning – Auto-Scaling

Auto-scaling is only available with positioners that feature a direct drive motor such as the XM, ILS-LM, IMS-LM or RGV100BL. To guarantee consistent performance of these stages, it is strongly recommended to perform Auto-scaling once the load is attached to the stage. During auto-scaling, the XPS controller measures the mass (inertia for rotation stages) on the positioner and returns recommended values for the Scaling Acceleration parameter.

Repeat Auto-scaling with any major change of the payload on the positioner. With no major change of the payload, there is no need to redo Auto-scaling.

To perform Auto-scaling, do the following:

1. Select the main tab TUNING. Then select a positioner name. The following screen appears:



Positioner tuning

ILS200LM.Pos

State: Not initialized state due to a GroupKill or KillAll command

Refresh delay (ms): 200

Current position: -0.000125

Absolute move:

Corrector parameters

ClosedLoopStatus	1 (closed)
KP	40000
KI	2000000
KD	400
KS	0.8
IntegrationTime	1e+99
DerivativeFilterCutOffFrequency	4000
GKP	0
GKI	0
GKD	0
KForm	0
KFeedForwardAcceleration	1
KFeedForwardJerk	0

Mode: Short settling

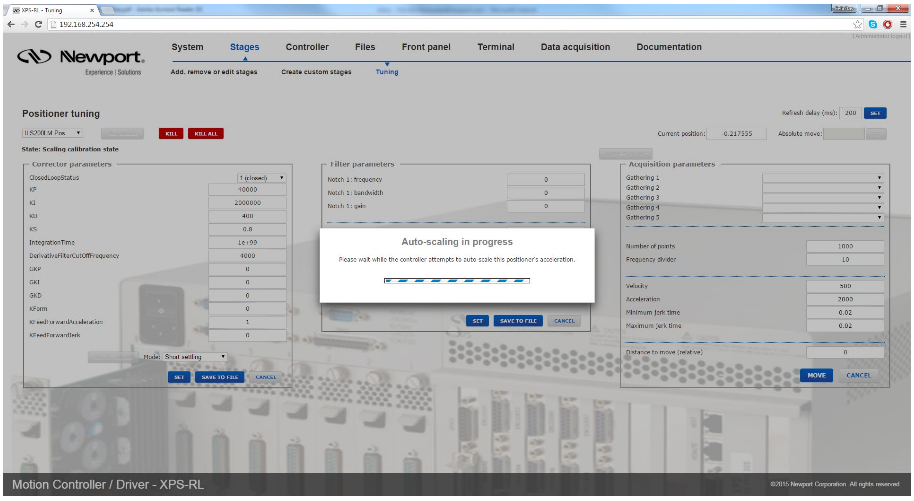
Filter parameters

Notch 1: frequency	0
Notch 1: bandwidth	0
Notch 1: gain	0
Notch 2: frequency	0
Notch 2: bandwidth	0
Notch 2: gain	0
Current velocity cut-off frequency	100
Current acceleration cut-off frequency	100

Acquisition parameters

Gathering 1	
Gathering 2	
Gathering 3	
Gathering 4	
Gathering 5	
Number of points	1000
Frequency divider	10
Velocity	500
Acceleration	2000
Minimum jerk time	0.02
Maximum jerk time	0.02
Distance to move (relative)	0

2. Click “Kill group”, then click “Auto-scaling”. The stage vibrates and an auto-scaling progress bar appears.



Positioner tuning

ILS200LM.Pos

State: Scaling calibration state

Refresh delay (ms): 200

Current position: -0.217555

Absolute move:

Auto-scaling in progress

Please wait while the controller attempts to auto-scale this positioner's acceleration.

Corrector parameters

ClosedLoopStatus	1 (closed)
KP	40000
KI	2000000
KD	400
KS	0.8
IntegrationTime	1e+99
DerivativeFilterCutOffFrequency	4000
GKP	0
GKI	0
GKD	0
KForm	0
KFeedForwardAcceleration	1
KFeedForwardJerk	0

Mode: Short settling

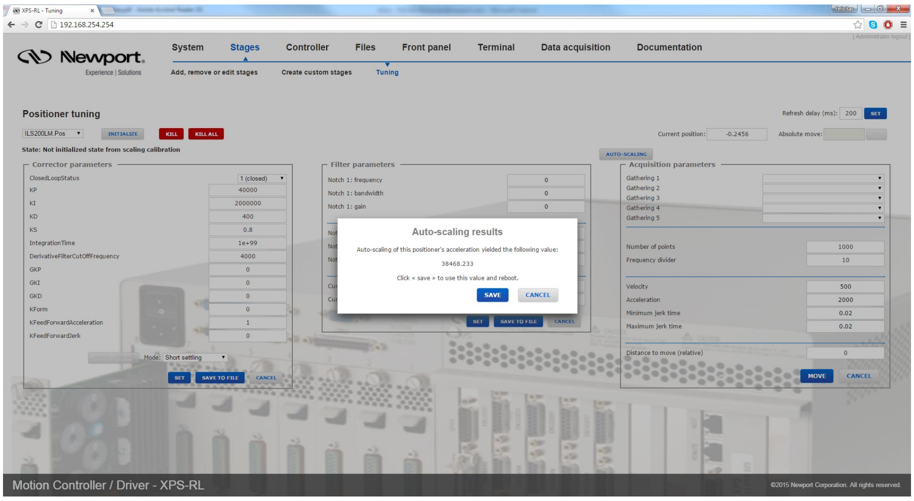
Filter parameters

Notch 1: frequency	0
Notch 1: bandwidth	0
Notch 1: gain	0

Acquisition parameters

Gathering 1	
Gathering 2	
Gathering 3	
Gathering 4	
Gathering 5	
Number of points	1000
Frequency divider	10
Velocity	500
Acceleration	2000
Minimum jerk time	0.02
Maximum jerk time	0.02
Distance to move (relative)	0

3. When the auto-scaling routine is complete, the results are displayed. To save the recommended values and reboot the controller, click “Save”. The positioner should now work properly.



Positioner tuning

ILS200LM.Pos

State: Not initialized state from scaling calibration

Refresh delay (ms): 200

Current position: -0.2456   Absolute move:

Corrector parameters

ClosedLoopStatus	1 (closed)
KP	40000
KI	2000000
KD	400
KS	0.8
IntegrationTime	1e+99
DerivativeFilterCutOffFrequency	4000
GKP	0
GKI	0
GKD	0
KForm	0
KFeedForwardAcceleration	1
KFeedForwardJerk	0

Mode: Short settling

Filter parameters

Notch 1: frequency	0
Notch 1: bandwidth	0
Notch 1: gain	0

Auto-scaling results

Auto-scaling of this positioner's acceleration yielded the following value:

38468.233

Click « save » to use this value and reboot.

SAVE CANCEL

Acquisition parameters

Gathering 1
Gathering 2
Gathering 3
Gathering 4
Gathering 5

Number of points	1000
Frequency divider	10
Velocity	500
Acceleration	2000
Minimum jerk time	0.02
Maximum jerk time	0.02
Distance to move (relative)	0

NOTE

All other functions of the tuning page should be used only by experienced users.

4.16.2 Tuning – Auto-Tuning

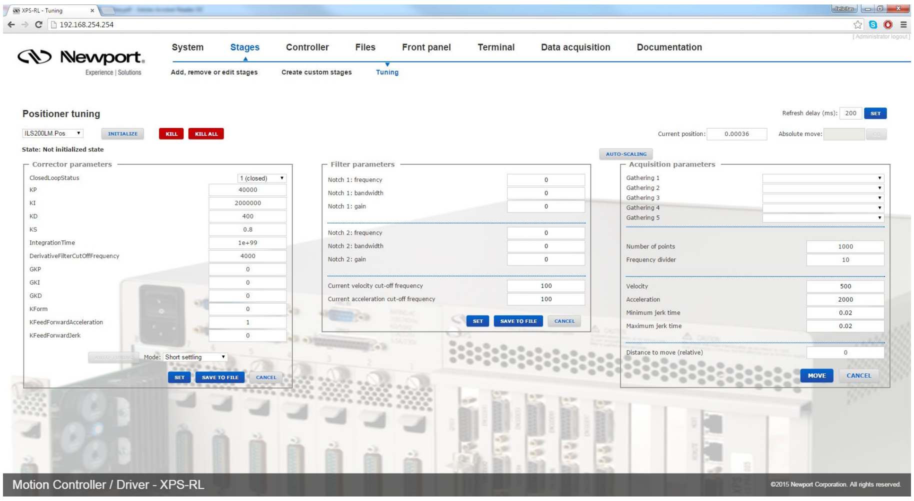
NOTE

Apart from the Auto-scaling feature, which is described in the previous chapter, only experienced motion control users should use the TUNING tool of the XPS controller.

All Newport positioners are supplied with default tuning parameters that provide consistently high performance for the vast majority of applications. Use the Tuning tool with Newport positioners only when not fully satisfied with the dynamic behavior of the positioners. Auto-Tuning works best with direct drive stages. Friction drive or ballscrew drive systems may not result in optimum tuning using this feature.

The following is a brief description of the TUNING tool:

1. Select a positioner name. The following screen appears:



Positioner tuning

ILS200LM.Pos

State: Not initialized state

Refresh delay (ms): 200

Current position: 0.00036   Absolute move:

Corrector parameters

ClosedLoopStatus	1 (closed)
KP	40000
KI	2000000
KD	400
KS	0.8
IntegrationTime	1e+99
DerivativeFilterCutOffFrequency	4000
GKP	0
GKI	0
GKD	0
KForm	0
KFeedForwardAcceleration	1
KFeedForwardJerk	0

Mode: Short settling

Filter parameters

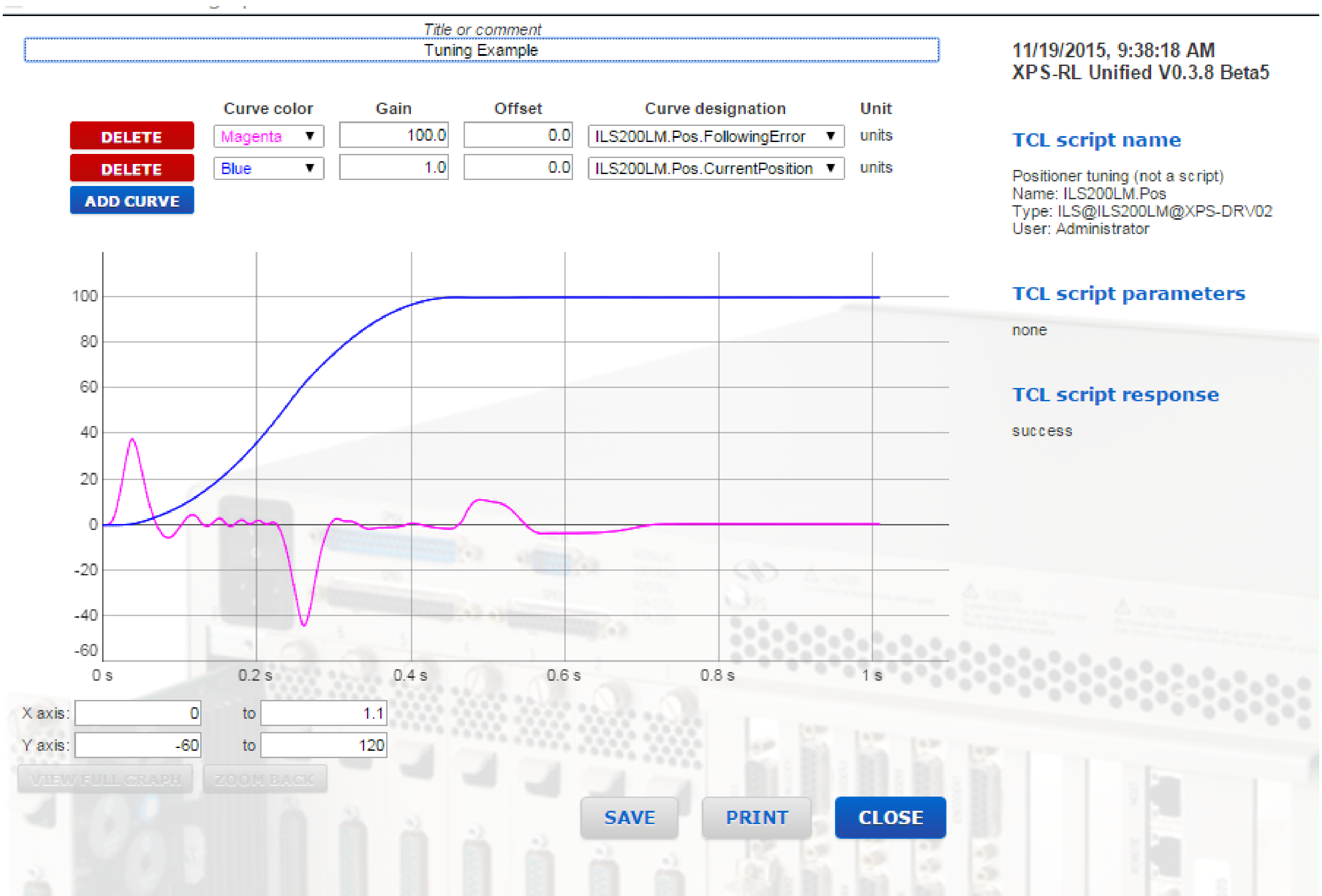
Notch 1: frequency	0
Notch 1: bandwidth	0
Notch 1: gain	0
Notch 2: frequency	0
Notch 2: bandwidth	0
Notch 2: gain	0
Current velocity cut-off frequency	100
Current acceleration cut-off frequency	100

Acquisition parameters

Gathering 1
Gathering 2
Gathering 3
Gathering 4
Gathering 5

Number of points	1000
Frequency divider	10
Velocity	500
Acceleration	2000
Minimum jerk time	0.02
Maximum jerk time	0.02
Distance to move (relative)	0

2. Perform a data gathering with your current parameter settings.
    1. Initialize and home the positioner, then move to the desired start position.
    2. Under "Acquisition parameters", define the gathering data: For the stage tuning, it is recommended to gather only the following error and the current position.
    3. Define a typical motion distance.
    4. Define the frequency divisor. The frequency divisor defines the sampling rate of the gathering. A frequency divisor equal to one means one data point is gathered every servo cycle. With most positioners, it is sufficient to set a value of 10, meaning one data point every 1.25 ms for the default servo rate of 8 kHz.
    5. Define the number of points in relation to the distance, the frequency divisor, the velocity and the acceleration.
    6. Define the velocity, acceleration and jerk time.
    7. When done, click "MOVE".



Title or comment: Tuning Example

11/19/2015, 9:38:18 AM
XPS-RL Unified V0.3.8 Beta5

	Curve color	Gain	Offset	Curve designation	Unit
DELETE	Magenta	100.0	0.0	ILS200LM.Pos.FollowingError	units
DELETE	Blue	1.0	0.0	ILS200LM.Pos.CurrentPosition	units

TCL script name

Positioner tuning (not a script)
Name: ILS200LM.Pos
Type: ILS@ILS200LM@XPS-DRV02
User: Administrator

TCL script parameters

none

TCL script response

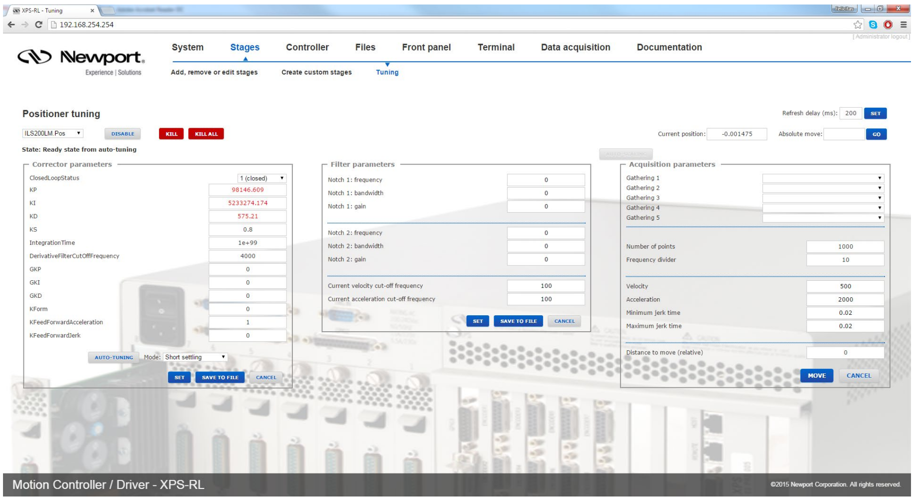
success

X axis: 0 to 1.1
Y axis: -60 to 120

3. When satisfied with the results, there is no need to tune the stage. If not satisfied, return to the tuning page and move back to the start position.

4. Next to the Auto-tuning button, there is a Mode field for Auto-tuning. Select "Short settling" or "High robustness". Choose "Short settling" to improve the settling time after a motion or to reduce the following error during the motion. Short settling will define "high" PID vales for your stage, but there is a risk of oscillation. Choose "High robustness" to improve the robustness of the motion system and to avoid oscillations during or after a motion. "High robustness", for instance, can avoid oscillations for a rotation stage with high payload inertia. When done with the selection, click Auto-tuning.

5. The stage vibrates for a couple of seconds. When done the following screen appears:



Positioner tuning

ILS200LM.Pos

State: Ready state from auto-tuning

Refresh delay (ms): 200

Current position: -0.001475 Absolute move:

Corrector parameters

ClosedLoopStatus	1 (closed)
KP	98146.609
KI	5233274.174
KD	575.21
KS	0.8
IntegrationTime	1e+99
DerivativeFilterCutOffFrequency	4000
GKP	0
GKI	0
GKD	0
KForm	0
KFeedForwardAcceleration	1
KFeedForwardJerk	0

Mode: Short settling

Filter parameters

Notch 1: frequency	0
Notch 1: bandwidth	0
Notch 1: gain	0
Notch 2: frequency	0
Notch 2: bandwidth	0
Notch 2: gain	0
Current velocity cut-off frequency	100
Current acceleration cut-off frequency	100

Acquisition parameters

Gathering 1	
Gathering 2	
Gathering 3	
Gathering 4	
Gathering 5	
Number of points	1000
Frequency divider	10
Velocity	500
Acceleration	2000
Minimum jerk time	0.02
Maximum jerk time	0.02
Distance to move (relative)	0

6. Press “Set” to apply the new parameters. “Set” only changes the working parameters during data gathering. Recover the previous parameters by rebooting the system.

7. To test the behavior of the motion system with the new parameters, repeat the same data gathering and compare the results. Make manual changes to the settings and verify the behavior.

8. To permanently save the settings to the stages.ini, press “SAVE TO FILE”. “SAVE TO FILE” overwrites the current settings in your stages.ini. Press “SAVE TO FILE” only when fully satisfied with the results. For recovery, Newport recommends making a copy of the stages.ini with the old settings.

NOTE

For further information about the meaning of the different tuning parameters, see chapter 14.0.

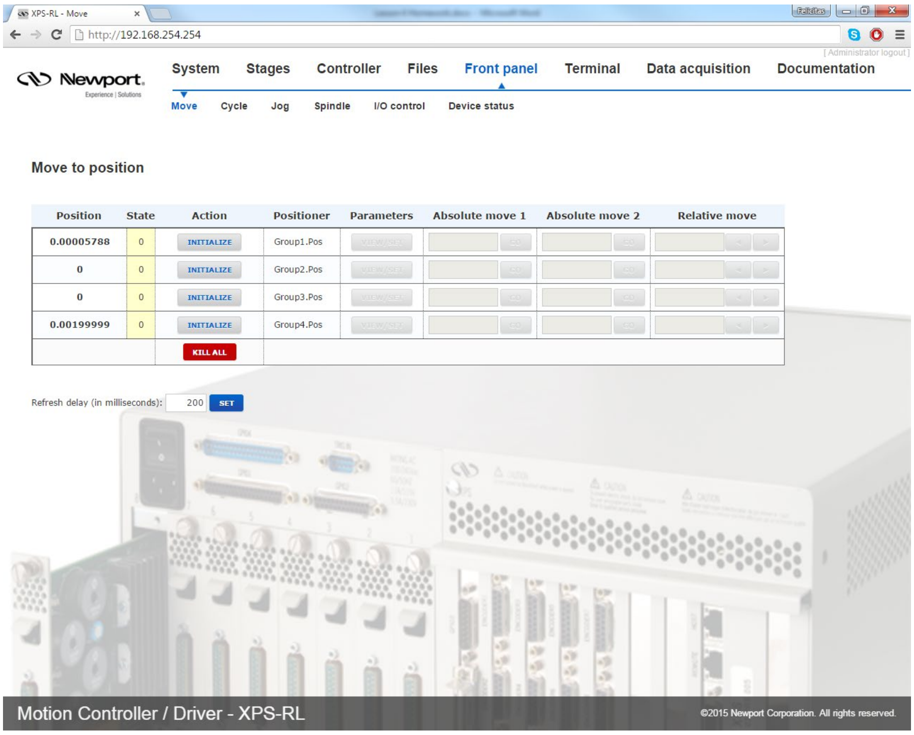
4.17 Front Panel – Move

The Move page provides access to basic group functions like initialize, home, or motor disable, and executes relative and absolute moves where speed, acceleration and jerk time can be modified during motion (but not during the acceleration period).

The Move page also provides a convenient review of all important group information like group names, group states and positions. All motion groups are listed in the Move page.

NOTE

A spindle group can do relative moves and absolute moves. So it can be used in the Move page. See section 4.14 for more information about Spindle moves.



Move to position

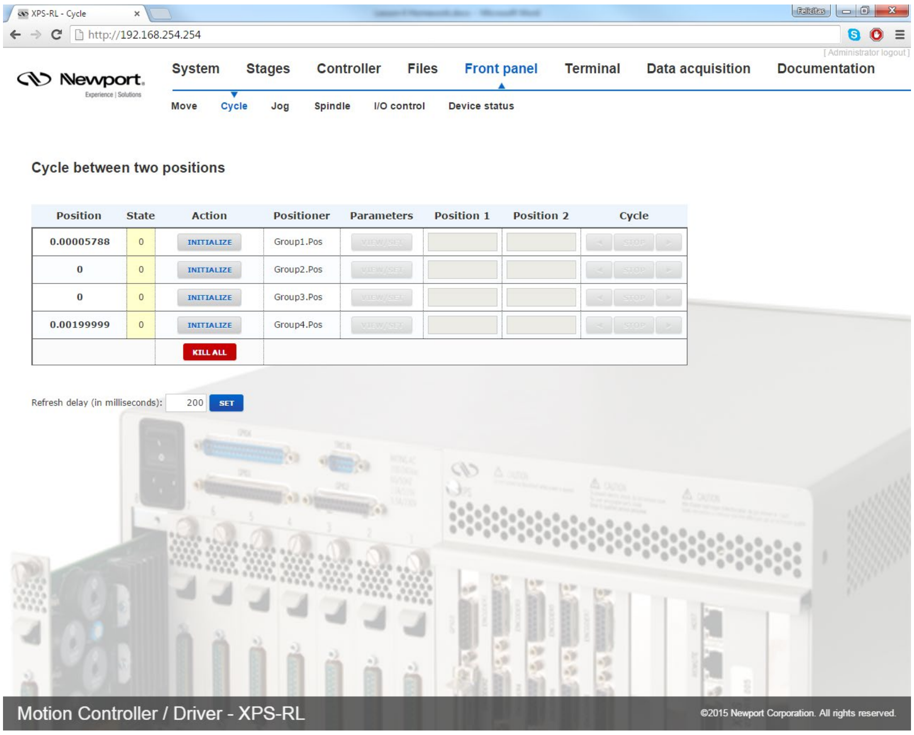
Position	State	Action	Positioner	Parameters	Absolute move 1	Absolute move 2	Relative move
0.00005788	0	INITIALIZE	Group1.Pos	VIEW/SET	GO	GO	< >
0	0	INITIALIZE	Group2.Pos	VIEW/SET	GO	GO	< >
0	0	INITIALIZE	Group3.Pos	VIEW/SET	GO	GO	< >
0.00199999	0	INITIALIZE	Group4.Pos	VIEW/SET	GO	GO	< >
		KILL ALL					

Refresh delay (in milliseconds): 200 SET

Motion Controller / Driver - XPS-RL
©2015 Newport Corporation. All rights reserved.

4.18 Front Panel – Cycle

The cycle page allows cycling of a stage. A cycle motion move back and forth between two use defined positions where speed, acceleration and jerk time can be modified during motion (but not during the acceleration period).



Cycle between two positions

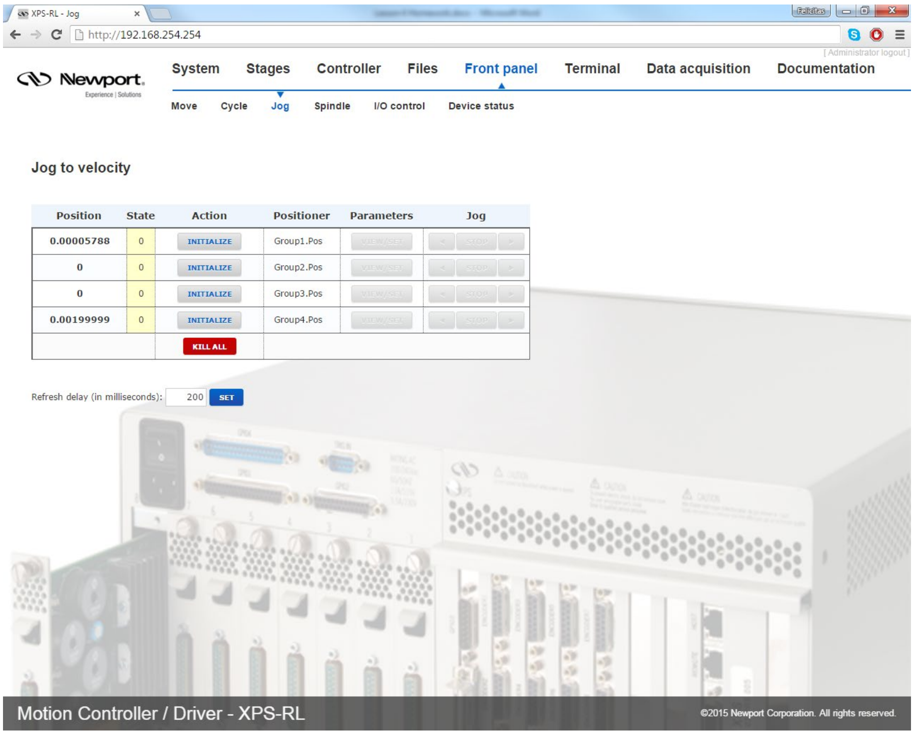
Position	State	Action	Positioner	Parameters	Position 1	Position 2	Cycle
0.00005788	0	INITIALIZE	Group1.Pos	VIEW/SET			< STOP >
0	0	INITIALIZE	Group2.Pos	VIEW/SET			< STOP >
0	0	INITIALIZE	Group3.Pos	VIEW/SET			< STOP >
0.00199999	0	INITIALIZE	Group4.Pos	VIEW/SET			< STOP >
		KILL ALL					

Refresh delay (in milliseconds): 200 SET

4.19 Front Panel – Jog

The Jog page allows executing a jog motion. A jog motion is a continuous motion, where only the speed and acceleration are defined, but no target position. Speed and acceleration can be changed during the motion (but not during the acceleration period).

For a Jog motion, the jog mode must be enabled, see “Action” button.



Jog to velocity

Position	State	Action	Positioner	Parameters	Jog
0.00005788	0	INITIALIZE	Group1.Pos	VIEW/SET	< STOP >
0	0	INITIALIZE	Group2.Pos	VIEW/SET	< STOP >
0	0	INITIALIZE	Group3.Pos	VIEW/SET	< STOP >
0.00199999	0	INITIALIZE	Group4.Pos	VIEW/SET	< STOP >
		KILL ALL			

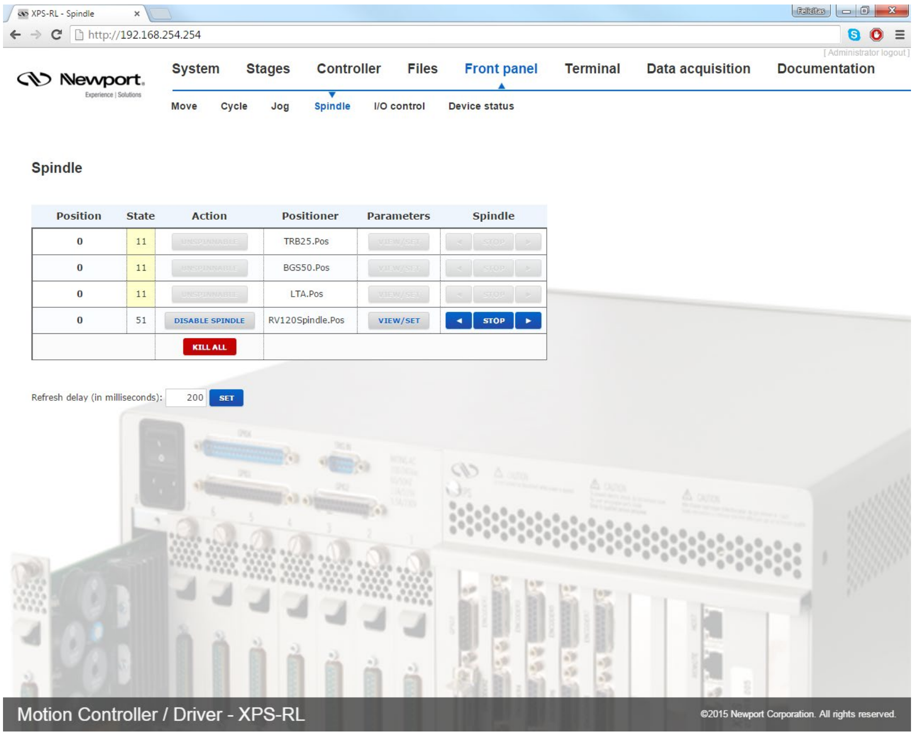
Refresh delay (in milliseconds): 200 SET

4.20 Front Panel – Spindle

The Spindle page provides similar functions to the Jog page. However, specific jog actions are replaced by spindle actions that only work for Spindle groups.

NOTE

Spindle configuration does not allow indexing of the rotary stage. For a rotary stage to have indexing and move more than 360 degrees, the user must configure the stage as a group type other than spindle and change the software travel limits, MinimumTargetPosition and MaximumTargetPosition, in the stages.ini file. Refer to section 4.14: “Stage – Add, Remove or Edit Stages” for more details. Additionally, the rotary may need to have optical travel limits disabled. Refer to the rotary stage User's Manual.



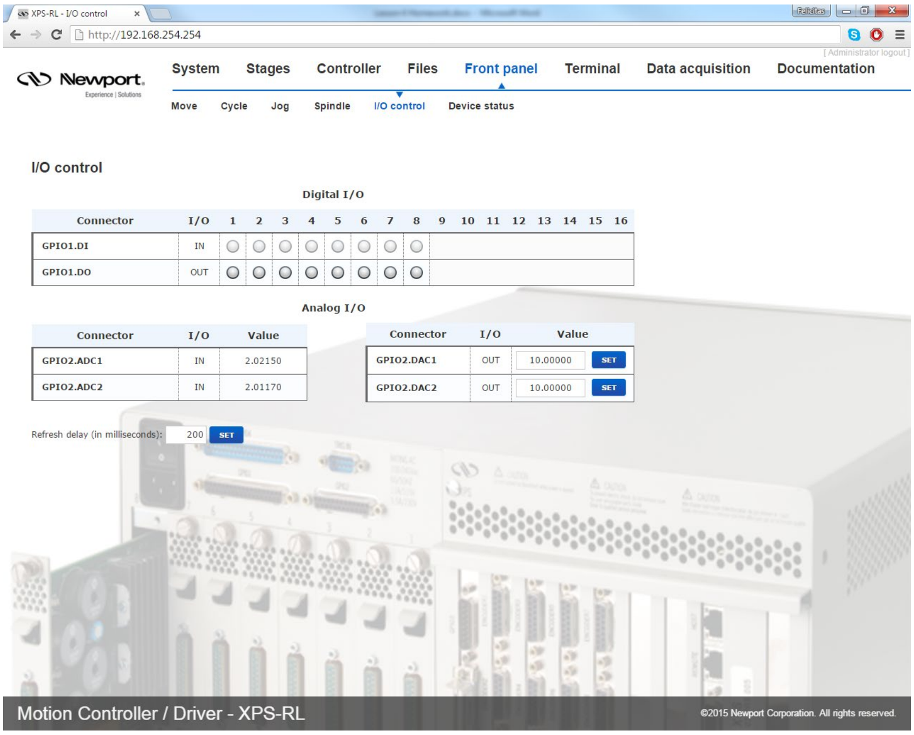
Spindle

Position	State	Action	Positioner	Parameters	Spindle
0	11	UNSPINNABLE	TRB25.Pos	VIEW/SET	◄ STOP ►
0	11	UNSPINNABLE	BGS50.Pos	VIEW/SET	◄ STOP ►
0	11	UNSPINNABLE	LTA.Pos	VIEW/SET	◄ STOP ►
0	51	DISABLE SPINDLE	RV120Spindle.Pos	VIEW/SET	◄ STOP ►
		KILL ALL			

Refresh delay (in milliseconds): 200 SET

4.21 Front Panel – I/O Control

The I/O Control page shows the current states or values of all analog and all digital I/O’s of the controller and allows the user to set all the analog and digital outputs of the controller.



I/O control

Digital I/O

Connector	I/O	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
GPIO1.DI	IN	○	○	○	○	○	○	○	○								
GPIO1.DO	OUT	○	○	○	○	○	○	○	○								

Analog I/O

Connector	I/O	Value
GPIO2.ADC1	IN	2.02150
GPIO2.ADC2	IN	2.01170

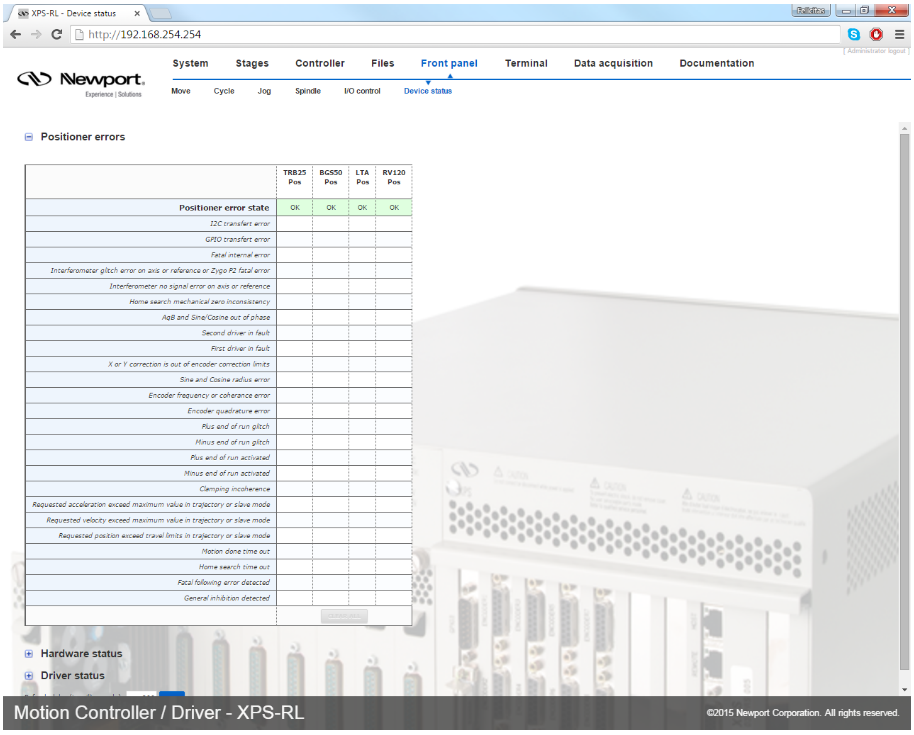
Connector	I/O	Value
GPIO2.DAC1	OUT	10.00000 SET
GPIO2.DAC2	OUT	10.00000 SET

Refresh delay (in milliseconds): 200 SET

4.22 Front Panel – Device Status

4.22.1 Device Status – Positioner Errors

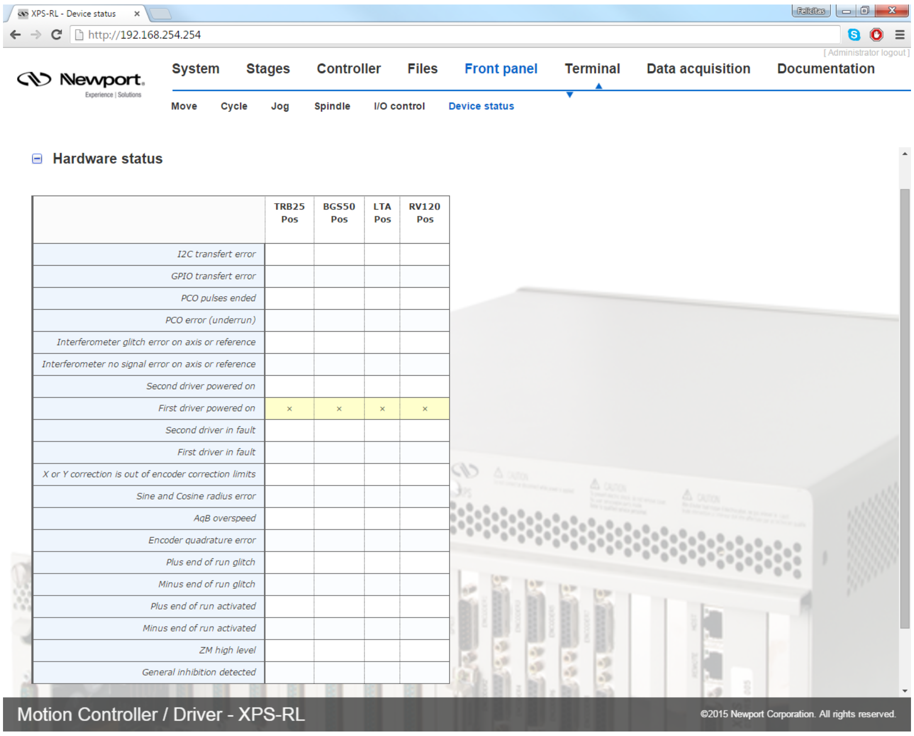
The Positioner Errors page is an important page for trouble-shooting. When encountering any problems during the use of the system, information about the errors related to the positioners are found in this page.



	TRB25 Pos	BGS50 Pos	LTA Pos	RV120 Pos
Positioner error state	OK	OK	OK	OK
I2C transfert error				
GPIO transfert error				
Fatal internal error				
Interferometer glitch error on axis or reference or Zygo P2 fatal error				
Interferometer no signal error on axis or reference				
Home search mechanical zero inconsistency				
AqB and Sine/Cosine out of phase				
Second driver in fault				
First driver in fault				
X or Y correction is out of encoder correction limits				
Sine and Cosine radius error				
Encoder frequency or coherance error				
Encoder quadrature error				
Plus end of run glitch				
Minus end of run glitch				
Plus end of run activated				
Minus end of run activated				
Clamping incoherence				
Requested acceleration exceed maximum value in trajectory or slave mode				
Requested velocity exceed maximum value in trajectory or slave mode				
Requested position exceed travel limits in trajectory or slave mode				
Motion done time out				
Home search time out				
Fatal following error detected				
General inhibition detected				

4.22.2 Device Status – Hardware Status

The Hardware Status page is another important page for trouble-shooting, but not all information is related to an error.

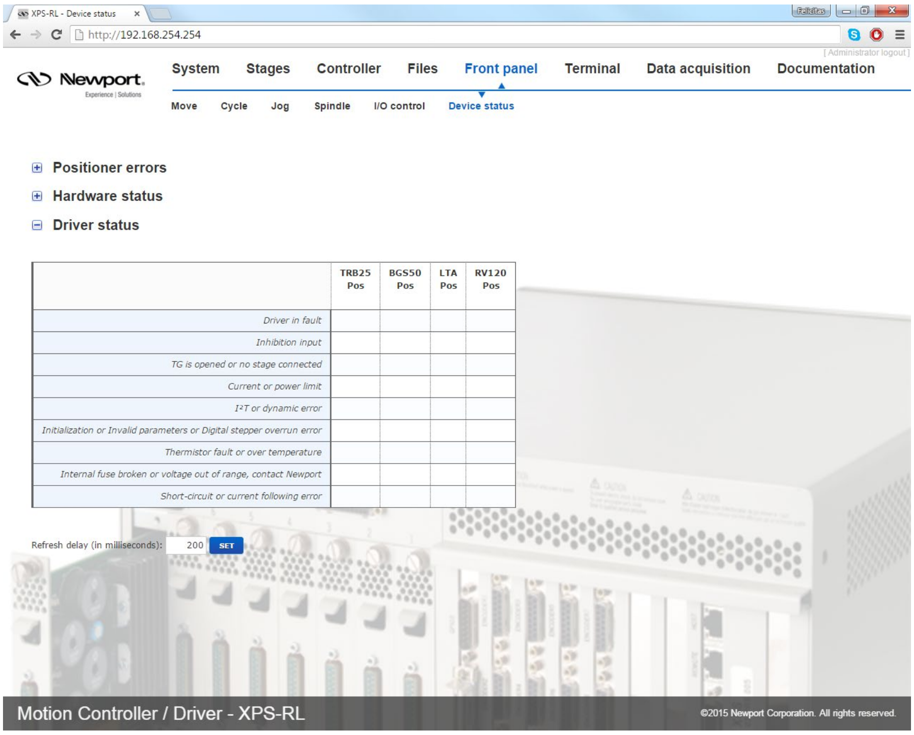


	TRB25 Pos	BGS50 Pos	LTA Pos	RV120 Pos
I2C transfert error				
GPIO transfert error				
PCO pulses ended				
PCO error (underrun)				
Interferometer glitch error on axis or reference				
Interferometer no signal error on axis or reference				
Second driver powered on				
First driver powered on	×	×	×	×
Second driver in fault				
First driver in fault				
X or Y correction is out of encoder correction limits				
Sine and Cosine radius error				
AqB overspeed				
Encoder quadrature error				
Plus end of run glitch				
Minus end of run glitch				
Plus end of run activated				
Minus end of run activated				
ZM high level				
General inhibition detected				

4.22.3 Device Status – Driver status

The Driver Status page is another important page for trouble-shooting, but not all information is related to an error.

The type of status information that you can get depends on the drivers used.

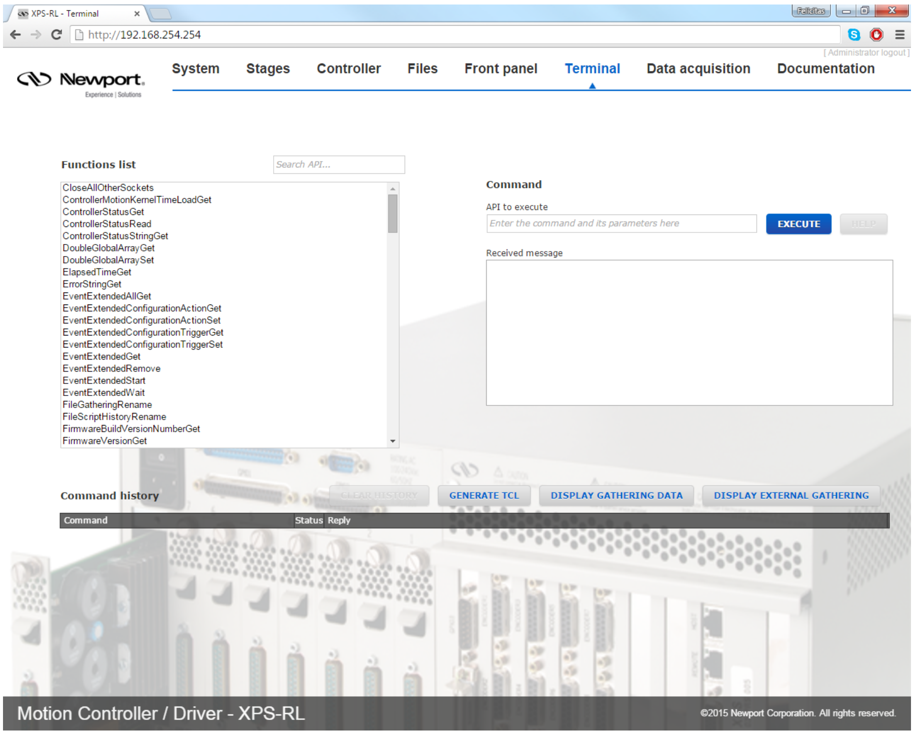


	TRB25 Pos	BGS50 Pos	LTA Pos	RV120 Pos
Driver in fault				
Inhibition input				
TG is opened or no stage connected				
Current or power limit				
I^2T or dynamic error				
Initialization or Invalid parameters or Digital stepper overrun error				
Thermistor fault or over temperature				
Internal fuse broken or voltage out of range, contact Newport				
Short-circuit or current following error				

Refresh delay (in milliseconds): 200

4.23 Terminal

The Terminal screen allows the execution of all XPS controller functions. It also provides a convenient method for generating executable TCL scripts. For more details about TCL scripts, see chapter 17.0.



Functions list

CloseAllOtherSockets
ControllerMotionKernelTimeLoadGet
ControllerStatusGet
ControllerStatusRead
ControllerStatusStringGet
DoubleGlobalArrayGet
DoubleGlobalArraySet
ElapsedTimeGet
ErrorStringGet
EventExtendedAllGet
EventExtendedConfigurationActionGet
EventExtendedConfigurationActionSet
EventExtendedConfigurationTriggerGet
EventExtendedConfigurationTriggerSet
EventExtendedGet
EventExtendedRemove
EventExtendedStart
EventExtendedWait
FileGatheringRename
FileScriptHistoryRename
FirmwareBuildVersionNumberGet
FirmwareVersionGet

Command

API to execute: Enter the command and its parameters here

Received message

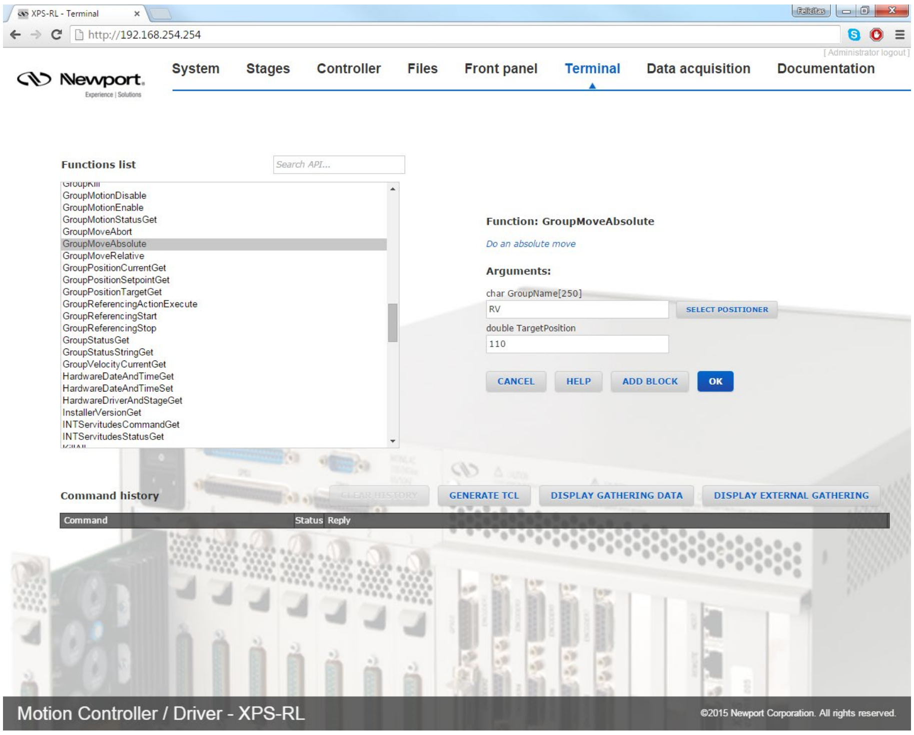
Command history

Command	Status	Reply

To execute a function from the Terminal, do the following:

1. Click to select a function, which then appears in the "API to execute" window.
2. Define the arguments for the function.

   For functions with dynamic arguments “ADD” and “REMOVE” buttons are available. Alternatively, you can use a “,” as a separator between different arguments.



Functions list

GroupKill
GroupMotionDisable
GroupMotionEnable
GroupMotionStatusGet
GroupMoveAbort
GroupMoveAbsolute
GroupMoveRelative
GroupPositionCurrentGet
GroupPositionSetpointGet
GroupPositionTargetGet
GroupReferencingActionExecute
GroupReferencingStart
GroupReferencingStop
GroupStatusGet
GroupStatusStringGet
GroupVelocityCurrentGet
HardwareDateAndTimeGet
HardwareDateAndTimeSet
HardwareDriverAndStageGet
InstallerVersionGet
INTServitudesCommandGet
INTServitudesStatusGet
KillAll

Function: GroupMoveAbsolute

Do an absolute move

Arguments:

char GroupName[250]: RV

double TargetPosition: 110

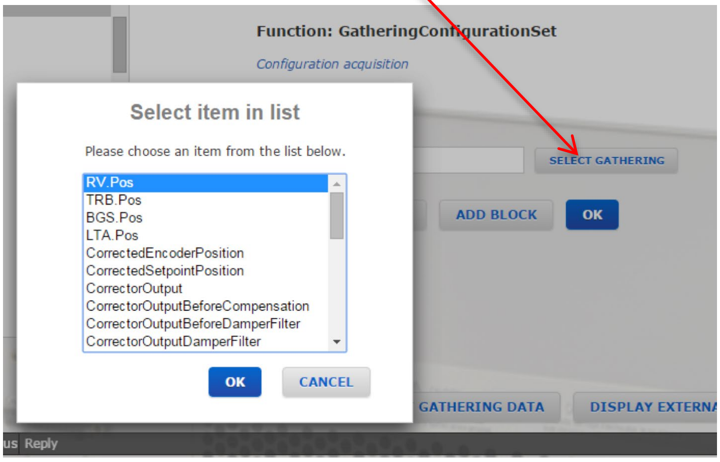
Command history

Command	Status	Reply

For some arguments like ExtendedEventName, ExtendedActionName or GatheringType, the argument name is not directly accessible. In these cases, define the first part of the argument name, then click in the field again and define the second part of the argument name. See the example below for defining the GatheringType with the function GatheringConfigurationSet():

Step 1:

Click **"SELECT GATHERING"** then select the positioner name and click **"OK"**.


Function: GatheringConfigurationSet

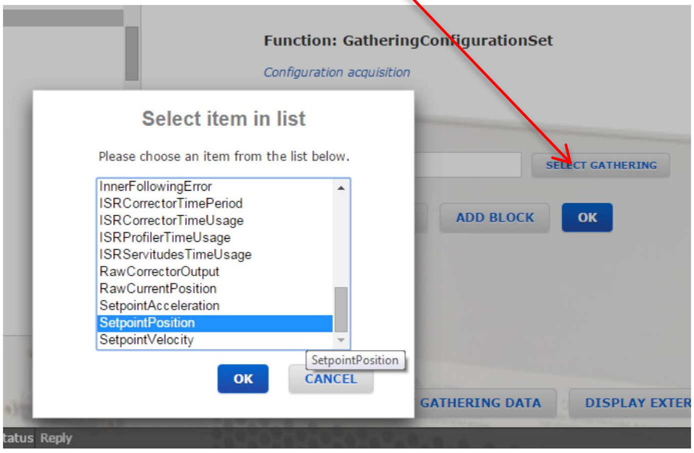
Configuration acquisition

Select item in list

Please choose an item from the list below.

Step 2:

Click **"SELECT GATHERING"** again then select the parameter name and click **"OK"**.


Function: GatheringConfigurationSet

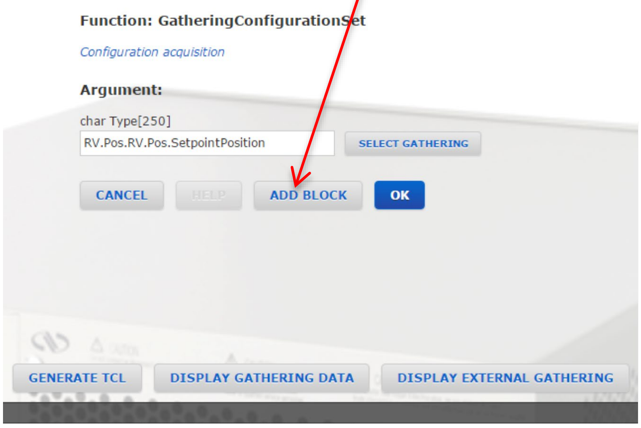
Configuration acquisition

Select item in list

Please choose an item from the list below.

Step 3:

To add another parameter, click **"ADD BLOCK".** Repeat Step 1 and Step 2.


Function: GatheringConfigurationSet

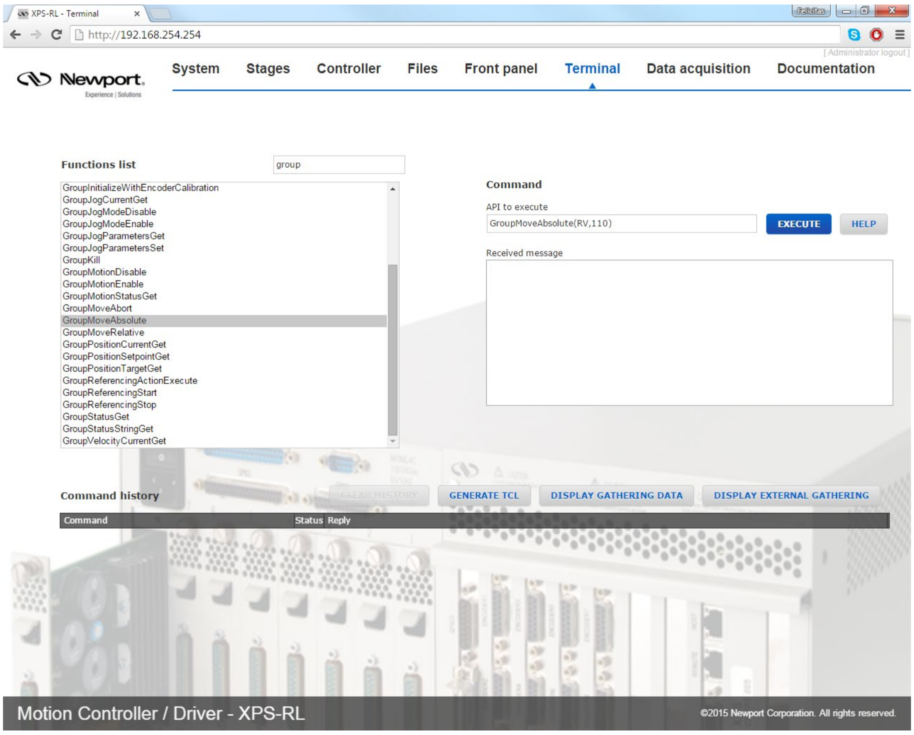
Configuration acquisition

Argument:

char Type[250]

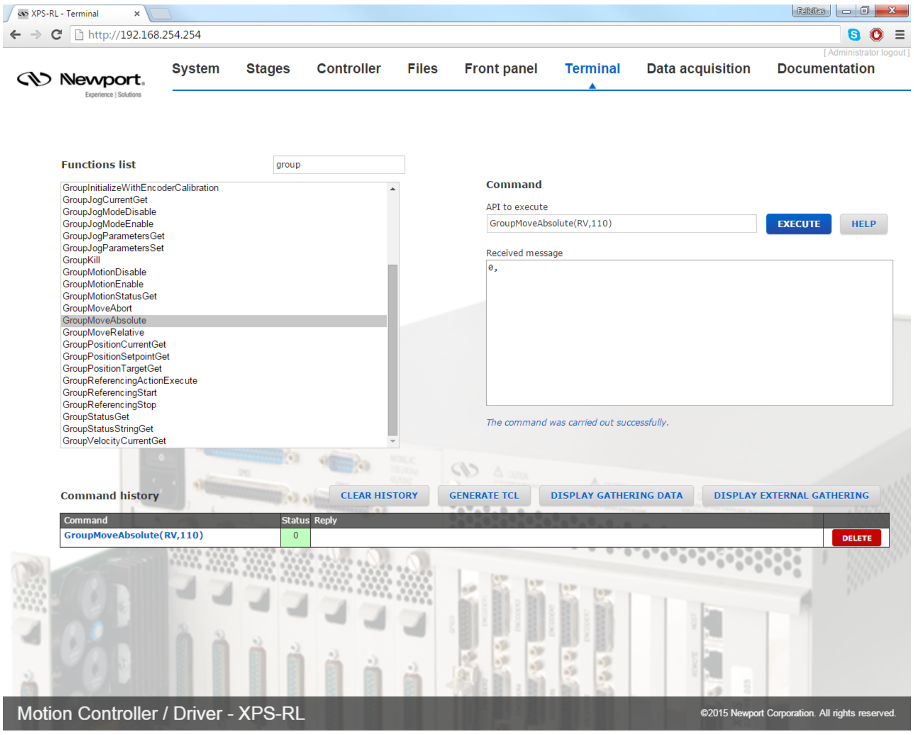
RV.Pos.RV.Pos.SetpointPosition

3. When all arguments are defined, click “OK”. Now review the final syntax of the function and make final text changes, as needed. When done, click “Execute”.



API to execute: GroupMoveAbsolute(RV,110)

4. When the function is executed, the controller’s response code will appear in the Received message window and a description will appear below the Received message window. If the command was carried out successfully, 0 is returned. In all other cases, there will be an error code. Use the function ErrorStringGet() to get the error code description.



API to execute: GroupMoveAbsolute(RV,110)

Received message:

```
0,
```

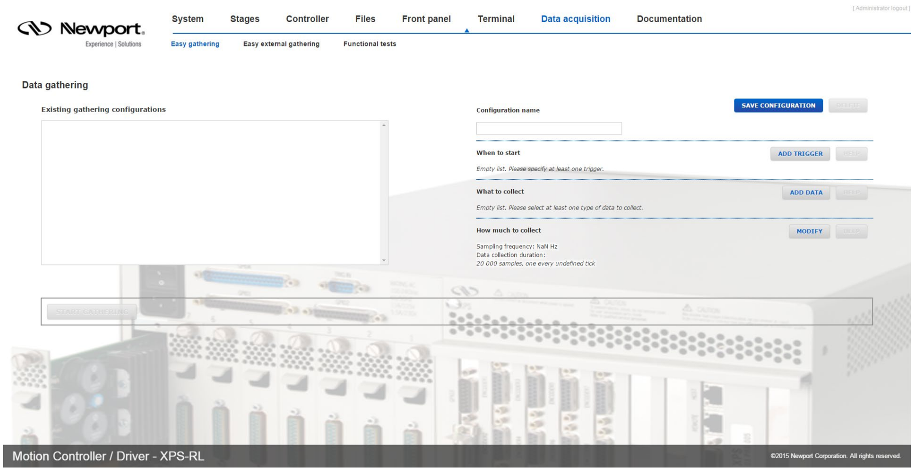
The command was carried out successfully.

Command	Status	Reply
GroupMoveAbsolute(RV,110)	0	

The functions are listed in alphabetical order and can be searched for using the search bar at the top of the Function list. Functions listed are those available for display through the **Controller → Terminal configurator** or functions available for the current system configuration. For example, if the system consists only of SingleAxis groups, no group specific functions for Spindles, XY groups, XYZ groups or MultipleAxis groups will be listed.

4.24 Data Acquisition – Easy gathering

Under **Data acquisition → Easy gathering,** users can define and save servo synchronous data gathering configurations for routine data acquisition operations including: time-based, event-based or function based gathering. For more in depth information regarding data gathering types, refer to chapter 12: “Data Gathering”.



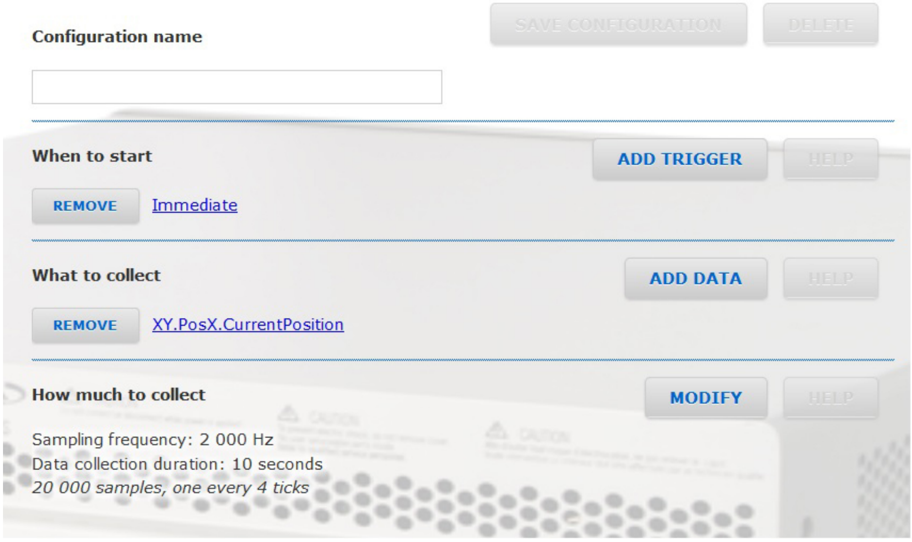
When to start: *Empty list. Please specify at least one trigger.*

What to collect: *Empty list. Please select at least one type of data to collect.*

How much to collect:

Sampling frequency: NaN Hz
Data collection duration:
20 000 samples, one every undefined tick

The following procedure describes how to use the webpage to configure easy data gathering. In the initial factory configuration, an example trigger and data configuration is set (see example in the image below). It should be deleted before to start a new gathering.



When to start: Immediate

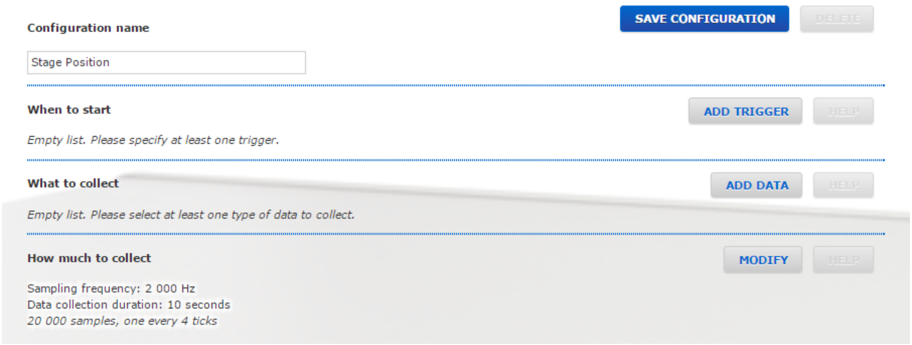
What to collect: XY.PosX.CurrentPosition

How much to collect:

Sampling frequency: 2 000 Hz
Data collection duration: 10 seconds
20 000 samples, one every 4 ticks

Step 1: Configuration name

1. Enter a name for the Gathering Configuration. *Example: Stage Position*.



Configuration name

Stage Position

When to start

Empty list. Please specify at least one trigger.

What to collect

Empty list. Please select at least one type of data to collect.

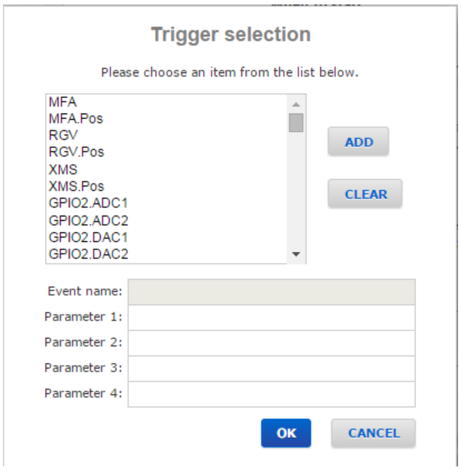
How much to collect

Sampling frequency: 2 000 Hz
Data collection duration: 10 seconds
20 000 samples, one every 4 ticks

Step2: When to start

This step configures the data collection trigger and is based upon the API function EventExtendedConfigurationTriggerSet([Actor].[Category].Event Name, Parameter1, Parameter2, Parameter3, Parameter4). For more information regarding this API, refer to section 11.1: “Events”. If this section is not empty, delete example event trigger.

1. Click on ADD TRIGGER and the following window appears:



Trigger selection

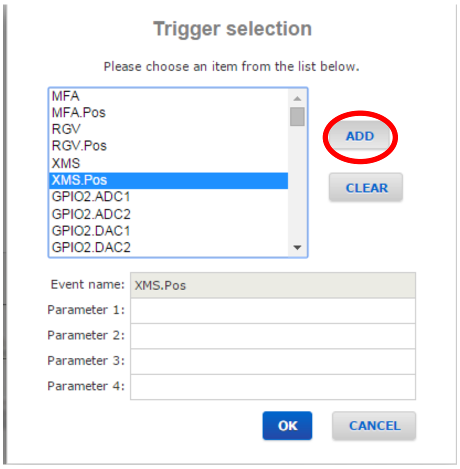
Please choose an item from the list below.

MFA
MFA.Pos
RGV
RGV.Pos
XMS
XMS.Pos
GPIO2.ADC1
GPIO2.ADC2
GPIO2.DAC1
GPIO2.DAC2

Event name:
Parameter 1:
Parameter 2:
Parameter 3:
Parameter 4:

2. From the list, highlight the group name, positioner name, TimerX or GPIO that triggers when to start collecting data and then click ADD. If the trigger selection is Immediately or Always, click OK and skip down to **Step 3: What to Collect**.

   Example: XMS.Pos



Trigger selection

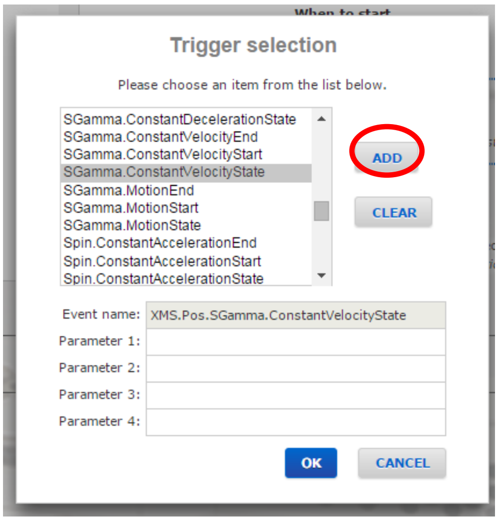
Please choose an item from the list below.

MFA
MFA.Pos
RGV
RGV.Pos
XMS
XMS.Pos
GPIO2.ADC1
GPIO2.ADC2
GPIO2.DAC1
GPIO2.DAC2

Event name: XMS.Pos
Parameter 1:
Parameter 2:
Parameter 3:
Parameter 4:

3. From the list, highlight the event that starts the data gathering and then click ADD. Depending on the event selection, Parameters 1 through Parameter 4 need to be filled in. Refer to section 11.1: "Events".

   Example: SGamma.ConstantVelocityState

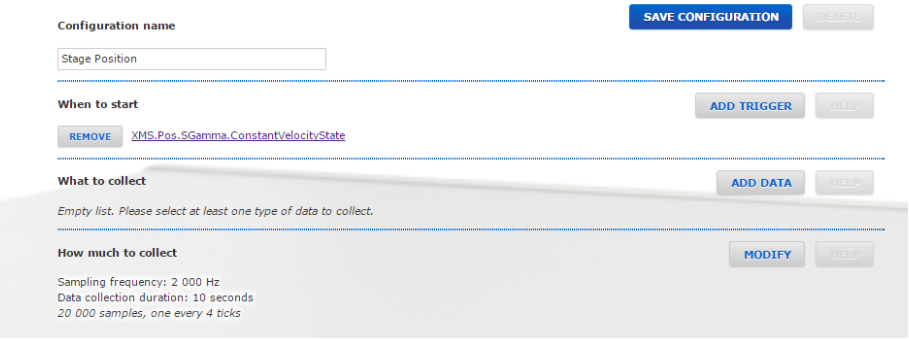


Please choose an item from the list below.

SGamma.ConstantDecelerationState
SGamma.ConstantVelocityEnd
SGamma.ConstantVelocityStart
SGamma.ConstantVelocityState
SGamma.MotionEnd
SGamma.MotionStart
SGamma.MotionState
Spin.ConstantAccelerationEnd
Spin.ConstantAccelerationStart
Spin.ConstantAccelerationState

Event name: XMS.Pos.SGamma.ConstantVelocityState
Parameter 1:
Parameter 2:
Parameter 3:
Parameter 4:

4. Click OK when the trigger has been specified. *Example: Gathering starts when the constant velocity state is reached for positioner XMS.Pos.*



Configuration name
Stage Position

When to start
XMS.Pos.SGamma.ConstantVelocityState

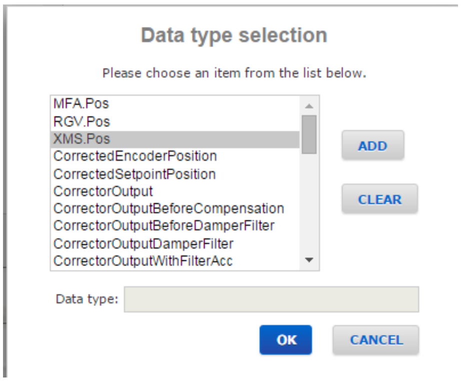
What to collect
Empty list. Please select at least one type of data to collect.

How much to collect
Sampling frequency: 2 000 Hz
Data collection duration: 10 seconds
20 000 samples, one every 4 ticks

Step 3: What to collect

This step configures the data type to be collected and is based upon the API function GatheringConfigurationSet([DataType]). Refer to chapter 12: "Data Gathering" for more information. If this section is not empty, delete example data collection type.

1. Click on ADD DATA and the following window appears.

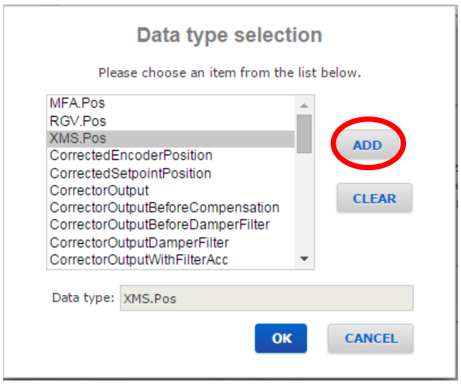


Please choose an item from the list below.

MFA.Pos
RGV.Pos
XMS.Pos
CorrectedEncoderPosition
CorrectedSetpointPosition
CorrectorOutput
CorrectorOutputBeforeCompensation
CorrectorOutputBeforeDamperFilter
CorrectorOutputDamperFilter
CorrectorOutputWithFilterAcc

Data type:

2. From the list, highlight the positioner name or GPIO from which data will be collected and then click ADD. *Example: XMS.Pos*



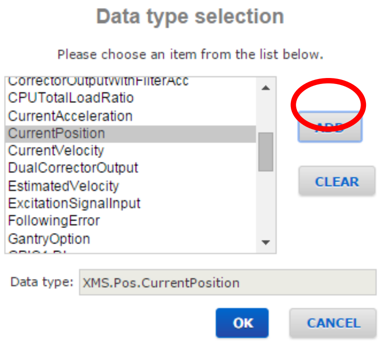
Data type selection

Please choose an item from the list below.

MFA.Pos
RGV.Pos
XMS.Pos
CorrectedEncoderPosition
CorrectedSetpointPosition
CorrectorOutput
CorrectorOutputBeforeCompensation
CorrectorOutputBeforeDamperFilter
CorrectorOutputDamperFilter
CorrectorOutputWithFilterAcc
ADD
CLEAR
OK
CANCEL

Data type: XMS.Pos

3. If a positioner name is selected, from the list, highlight the data type to be collected and then click ADD. *Example: CurrentPosition*

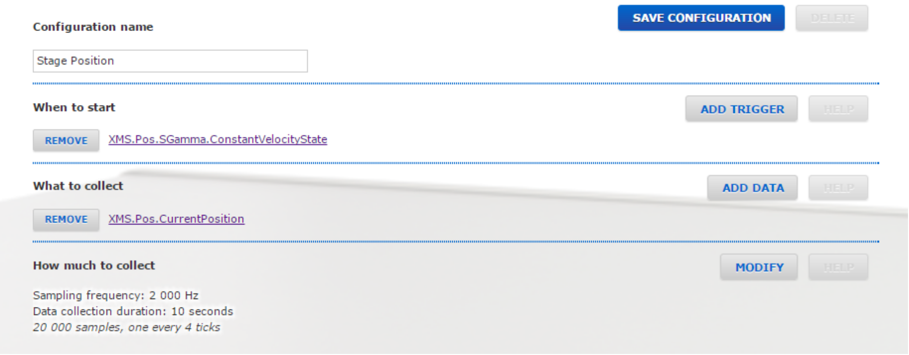


Data type selection

Please choose an item from the list below.

Data type: XMS.Pos.CurrentPosition

4. Click OK when the data type has been specified. *Example: once the trigger event occurs, current position values will be collected for positioner XMS.Pos.*



Configuration name

Stage Position

When to start

XMS.Pos.SGamma.ConstantVelocityState

What to collect

XMS.Pos.CurrentPosition

How much to collect

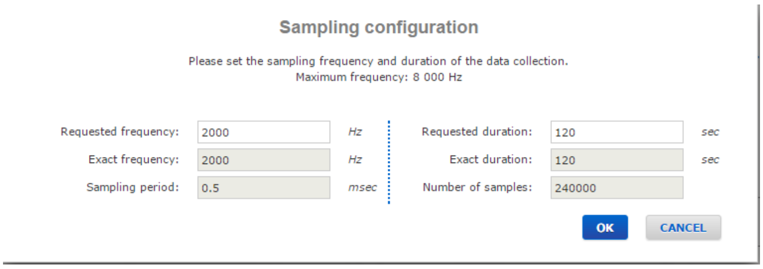
Sampling frequency: 2 000 Hz
Data collection duration: 10 seconds
20 000 samples, one every 4 ticks

5. Repeat **Step 3 Add Data** to add other data types to be collected under this specific Gathering Configuration.

Step 4: How much to collect

This step specifies the sampling frequency and sampling duration for the gathering configuration and is based upon the API function EventExtendedConfigurationActionSet(GatheringRun, Nb of points, Divisor, 0, 0). Refer to section 11.2: "Actions" for more information.

1. Click on MODIFY to specify the frequency at which data is collected and the duration of the data sampling. The maximum sampling frequency is at the XPS servo rate (8 kHz by default). *Example: Data will be collected at 2 kHz frequency for a total of 2 minutes.*



Sampling configuration

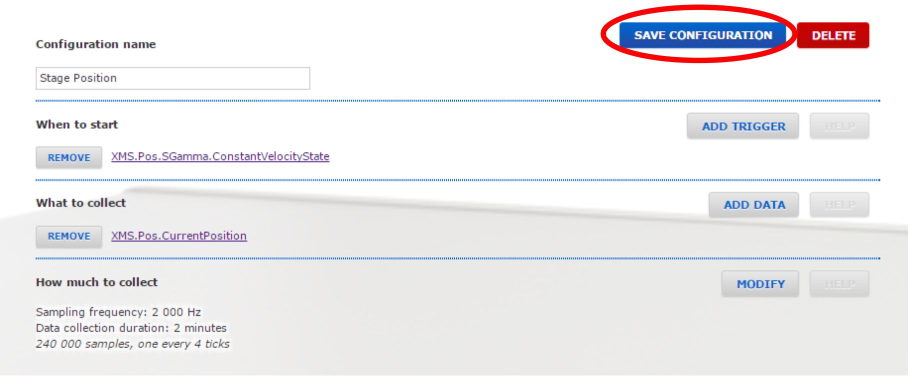
Please set the sampling frequency and duration of the data collection.
Maximum frequency: 8 000 Hz

Requested frequency: 2000 Hz
Exact frequency: 2000 Hz
Sampling period: 0.5 msec
Requested duration: 120 sec
Exact duration: 120 sec
Number of samples: 240000

Step 5: Save Configuration

1. Click on SAVE CONFIGURATION to save the gathering configuration.

   Example: Gathering starts when the constant velocity state is reached for positioner XMS.Pos. Once the trigger event occurs, current position values will be collected for positioner XMS.Pos. Data will be collected at 2 kHz frequency for a total of 2 minutes.



Configuration name

Stage Position

When to start

XMS.Pos.SGamma.ConstantVelocityState

What to collect

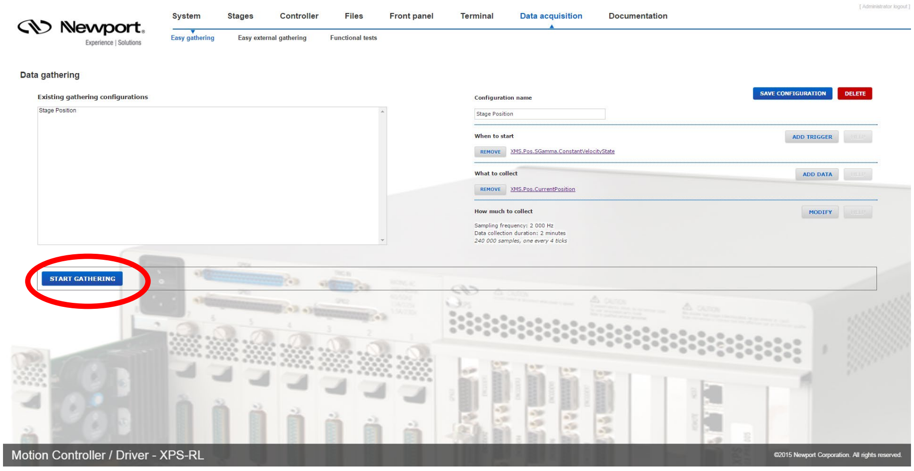
XMS.Pos.CurrentPosition

How much to collect

Sampling frequency: 2 000 Hz
Data collection duration: 2 minutes
240 000 samples, one every 4 ticks

Step 6: Start Gathering

1. To begin gathering data, click on START GATHERING.



Data gathering

Existing gathering configurations

Stage Position

Configuration name

Stage Position

When to start

XMS.Pos.SGamma.ConstantVelocityState

What to collect

XMS.Pos.CurrentPosition

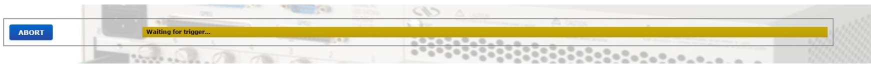
How much to collect

Sampling frequency: 2 000 Hz
Data collection duration: 2 minutes
240 000 samples, one every 4 ticks

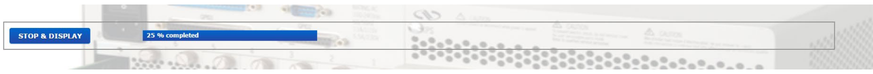
Motion Controller / Driver - XPS-RL

©2015 Newport Corporation. All rights reserved.

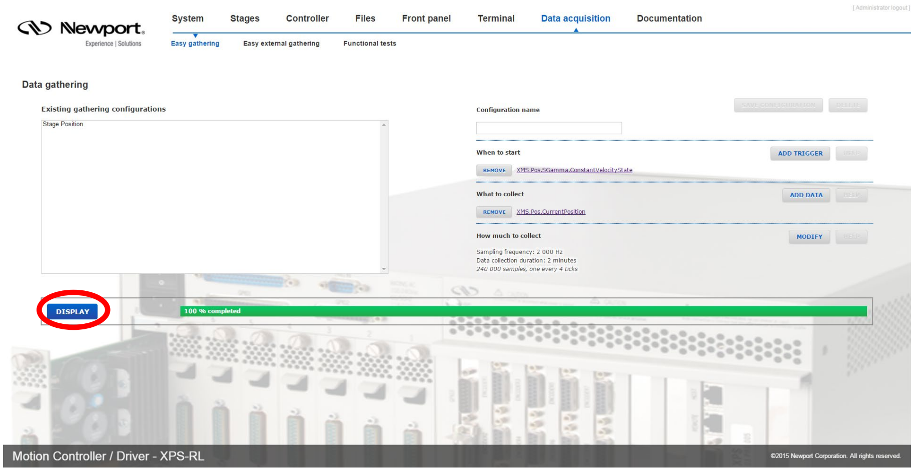
2. The controller than begins to monitor for the configured trigger.



3. Once the trigger even occurs, the data acquisition begins and a status bar appears.



Once the data acquisition is complete, click on DISPLAY to view the data.



Data gathering

Existing gathering configurations

Stage Position

Configuration name

When to start

XMS.Pos.SGamma.ConstantVelocityState

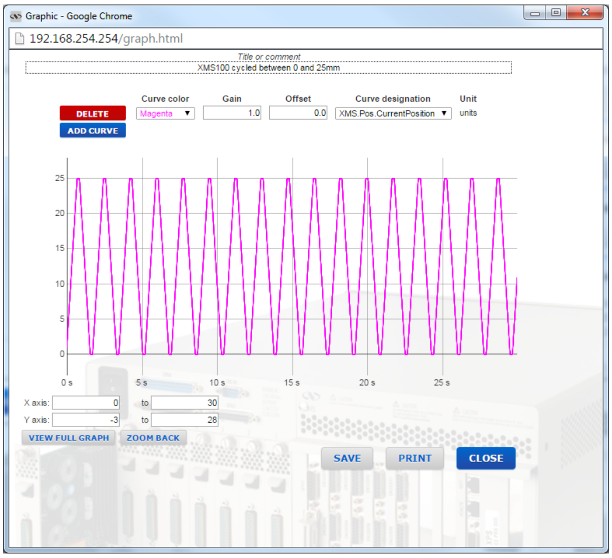
What to collect

XMS.Pos.CurrentPosition

How much to collect

Sampling frequency: 2 000 Hz
Data collection duration: 2 minutes
240 000 samples, one every 4 ticks

Example: Data Graph



Title or comment: XMS100 cycled between 0 and 25mm

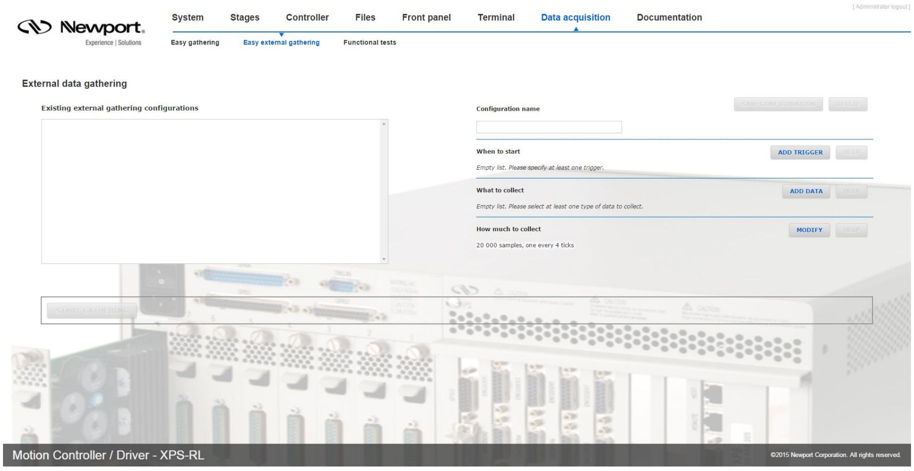
Curve color	Gain	Offset	Curve designation	Unit
Magenta	1.0	0.0	XMS.Pos.CurrentPosition	units

X axis: 0 to 30

Y axis: -3 to 28

4.25 Data Acquisition – Easy External Gathering

Under **Data acquisition → Easy external gathering,** users can define and save data gathering configurations for routine data acquisition operations triggered by an external device. For more in depth information regarding data gathering types, refer to section 12.4: "Trigger-Based (External) Data Gathering". In the initial factory configuration, an example trigger and data configuration is set. It should be deleted before to start a new gathering.



External data gathering

Existing external gathering configurations

Configuration name

When to start

Empty list. Please specify at least one trigger.

What to collect

Empty list. Please select at least one type of data to collect.

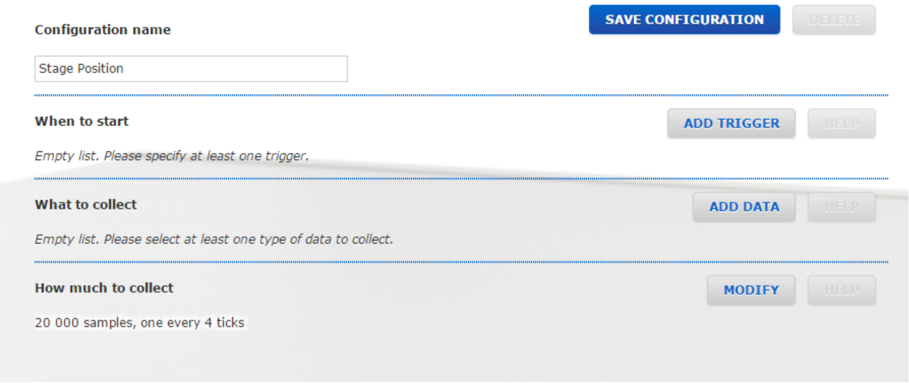
How much to collect

20 000 samples, one every 4 ticks

The following procedure describes how to use the webpage to configure easy external data gathering.

Step 1: Configuration name

1. Enter a name for the Gathering Configuration. *Example: Stage Position.*



Configuration name

Stage Position

When to start

Empty list. Please specify at least one trigger.

What to collect

Empty list. Please select at least one type of data to collect.

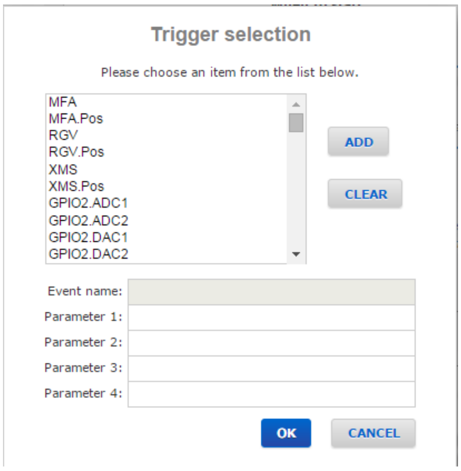
How much to collect

20 000 samples, one every 4 ticks

Step 2: When to start

This step configures the data collection trigger and is based upon the API function EventExtendedConfigurationTriggerSet([Actor].[Category].Event Name, Parameter1, Parameter2, Parameter3, Parameter4). For more information regarding this API, refer to section 11.1: “Events”. If this section is not empty, delete example event trigger.

1. Click on ADD TRIGGER and the following window appears:



Trigger selection

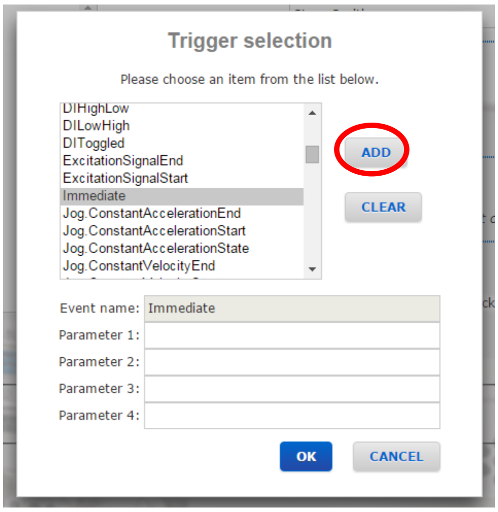
Please choose an item from the list below.

MFA
MFA.Pos
RGV
RGV.Pos
XMS
XMS.Pos
GPIO2.ADC1
GPIO2.ADC2
GPIO2.DAC1
GPIO2.DAC2

Event name:
Parameter 1:
Parameter 2:
Parameter 3:
Parameter 4:

2. From the list, highlight Immediate or Always. Click ADD and then click OK.

   Example: Immediate



Trigger selection

Please choose an item from the list below.

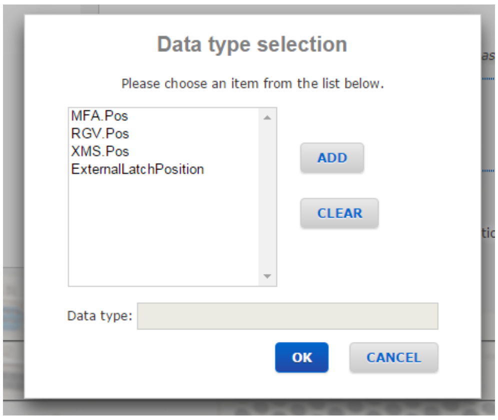
DIHighLow
DILowHigh
DIToggled
ExcitationSignalEnd
ExcitationSignalStart
Immediate
Jog.ConstantAccelerationEnd
Jog.ConstantAccelerationStart
Jog.ConstantAccelerationState
Jog.ConstantVelocityEnd

Event name: Immediate
Parameter 1:
Parameter 2:
Parameter 3:
Parameter 4:

Step 3: What to collect

This step configures the data type to be collected and is based upon the API function GatheringExternalConfigurationSet([DataType]). Refer to section 12.4: “Trigger-Based (External) Data Gathering”. If this section is not empty, delete example data collection type.

1. Click on ADD DATA and the following window appears.



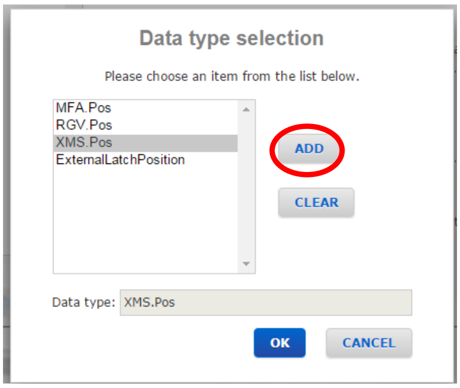
Data type selection

Please choose an item from the list below.

MFA.Pos
RGV.Pos
XMS.Pos
ExternalLatchPosition

Data type:

2. From the list, highlight the positioner name from which position data will be collected and then click ADD. *Example: XMS.Pos*



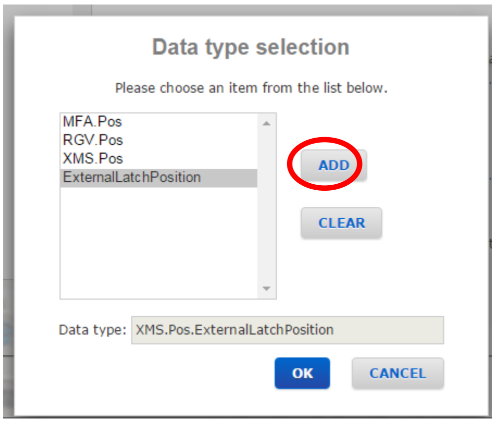
Data type selection

Please choose an item from the list below.

MFA.Pos
RGV.Pos
XMS.Pos
ExternalLatchPosition

Data type: XMS.Pos

3. From the list, highlight the data type ExternalLatchPosition, and then click ADD. *Example: ExternalLatchPosition*



Data type selection

Please choose an item from the list below.

MFA.Pos
RGV.Pos
XMS.Pos
ExternalLatchPosition

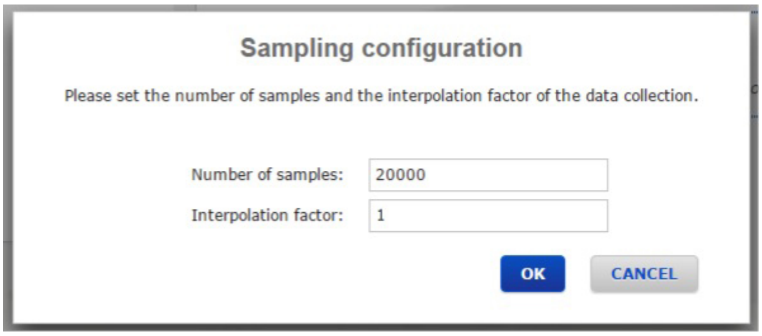
Data type: XMS.Pos.ExternalLatchPosition

4. Click OK when the data type has been specified.
5. Repeat **Step 3 Add Data** to add other data types to be collected under this specific Gathering Configuration.

Step 4: How much to collect

This step specifies the number of data points per set and the ratio for collecting data sets relative to external triggers is fixed at 1:1. This step is based upon the API function EventExtendedConfigurationActionSet(ExternalGatheringRun, Nb of points, 1, 0, 0). Refer to section 11.2: “Actions” for more information.

1. Click on MODIFY to specify the number of data sets to be collected and the ratio of external triggers to data set collection. *Example: 20000 data samples will be collected for every external trigger.*



Sampling configuration

Please set the number of samples and the interpolation factor of the data collection.

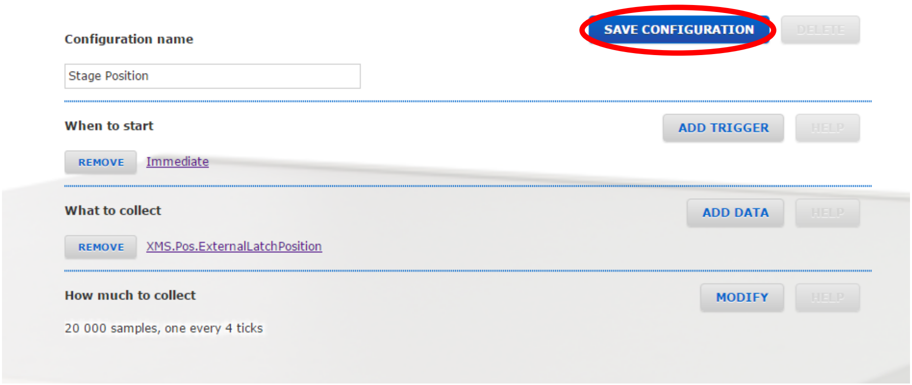
Number of samples: 20000

Interpolation factor: 1

Step 5: Save Configuration

1. Click on SAVE CONFIGURATION to save the gathering configuration.

Example:



Configuration name: Stage Position

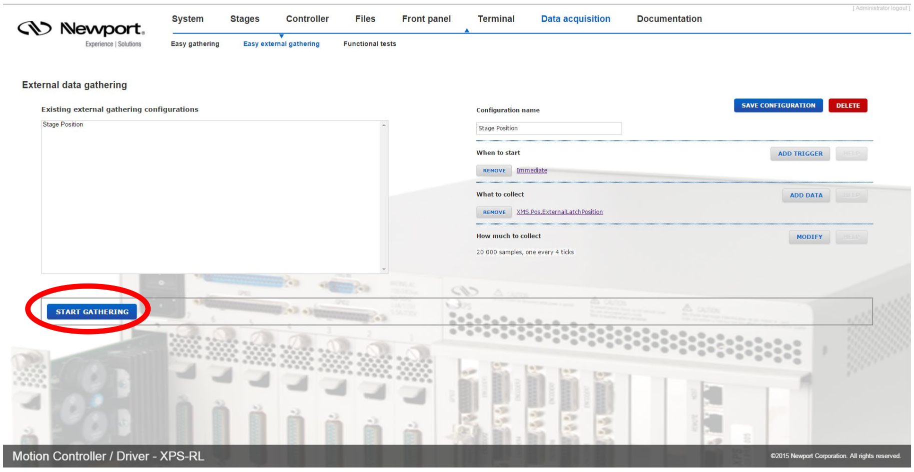
When to start: Immediate

What to collect: XMS.Pos.ExternalLatchPosition

How much to collect: 20 000 samples, one every 4 ticks

Step 6: Start Gathering

1. To begin gathering data, click on START GATHERING.



External data gathering

Existing external gathering configurations

Stage Position

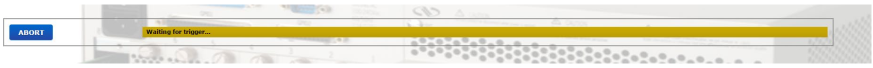
Configuration name: Stage Position

When to start: Immediate

What to collect: XMS.Pos.ExternalLatchPosition

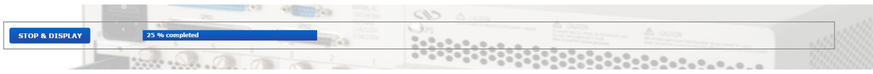
How much to collect: 20 000 samples, one every 4 ticks

2. The controller than immediately begins to monitor for the configured external trigger.



Waiting for trigger...

3. Once the trigger even occurs, the data acquisition begins and a status bar appears.



25 % completed

4. Once the data acquisition is complete, click on DISPLAY to view the data.

[INSERT SCREENSHOT]

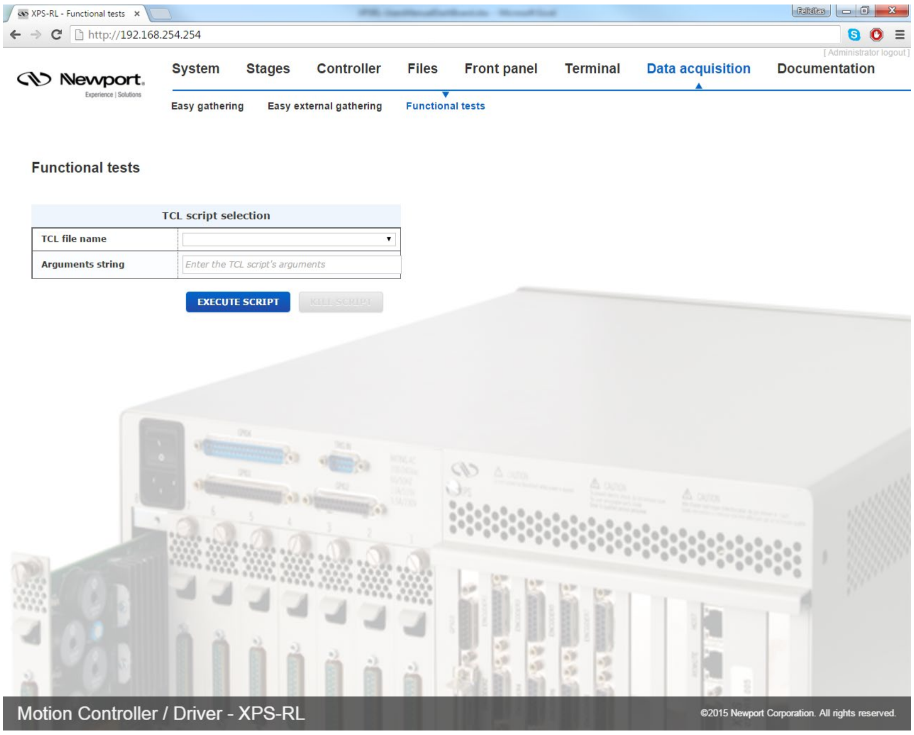
Example: Data Graph

[INSERT SCREENSHOT]

4.26 Data Acquisition – Functional Tests

The FUNCTIONAL TESTS page allows running TCL scripts saved in the "/Admin/Public/Scripts/ FunctionalTests" folder of the XPS controller. Supplied in the firmware, the Functional Tests scripts will then display the results of a gathering file.

Select the TCL Script name then press "Execute script" to run the script or "Kill script" to stop its execution.



Functional tests

TCL script selection	
TCL file name	▾
Arguments string	Enter the TCL script's arguments

EXECUTE SCRIPT
KILL SCRIPT
Motion Controller / Driver - XPS-RL
©2015 Newport Corporation. All rights reserved.

4.27 Files – Gathering Files

In this webpage gathering files stored on the XPS controller can be downloaded, viewed, edited or deleted. To generate gathering files refer to the data gathering sections of the user's manual.

Download/Edit/View/Delete

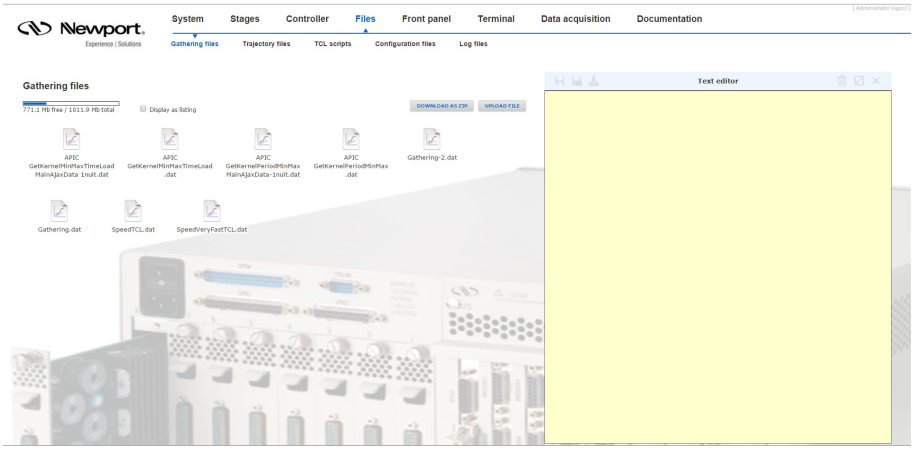
Click on the gathering file name to open the file in the text editor and a window will appear with a graphical representation of the data. In the text editor the user can edit, save, save as, view, download or delete the gathering file.

Upload

Click UPLOAD FILE to upload a gathering file from the user's PC to the XPS controller.

Download All

Click DOWNLOAD AS ZIP to download all gathering files to the user's PC.



Gathering files

771.1 Mb free / 1011.9 Mb total

APIC GetKernelMinMaxTimeLoad MainAjaxData 1nuit.dat
APIC GetKernelMinMaxTimeLoad .dat
APIC GetKernelPeriodMinMax MainAjaxData-1nuit.dat
APIC GetKernelPeriodMinMax .dat
Gathering-2.dat
Gathering.dat
SpeedTCL.dat
SpeedVeryFastTCL.dat

WARNING

Ensure the web browser zoom is set at 100% to avoid image corruption when displaying a gathering file.

4.28 Files – Trajectory Files

In this webpage trajectory files stored on the XPS controller can be downloaded, uploaded, viewed, edited, created or deleted.

Download/Edit/View/Delete

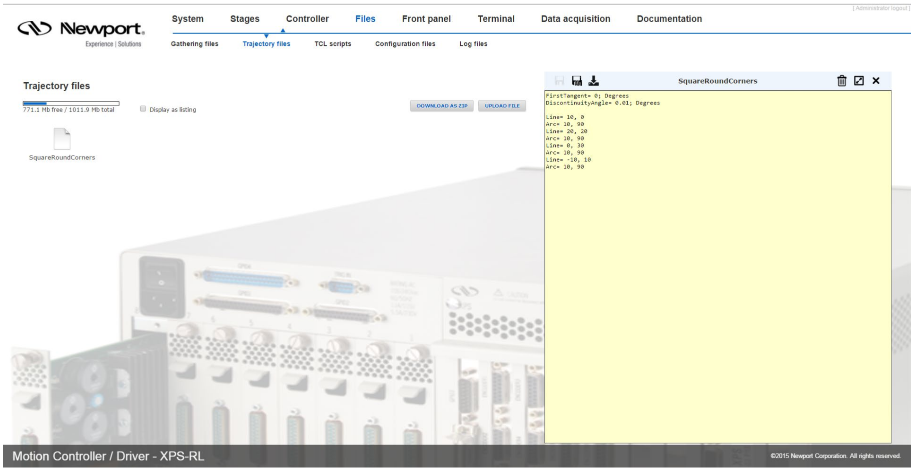
Click on the trajectory file name to open the file in the text editor. In the text editor the user can edit, save, save as, view, download or delete the trajectory file.

Upload

Click UPLOAD FILE to upload a trajectory file from the user's PC to the XPS controller.

Download All

Click DOWNLOAD AS ZIP to download all trajectory files to the user's PC.



```
FirstTangent= 0; Degrees
DiscontinuityAngle= 0.01; Degrees

Line= 10, 0
Arc= 10, 90
Line= 20, 20
Arc= 10, 90
Line= 0, 30
Arc= 10, 90
Line= -10, 10
Arc= 10, 90
```

4.29 Files – TCL Scripts

In this webpage TCL script files stored on the XPS controller can be downloaded, viewed, edited or deleted. TCL scripts could also be uploaded through this webpage. After a TCL scripted has been generated from the

Download/Edit/View/Delete

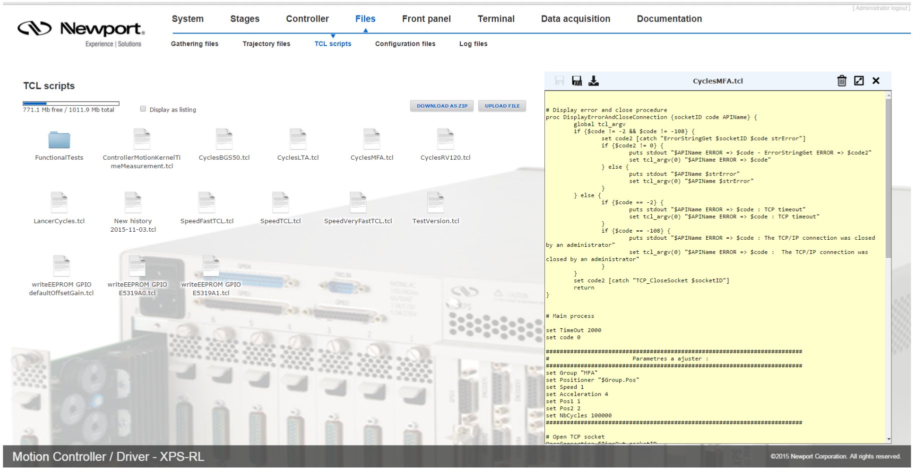
Click on the TCL script file name to open the file in the text editor. In the text editor the user can edit, save, save as, view, download or delete the TCL script.

Upload

Click UPLOAD FILE to upload a TCL script file from the user's PC to the XPS controller.

Download All

Click DOWNLOAD AS ZIP to download all TCL script files to the user's PC.



```
# Display error and close procedure
proc DisplayErrorAndCloseConnection {socketID code APIName} {
        global tcl_argv
        if {$code != -2 && $code != -108} {
                set code2 [catch "ErrorStringGet $socketID $code strError"]
                if {$code2 != 0} {
                        puts stdout "$APIName ERROR => $code - ErrorStringGet ERROR => $code2"
                        set tcl_argv(0) "$APIName ERROR => $code"
                } else {
                        puts stdout "$APIName $strError"
                        set tcl_argv(0) "$APIName $strError"
                }
        } else {
                if {$code == -2} {
                        puts stdout "$APIName ERROR => $code : TCP timeout"
                        set tcl_argv(0) "$APIName ERROR => $code : TCP timeout"
                }
                if {$code == -108} {
                        puts stdout "$APIName ERROR => $code : The TCP/IP connection was closed
by an administrator"
                        set tcl_argv(0) "$APIName ERROR => $code :  The TCP/IP connection was
closed by an administrator"
                }
        }
        set code2 [catch "TCP_CloseSocket $socketID"]
        return
}


# Main process

set TimeOut 2000
set code 0

###########################################################################
#                         Parametres a ajuster :
###########################################################################
set Group "MFA"
set Positioner "$Group.Pos"
set Speed 1
set Acceleration 4
set Pos1 1
set Pos2 2
set NbCycles 100000
###########################################################################

# Open TCP socket
OpenConnection $TimeOut socketID
```

4.30 Files – Configuration Files

In this webpage Configuration files stored on the XPS controller can be downloaded, uploaded, viewed, edited or deleted. Note: Users must be logged in with Administrator rights in order to access this webpage.

Download/Edit/View/Delete

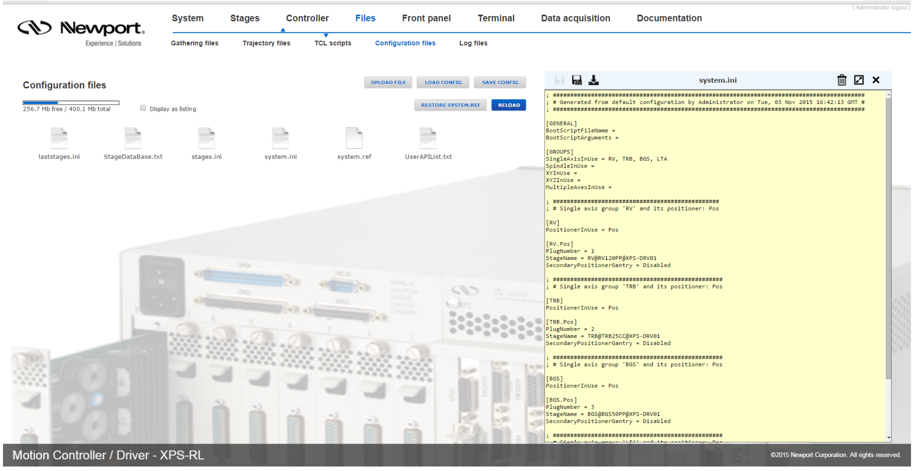
Click on the configuration name to open the file in the text editor. In the text editor the user can edit, save, save as, view, download or delete the configuration file.

Upload

Click UPLOAD FILE to upload a configuration file from the user's PC to the XPS controller. When uploading a file, take note of the file extension.

Download All

Click DOWNLOAD AS ZIP to download all configuration files to the user's PC.



```
; ####################################################################################
; # Generated from default configuration by Administrator on Tue, 03 Nov 2015 16:42:13 GMT #
; ####################################################################################

[GENERAL]
BootScriptFileName =
BootScriptArguments =

[GROUPS]
SingleAxisInUse = RV, TRB, BGS, LTA
SpindleInUse =
XYInUse =
XYZInUse =
MultipleAxesInUse =

; ###############################################
; # Single axis group 'RV' and its positioner: Pos

[RV]
PositionerInUse = Pos

[RV.Pos]
PlugNumber = 1
StageName = RV@RV120PP@XPS-DRV01
SecondaryPositionerGantry = Disabled

; ################################################
; # Single axis group 'TRB' and its positioner: Pos

[TRB]
PositionerInUse = Pos

[TRB.Pos]
PlugNumber = 2
StageName = TRB@TRB25CC@XPS-DRV01
SecondaryPositionerGantry = Disabled

; ################################################
; # Single axis group 'BGS' and its positioner: Pos

[BGS]
PositionerInUse = Pos

[BGS.Pos]
PlugNumber = 3
StageName = BGS@BGS50PP@XPS-DRV01
SecondaryPositionerGantry = Disabled

; ################################################
; # Single axis group 'LTA' and its positioner: Pos
```

4.31 Files – Log Files

In this webpage log files stored on the XPS controller can be downloaded, uploaded, viewed, edited or deleted. Note: Users must be logged in with Administrator rights in order to access this webpage.

Download/Edit/View/Delete

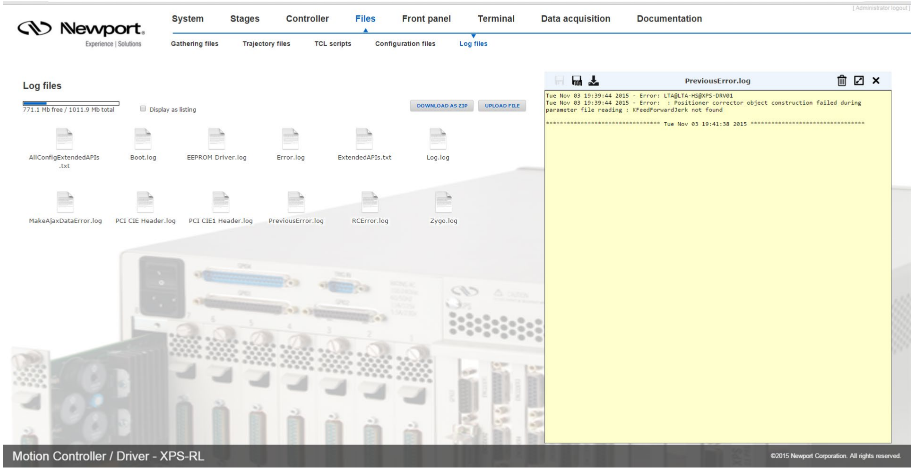
Click on the log file name to open the file in the text editor. In the text editor the user can edit, save, save as, view, download or delete the log file.

Upload

Click UPLOAD FILE to upload a log file from the user's PC to the XPS controller. When uploading a file, take note of the file extension.

Download All

Click DOWNLOAD AS ZIP to download all log files to the user's PC.



```
Tue Nov 03 19:39:44 2015 - Error: LTA@LTA-HS@XPS-DRV01
Tue Nov 03 19:39:44 2015 - Error:  : Positioner corrector object construction failed during
parameter file reading : KFeedForwardJerk not found

******************************* Tue Nov 03 19:41:38 2015 *******************************
```

4.32 FTP (File Transfer Protocol) Connection

All usual file management can be done from the controller website interface. Nevertheless, FTP connection is another option to manage file transfers from an external application. FTP works in the same way as HTTP for transferring web pages from a server to a user's browser and SMTP for transferring electronic mail across the Internet. FTP uses the Internet TCP/IP protocol to enable data transfer.

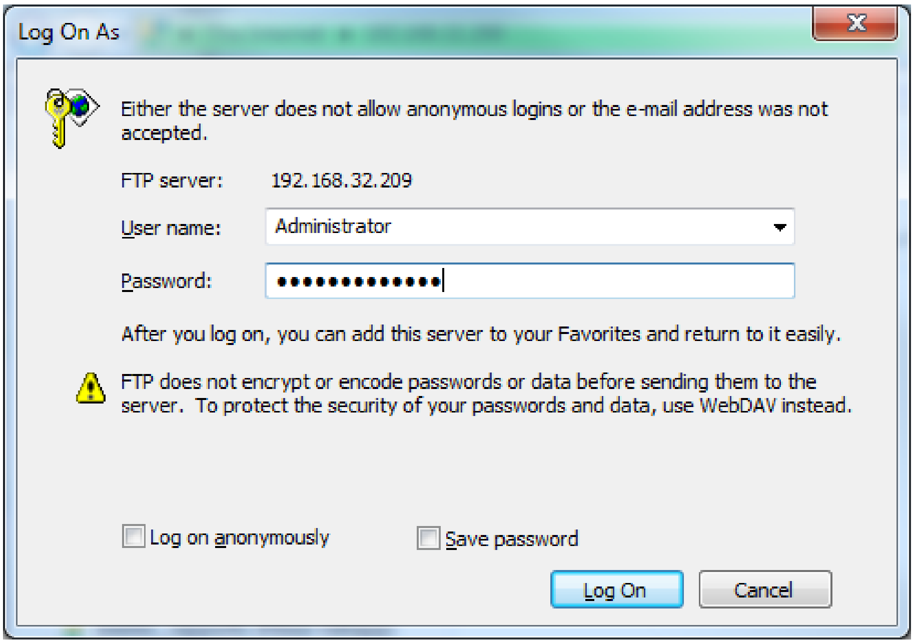
An FTP connection is needed to view the information saved in the XPS controller, to download documentation, to transfer configuration files (to modify them locally), to transfer TCL scripts, etc…

To connect to the FTP server:

- Start the XPS controller and wait until the boot sequence completes.
- Open an Internet browser window. Windows explorer is another option to access the files.
- Connect to the FTP server with the IP address of the controller:

Example:

- Select "File" from the menu of the Internet browser, and then "Connect as…". The following window appears:



Either the server does not allow anonymous logins or the e-mail address was not accepted.

FTP server: 192.168.32.209

User name: Administrator

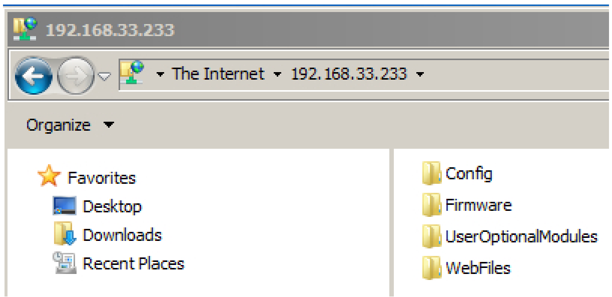
Password: •••••••••••••

After you log on, you can add this server to your Favorites and return to it easily.

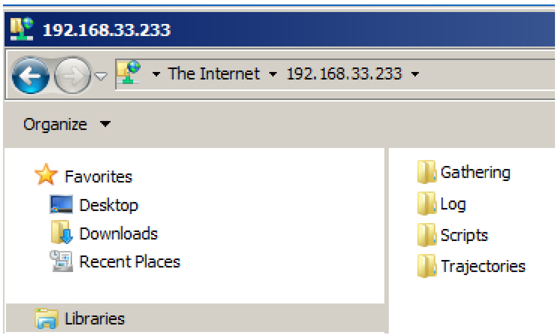
FTP does not encrypt or encode passwords or data before sending them to the server. To protect the security of your passwords and data, use WebDAV instead.

Specify the user name and password. Press log on. The folders of the XPS controller are displayed (see below). Browse through the different folders and transfer data from or to your host PC the same way as Windows Explorer.

When connected to the controller with FTP as Administrator, the user has access to configuration files. The Administrator home directory contains folders: Config, Firmware, UserOptionalModules and Webfiles.



When connected to the controller with FTP as an ordinary user such as Anonymous, the user has access to Public files. The ordinary user the home directory contains folders: Gathering, Log, Scripts, and Trajectories.



5.0 Maintenance and Service

5.1 Enclosure Cleaning

The XPS Controller/Driver should only be cleaned with a sufficient amount of soapy water solution. Do not use an acetone or alcohol solution, this will damage the finish of the enclosure.

5.2 Obtaining Service

The XPS Controller/Driver contains no user serviceable parts. To obtain information regarding factory service, contact Newport Corporation or your Newport representative and be ready with the following information:

- Instrument model number (on front panel) and original order number.
- Instrument serial number (on rear panel).
- Description of the problem.

If the XPS is to be returned to Newport Corporation, a Return Number will be issued, which should be referenced in the shipping documents.

Complete a copy of the Service Form found at the end of this User's Manual and include it with your shipment.

5.3 Troubleshooting

For troubleshooting, the user can query different error and status information from the controller. The XPS controller provides the Positioner Error, the Positioner Hardware Status, the Positioner Driver Status, the Group Status, and also a general system error.

If there is an error during command execution, the controller will return an error code. The command ErrorStringGet can be used to retrieve the description corresponding to the error code.

The following function commands are used to retrieve Positioner Error and Positioner Hardware Status:

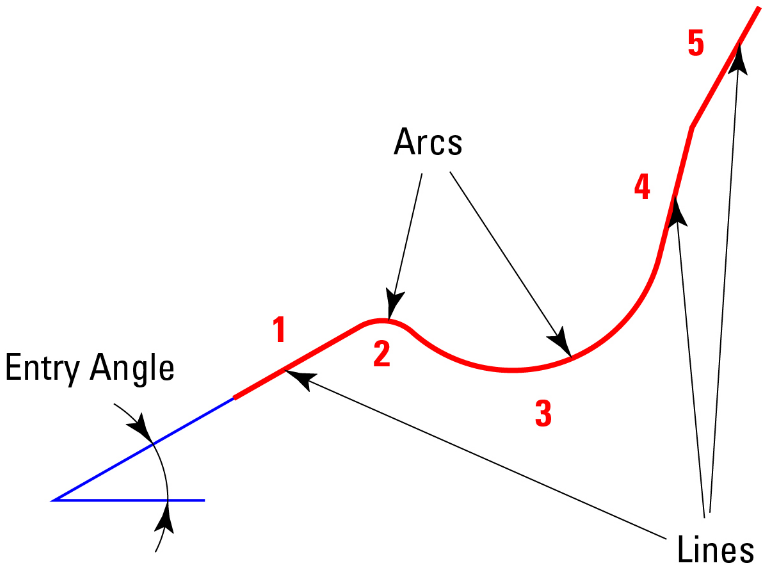
- PositionerErrorGet: Returns an error code.
- PositionerErrorStringGet: Returns the description of the error code.
- PositionerHardwareStatusGet: Returns the status code.
- PositionerHardwareStatusStringGet: Returns the description corresponding to the status code.

In a fault condition, it is also very important to know the current status of the group and the cause of the transition from the previous group status to the current group state. The following functions can be used to retrieve the Group Status:

- GroupStatusGet: Returns the group status code.
- GroupStatusStringGet: Returns the description corresponding to the group status code.

NOTE

Refer to the Programmer's Manual for a complete list of status and error codes. Also refer to chapter 4.0 for troubleshooting the XPS controller with the help of its web utilities.



Motion Tutorial

6.0 XPS Architecture

6.1 Introduction

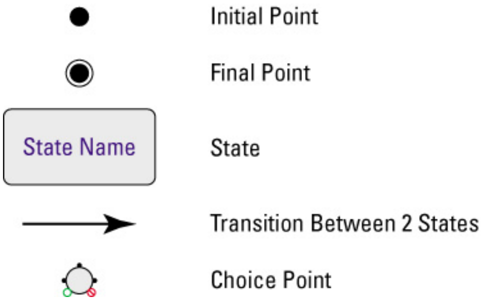
The architecture of the XPS firmware is based on an object-oriented approach. Objects are key to understanding this approach. Real-world objects share two characteristics: state and behavior. Software objects are modeled after real-world objects, so they have state and behavior too. A software object maintains its state in one or more variables. A variable is an item of data named by an identifier. A software object implements its behavior with methods. A method is a function (subroutine) associated with an object. Therefore, an object is a software bundle of variables and related methods. Encapsulating related variables and methods into a neat software bundle is a simple yet powerful idea that provides two primary benefits to software developers:

- **Modularity:** The source code for an object can be written and maintained independent of the source code for other objects. Also, an object can be easily passed around in the system.
- **Hidden information:** An object has a public interface that other objects can use to communicate with it. The object can maintain private information and methods that can be changed at any time without affecting the other objects that depend on it.

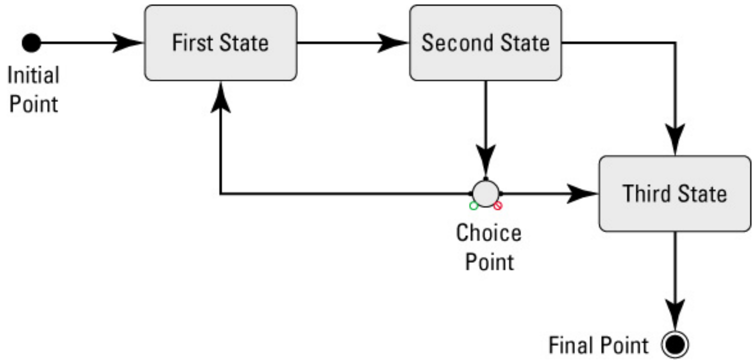
All objects have a life cycle and state diagrams are used to show the life cycle of the objects. The transition from one state to another is initiated after receiving a message from another object. Like all other diagrams, state diagrams can be nested in different layers to keep them simple and easy to read.

6.2 State Diagrams

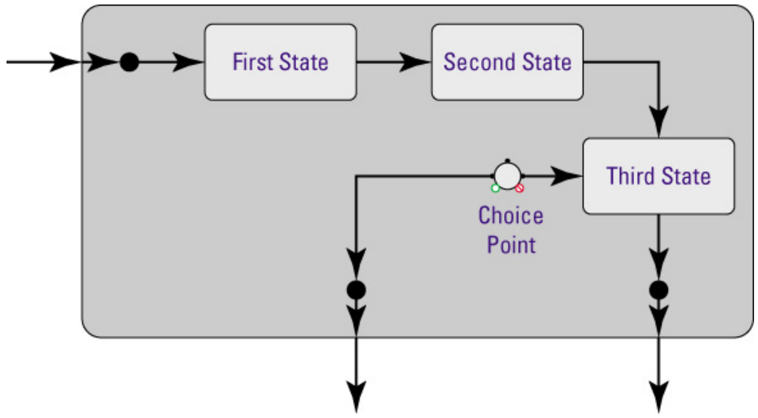
State diagrams are a way to describe the behavior of each group or object. They represent each steady state of a group and every transition between states in an exhaustive way. State diagrams contain the following components:



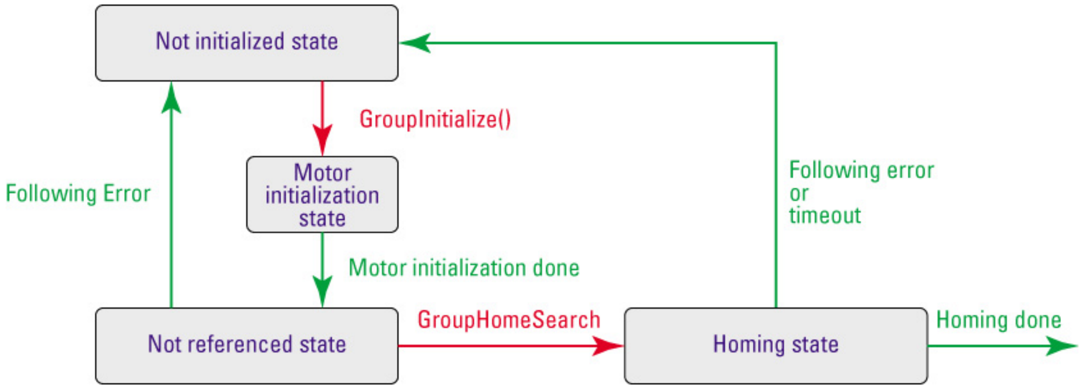
Here is an example of a simple stage diagram:



State diagrams can also include sub state diagrams:



The state diagrams that are specific to the XPS controller follow the same format. Within the XPS controller, all positioners are assigned to different motion groups. These motion groups have the following common state diagram:



As shown in the above state diagram, all groups have to be first initialized and then homed before any group is ready to perform any other function. Once the group is homed, it is in a ready state. There are five different motion groups available with the XPS controller:

- SingleAxis group
- Spindle group
- XY group
- XYZ group
- MultipleAxes group

Each group also has group specific states. Please refer to the Programmer's Manual for group-specific state diagrams for the five different groups.

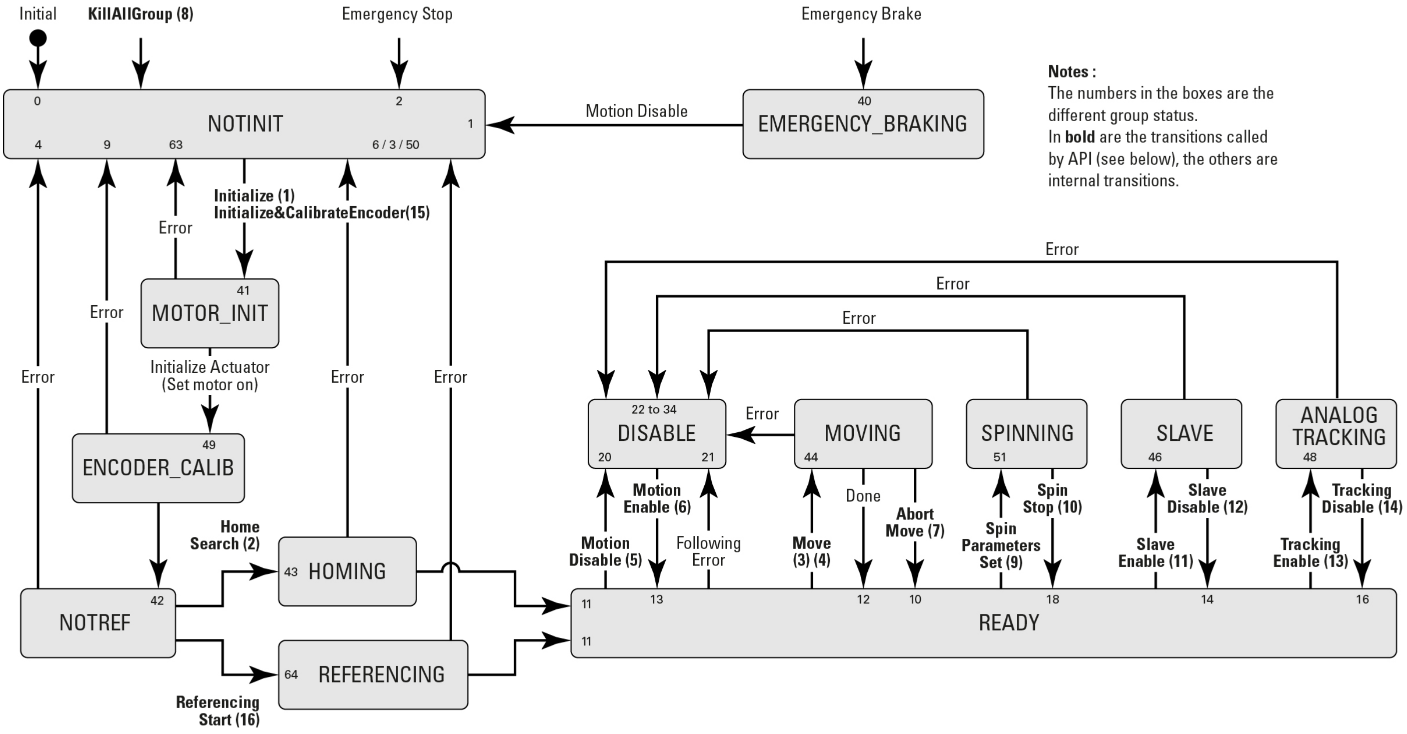
All positioners of a group are bundled together for security handling. Security handling of different groups is treated independently. Following is a list of the different faults and consequences that can happen in the XPS controller:

Error type	**Consequence**
General inhibition Motor fault Encoder fault	Emergency stop
End of travel	Emergency brake
Following error	Motion disable

- After an emergency brake or an emergency stop, both considered major faults, the corresponding group goes to a "not initialized" state: the system has to be initialized and homed again before any further motion.
- After a following error, as it is considered a minor fault, the corresponding group goes to a "Disable" state: a GroupMotionEnable() command puts the system back into "ready" state.

At any given time the group status can be queried from the controller. The function **GroupStatusGet (GroupName)** returns the current state number. The state numbers correspond to the state and to the event that generated the transition (if any). The function **GroupStatusStringGet (StateNumber)** returns the state description corresponding to the state number.

Similar to the Group State, the Controller Status can also be queried using the API **ControllerStatusGet**() or **ControllerStatusRead**(). The status numbers correspond to the status and event that generated the status. To get a description of the status code use **ControllerStatusStringGet**(). For more information refer to the XPS Programmer's Manual.



Called function

1. GroupInitialize	**7.** GroupMoveAbort	**13.** GroupAnalogTrackingModeEnable
2. GroupHomeSearch	**8.** GroupKill or KillAll	**14.** GroupAnalogTrackingModeDisable
3. GroupMoveAbsolute	**9.** GroupSpinParametersSet	**15.** GroupInitializeWithEncoderCalibration
4. GroupMoveRelative	**10.** GroupSpinModeStop	**16.** GroupReferencingStart
5. GroupMotionDisable	**11.** SpinSlaveModeEnable	**17.** GroupReferencingStop
6. GroupMotionEnable	**12.** SpinSlaveModeDisable	

State diagram of the XPS controller.

6.3 Motion Groups

Within the XPS controller, each positioner or axis of motion must be assigned to a motion group. This “group” can either be a SingleAxis group, a Spindle group, an XY group, an XYZ group or a MultipleAxes group. Once defined, the XPS automatically manages all safeties and trajectories of the motion group from the same function. For instance, the function **GroupHomeSearch (GroupName)** automatically homes the whole motion group GroupName independent of its definition as a SingleAxis group, a Spindle group, an XY group, an XYZ group or a MultipleAxes group. Within the system configuration file, system.ini, select the home sequence as “sequential”, one positioner after the other, or in “parallel”, with all positioners homing at the same time. With a single function such as **GroupMoveAbsolute (GroupName, Position),** the whole motion group, GroupName, is moved synchronously to the defined absolute position, where **“Position”** may be one or more parameters depending on the number of positioners this motion group contains. This same command can be used to move a single positioner of a group to an absolute position by using the syntax **GroupMoveAbsolute (GroupName.PositionerName, Position1)**. These powerful, object-oriented functions are not only extremely intuitive and easy to use, they are also more consistent with other programming methods and reduce the number of commands learned compared to traditional mnemonic commands.

Another benefit provided by motion groups is improved error handling. For instance, whenever an error occurs due to a following error or a loss of the end-of-run signal, only the motion group where the error originated is affected (disabled) while all other motion groups remain active and enabled. The XPS manages these events automatically. This greatly reduces complexity and improves the security and safety of sensitive applications.

To illustrate this, let’s consider a typical scanning application. If there is an error on the stepping axis of the XY table (which is set-up as an XY group), only the XY table is disabled while the auto-focusing tool (a vertical stage that is defined as a separate SingleAxis group) continues to function.

Each of the five available motion groups has specific features:

6.3.1 Specific SingleAxis Group Features

Master-Slave – To enable this function, the slaved positioner must be defined as a SingleAxis group. The master positioner can be a member of any motion group. So it is possible to define a Positioner as a slave of another positioner that is part of an XYZ group.

6.3.2 Specific Spindle Group Features

The Spindle Group is a single positioner group that enables continuous rotations with no limits and with a periodic position reset.

Master-Slave - In Master-Slave spindle mode the master and the slave group must be Spindle groups.

6.3.3 Specific XY Group Features

Line-Arc trajectories, XY mapping – These features are only available with XY groups. It is not possible for an XY group to perform a Spline or a PVT trajectory. Also, an XY group cannot be slaved to another group, however, any positioner of an XY group can be a master to a slaved SingleAxis group.

6.3.4 Specific XYZ Group Features

Spline trajectories, XYZ mapping – These features are only available with XYZ groups. It is not possible for an XYZ group to perform a Line-Arc or a PVT trajectory. Also, an XYZ group cannot be slaved to another group, however, any positioner of an XYZ group can be a master to a slaved SingleAxis group.

6.3.5 Specific MultipleAxes Features

PVT trajectories – PVT trajectories are only available with MultipleAxes groups. It is not possible for a MutipleAxes group to perform a Line-Arc or a Spline trajectory. Also, a MultipleAxes group cannot be slaved to another group. However, any positioner of a MultipleAxes group can be a master to a slaved SingleAxis group.

6.4 Native Units

The XPS controller supports user-defined native units like µm, inches, degrees or arcsecs. The units for each positioner are set in the configuration file where the parameter EncoderResolution indicates the number of units per encoder count. When using the XPS controller with Newport stages, this part of the configuration is done automatically. Once defined, all motions, speeds and accelerations can be commanded in the same native unit without any math needed. All other parameters like stage travel, maximum speed and all compensations are defined on the same scale as well. This is a great advantage compared to other controllers that can be commanded only in multiples of encoder counts, which can be an odd number.

In the XPS controller there are 4 types of position information for each positioner: TargetPosition, SetpointPosition, FollowingError and CurrentPosition. These are described as follows:

The **CurrentPosition** is the current physical position of the positioner. It is equal to the encoder position after all compensations (backlash, linear error and mapping) have been taken into account.

The **SetpointPosition** is the theoretical position commanded to the servo loop. It is the position where the positioner should be, during and after the end of the move.

The **FollowingError** is the difference between the CurrentPosition and the SetpointPosition.

The **TargetPosition** is the position where the positioner must be after the completion of a move.

When the controller receives a new motion command after the previous move is completed, a new TargetPosition is calculated.

This new target is received as an argument for absolute moves. For relative moves, the argument is the length of the move and the new target is calculated as the addition of the current target and the move length. Then the profiler of the XPS calculates a set of SetpointPositions to determine where the positioner should be at each given time.

When the positioner is controlled by a digital servo loop with a PID corrector, part of the signals sent to the motor of the positioner is a function of the following error. Part of this function is the integral gain of the PID filter that requires a following error equal to zero to reach a constant value.

The encoder in the positioner delivers a discrete signal (encoder counts). Take the example of an encoder with a resolution of 1 and a target position equal to 1.4. The real position cannot reach the value of the target position (1 or 2 instead of 1.4), so the following error will never be equal to zero (closest values are +0.6 and -0.4). Thus, due to the integral gain of the PID filter, the system will never settle, but will oscillate between the positions 1 and 2.

The XPS controller avoids this instability while allowing the use of native units instead of encoder counts by using a rounded value of the TargetPosition to calculate the motion profile and a rounded value for the following error. But the non-rounded value of the TargetPostion will be stored as final position, so that there is no accumulation of errors due to rounding, in case of successive relative moves.

To understand the difference, consider a positioner with a resolution of 1 that is at the position 0. This positioner receives a relative motion command of 10.4. At the end of the motion the CurrentPosition will be 10 and the SetpointPosition will be 10, but the TargetPosition will be 10.4. The positioner then receives the same relative motion command again. At the end of this motion the CurrentPosition will be 21, the SetpointPosition will be 21 and the TargetPosition will be 20.8.

NOTE

When an application requires a sequence of small incremental motion of constant step size close to the encoder resolution, make sure that the commanded incremental motion is equal to a multiple of encoder steps.

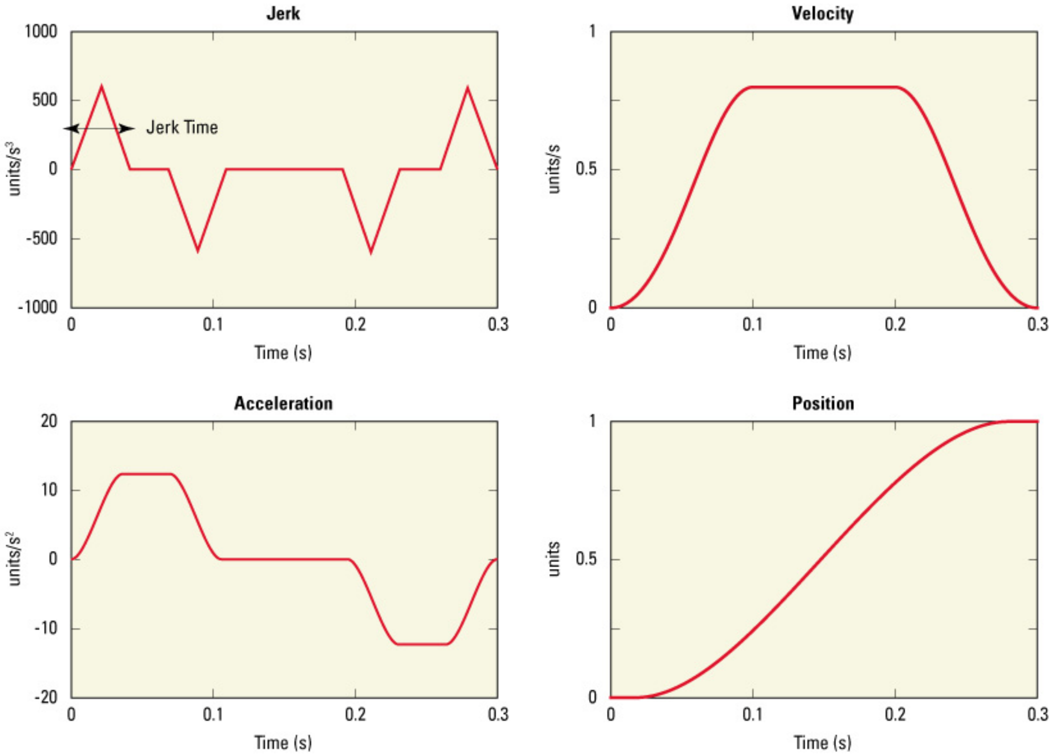
The TargetPosition, SetpointPosition, CurrentPositon and FollowingError can be queried from the controller using the appropriate function calls.

7.0 Motion

7.1 Motion Profiles

Motion commands refer to strings sent to a motion controller that will initiate a motion. The XPS controller provides several modes of positioning from simple point-to-point motion to the most complex trajectories. On execution of a motion command, the positioner moves from the current position to the desired destination. The exact trajectory for the motion is calculated by a motion profiler. So the motion profiler defines where each of the positioners should be at each point in time. There are details worth mentioning about the motion profiler in the XPS controller:

In a classical trapezoidal motion profiler (trapezoidal velocity profile), the acceleration is an abrupt change. This sudden change in acceleration can cause mechanical resonance in a dynamic system. In order to eliminate the high frequency portion of the excitation spectrum generated by a conventional trapezoidal velocity motion profile, the XPS controller uses a sophisticated SGamma motion profile. Figure 16 shows the acceleration, velocity and position plot for the SGamma profile.



Displacement: 150 e^{-3} units
Maximum velocity: 0.8 units/s
Maximum acceleration: 12 units/s^2
Minimum jerk time: 0.004 s
Maximum jerk time: 0.04 s

Notice: The minimum displacement lasts at least 4 times the minimum jerk time.

Figure 16: SGamma motion profile.

The SGamma motion profile provides better control of dynamic systems. It allows for perfect control of the excitation spectrum that a move generates. In a multi-axes system this profile gives better control of each axis independently, but also allows control of the cross-coupling that are induced by the combined motion of the axes. As shown in figure 17, the acceleration plot is parabolic. The parabola is controlled by the jerk time (jerk being the derivative of the acceleration). This parabolic characteristic of the acceleration results in a much smoother motion. The jerk time defines the time needed to reach the necessary acceleration. One feature of the XPS controller is that it automatically adapts the jerk time to the step width by defining a minimum and a maximum jerk time. This auto-adaptation of the jerk time allows a perfect adjustment of the system's behavior with different motion step sizes.

NOTE

Because of jerk-controlled acceleration, any move has a duration of at least four times the jerk time.

For the XPS controller, the following parameters need to be configured for the SGamma profile:

- MaximumVelocity (units/s)
- MaximumAcceleration (units/s^2)
- EmergencyDecelerationMultiplier (Applies to Emergency Stop)
- MinimumJerkTime (s)
- MaximumJerkTime (s)

The above parameters are set in the stages.ini file for a positioner. When using the XPS controller with Newport stages, these parameters are automatically set during the configuration of the system.

The velocity, acceleration and jerk time parameters is modified by the function **PositionerSGammaParametersSet**().Note that for continuity reason, the effective maximum velocity of the motion must be adjusted and may not be exactly the value defined by the parameter MaximumVelocity. For motion where a very accurate value of the velocity is needed, the length of the displacement has to be adjusted to the value given by the API **SGammaExactVelocityAjustedDisplacementGet**.

Example

PositionerSGammaParametersSet (MyGroup.MyStage, 10, 80, 0.02, 0.02)

This function sets the positioner "MyStage" velocity to 10 units/s, acceleration to 80 units/s^2 and minimum and maximum jerk time to 0.02 seconds. The set velocity and acceleration must be less than the maximum values set in the stages.ini file. These parameters are not saved if the controller is shut down. After a re-boot of the controller, the parameters will retain the values set in the stages.ini file.

In actual use, the XPS places a priority on the displacement position value over the velocity value. To reach the exact position, the speed of the positioner may vary slightly from the value set in the stages.ini file or by the **PositionerSGammaParametersSet** function. So the drawback of the SGamma profile is that the velocity used during the move can be a little bit different from the velocity defined in the parameters. For example, the exact velocity will change when the move distance is changed, move 100 mm, then 100.001 mm then 100.011 mm. There will be some changes to the commanded velocity. This change can be ignored for many applications except where an accurate time synchronization during the motion is required.

The function, **PositionerSGammaExactVelocityAdjustedDisplacementGet(),** can be used as described below to achieve the exact desired speed in applications that require an accurate value of the velocity during a move. In this case, the velocity value is adhered to, but the target position may be slightly different from the one required. In other words, according to the application requirements, the user can choose between very accurate positions or very accurate velocities.

Example

PositionerSGammaExactVelocityAdjustedDisplacementGet (MyGroup.MyStage, 50.55, ExactDisplacement)

This function returns the exact displacement for that move with the exact constant velocity set shown in the example above (10 mm/s). The result is stored in the variable ExactDisplacement, for instance 50.552.

GroupMoveAbsolute (MyGroup.MyStage, 50.552)

In the above example, for a position of 50.55 mm, the command returns a value of 50.552. This means that in order for the positioner "MyStage" to achieve the desired velocity in the most accurate way, the commanded position should be 50.552 mm instead of 50.55 mm.

The XPS can report two different positions. The first one is the SetpointPosition or theoretical position. This is the position where the stage should be according to the profile generator.

The second position is the CurrentPosition. This is the actual position as reported by the positioner's encoder after taking into account all compensation. The relationship between the SetpointPosition and the CurrentPosition is as follows:

Following error = SetpointPosition - CurrentPosition

The functions to query the SetpointPosition and the CurrentPosition values are:

GroupPositionCurrentGet() **and GroupPositionSetpointGet**()

7.2 Home Search

Home search is a specific motion process. Its goal is to define a reference point along the course of travel accurately and repeatably. The need for this absolute reference point is twofold. First, in many applications, it is important to know the exact position in space, even after a power-off cycle. Secondly, to prevent the motion device from hitting a travel obstruction set by the application (or its own hardware travel limits), the controller uses software limits. To be efficient, the software limits must be referenced accurately to the home before running the application.

After motor initialization, any motion group must first be homed or referenced before any further motion can be executed. Here, homing refers to a predefined motion process that moves a stage to a unique reference position and defines this as Home. Referencing refers to a group state that allows the execution of different motions and the setting of the position counters to any value (see next section for details). The referencing state provides flexibility for the definition of custom home search and system recovery processes. It should only be used by experienced users.

A number of hardware solutions may be used to determine the position of a motion device, the most common are incremental encoders. By definition, these encoders can only measure relative position changes and not absolute positions. The controller keeps track of position changes by incrementing or decrementing a dedicated counter according to the information received from the encoder. Since there is no absolute position information, position "zero" is where the controller was powered on (and the position counter was reset).

To determine an absolute position from incremental encoders, the controller must use a reference position that is unique to the entire travel, called a home switch or origin switch, usually in conjunction with an index pulse.

An important requirement is that this switch must have the same resolution as the encoder pulses.

If the motion device uses a linear scale as a position encoder, the home switch is usually placed on the same scale and read with the same resolution.

If, on the other hand, a rotary encoder is used, homing becomes more complicated. To have the same resolution, a mark on the encoder disk could be used (called index pulse), but because the mark repeats every revolution, it does not define a unique point over the entire travel. An origin switch, on the other hand, placed in the travel of the motion device is unique, but typically is not precise or repeatable enough. The solution is to use both in a dedicated search algorithm as follows.

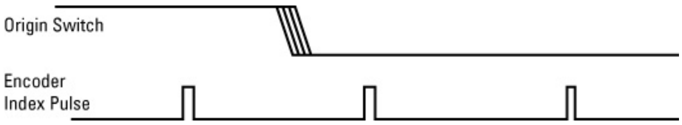


Figure 17: Home (Origin) switch and encoder index pulse.

A Home switch (Figure 17) separates the entire travel in two areas: one has a high level and the other has a low level. The most important part is the transition between the two areas. Just by looking at the origin switch level, the controller knows already on which side of the transition the positioner is and which direction to start the homing process.

The task of the home search process is to define one unique index pulse as the absolute position reference. This is first done by finding the home switch transition and then the very first index pulse (Figure 18).

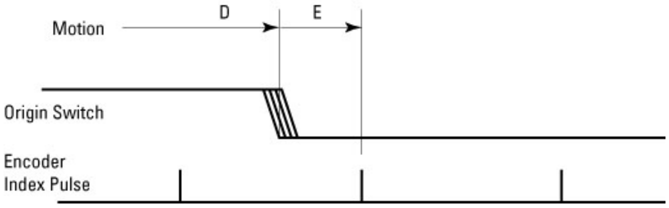


Figure 18: Slow-Speed Origin Switch Search.

Labeling the two motion segments D and E, the controller searches for the origin switch transition in D and for the index pulse in E. To guarantee the best repeatability possible, both D and E segments must perform at a very low speed and without stopping in between.

The homing process described above has a drawback. At low search speeds, the process could take a very long time if the positioner happens to start from the one end of travel. To speed things up, the positioner is moved fast until it is in the vicinity of the origin switch and then performs the two slow motions, D and E, at half the home search velocity. The new sequence is shown in Figure 19.

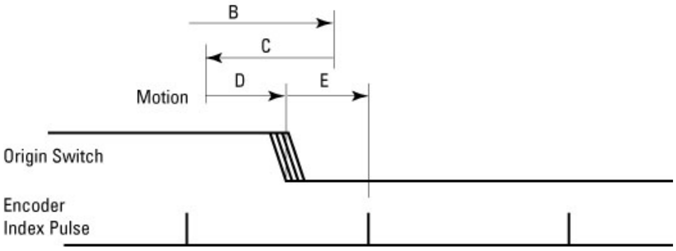


Figure 19: High/Low-Speed Home (Origin) Switch Search.

Motion segment B is performed at the pre-programmed home search speed. When the home switch transition is encountered, the motion device stops (with an overshoot), reverses direction and searches for the switch transition again, this time at half the speed (segment C). Once the switch transition is encountered, it stops again with an overshoot, reverses direction and executes D and E with one tenth of the programmed home search speed.

In the case when the positioner starts from the other end of the home switch transition, the routine is shown in Figure 20.

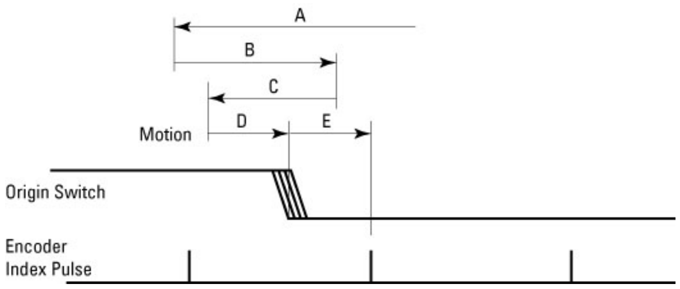


Figure 20: Home (Origin) Search from Opposite Direction.

The positioner moves at the home speed up to the home switch transition (segment A) and then executes segments B, C, D and E as in Figure 20.

This home search process guarantees that the last segment, E, is always performed in the positive direction of travel and at the same reduced speed. This method ensures an precise and repeatable reference position.

There are 7 different home search processes available in the XPS controller:

1. **MechnicalZeroAndIndexHomeSearch** is used when the positioner has a hardware home switch plus a zero index from the encoder. This process is the default for most Newport standard stages.
2. **MechanicalZeroHomeSearch** is used with positioners that have a hardware home switch but with no zero index from the encoder.
3. **IndexHomeSearch** is used with positioners that have a home index, but with no hardware home switch signal. In this process, the positioner initially moves in the positive direction to find the index. When a + limit switch is detected, the direction of motion reverses until the index is found.
4. **CurrentPositionAsHome** is used when the positioner has no home switch or index. This process will keep the positioner's home at its current location. Setting the home too close to the EOR could generate unwanted emergency stops. Start with around 50 MIM (Minimum Incremental Movement) units, but an optimum distance may be determined by trial and error, depending on the stage.
   This feature can also be used to set home arbitrarily and bypass a home switch.
5. **MinusEndOfRunAndIndexHomeSearch** uses the positioner's minus end-of-run limit as a hardware home switch and a zero index from the encoder. This process is comparable to MechanicalZeroAndIndexHomeSearch, but uses the minus end-of-run limit signal as hardware home switch and moves in the positive direction until the Index is reached. Otherwise, it will reach the positive limit or a timeout will occur. The positioner homes to a position that is different from the MechanicalZeroAndIndexHomeSearch location.
6. **MinusEndOfRunHomeSearch** uses the positioner's minus end-of-run limit for homing. Note that the emergency stop at the negative limit is disabled during homing.
7. **PlusEndOfRunHomeSearch** uses the positioner's plus end-of-run limit for homing and the emergency stop at the positive limit is disabled during homing.

The home search process is set up in the stages.ini file. When using the XPS controller with Newport ESP-compatible stages, this setting is done automatically with the configuration of the system. The home search velocity, acceleration and time-out are also set up in the stages.ini file.

Each motion group can either be homed "together" or "sequentially", meaning all positioners belonging to that group home at the same time in parallel or all the positioners home one after the other, respectively. This option is also set up in the system.ini file or during configuration.

A Home search can be executed with all motion groups and any motion group MUST be homed before any further motion can be executed. To home a motion group that is in a "ready" state, that motion group must first be "killed" and then "re-initialized".

Example

This is the sequence of functions that initialize and home a motion group.

GroupInitialize (MyGroup)

GroupHomeSearch (MyGroup)

…

GroupKill (MyGroup)

7.3 Referencing State

The predefined home search processes described in the previous section might not be compatible with all motion devices or might not be always executable. For instance, if there is a risk of collision during a standard home search process. In other situations, a home search process might not be desirable. For example, to ensure that the stages have not moved, the current positions are stored into memory. In this case, it is sufficient to reinitialize the system by setting the position counters to the stored position values.

For these special situations, the XPS controller's referencing state as in alternative to the predefined home search processes.

NOTE

The Referencing state should be only used by experienced users. Incorrect use could cause equipment damage.

The Referencing state is a parallel state to the homing state, see the state diagram on page 103, Figure 21. To enter the referencing state, send the function **GroupReferencingStart(GroupName)** while the group is in the NOT REFERENCED state.

In the Referencing state, the function **GroupReferencingActionExecute(PositionerName, Action, Sensor, Parameter)** will perform certain actions like moves, position latches of reference signal transitions, or position resets. The function **PositionerSGammaParametersSet(PositionerName)** can be used to change the velocity, acceleration and jerk time parameters.

To leave the referencing state, send the function **GroupReferencingStop(GroupName)**. The Group will then be in the HOMED state, state number 11.

The syntax and function of the function **GroupReferencingActionExecute(PositionerName, Action, Sensor, Parameter)** will be discussed in detail. With this function, there are four parameters to specify:

- PositionerName is the name of the positioner on which this function is executed.
- Action is the type of action that is executed. There are eight actions that can be distinguished into three categories: Moves that stop on a sensor event, moves of certain displacement, and position counter reset categories.
- Sensor is the sensor used for those actions that stop on a sensor event. It can be MechanicalZero, MinusEndOfRun, or None.
- Parameter is either a position or velocity value and provides further input to the function.

The following table summarizes all possible configurations:

	Sensor			Parameter	
Action	**MechanicalZero**	**MinusEndOfRun**	**None**	**Position**	**Velocity**
LatchOnLowToHighTransition	■	■			■
LatchOnHighToLowTransition	■	■			■
LatchOnIndex			■		■
LatchOnIndexAfterSensorHighToLowTransition	■	■			■
SetPosition			■	■	
SetPositionToHomePreset			■		
MoveToPreviouslyLatchedPosition			■		■
MoveRelative			■	■	

7.3.1 Move on sensor events

The “move on sensor events” starts a motion at a defined velocity, latches the position when a state transition of a certain sensor is detected, then stops the motion. There are four possible actions under this category:

- **LatchOnLowToHighTransition**
- **LatchOnHighToLowTransition**
- **LatchOnIndex**
- **LatchOnIndexAfterSensorHighToLow**

With **LatchOnLowToHighTransition** and **LatchOnHighToLowTransition**, latching happens when the right transition on the defined sensor occurs. The sensor can be latched to either **MechanicalZero**, **MinusEndOfRun** and **PositiveEndOfRun** when supported by the hardware, refer to section 7.2: “Home Search” to know which hardware supports the function. With **LatchOnIndex** and **LatchOnIndexAfterSensorHighToLow**, latching happens on the index signal. With **LatchOnIndexAfterSensorHighToLow,** latching happens on the first index after a high to low transition at the defined sensor (**MechanicalZero** or **MinusEndOfRun**). Because of the dedicated hardware circuits used for the position latch, there is essentially no latency between sensor transition detection and position acquisition.

In all cases, motion stops after the latch. However, this means that the stopped position doesn’t rest on the sensor transition, but at some short distance from it. To move exactly to the position of the sensor transition, use the action **MoveToPreviouslyLatchedPosition**.

The latch does not change the current position value. In order to set the current position value, use the action **SetPosition** or **SetPositionToHomePreset**, for instance, after a **MoveToPreviouslyLatchedPosition**.

In the Referencing state, the limit switch safeties (emergency stop) are still enabled until the **MinusEndOfRun** sensor is specified with a **GroupReferencingActionExecute**() function. When specified, the limit switch safeties are disabled and will only be re-enabled with the function **GroupReferencingStop**().

The Parameter has a sign, if it is assigned as velocity (floating point). This means that the direction of motion is dictated by the sign of the velocity parameter.

7.3.2 Moves of Certain Displacements

These two move commands which don’t use the same parameters, are explained below.

- **MoveRelative**

  The action **MoveRelative** commands a relative move of a positioner similar to the function **GroupMoveRelative**. However, the function **GroupMoveRelative** is not available in the Referencing state. The relative move is specified by a positive or negative displacement. The move is done with the SGamma profiler. The speed and acceleration are the default values, or the last value defined by either a move on sensor event, a **MoveToPreviouslyLatchedPosition**, or a **PositionerSGammaParametersSet**.

- **MoveToPreviouslyLatchedPosition**

  This action moves the positioner to the last latched position, see section 7.3.1: “Move on sensor events“ for details. It verifies there was a position latched since this last **GroupReferencingStart** call. This is important because an old latched position can still be in memory from a previous home search or referencing. And moving to this previous latched position could have unexpected results. The move is done with the SGamma profiler. The speed is specified by a parameter. The acceleration is the default value, or the last value defined by a **PositionerSGammaParametersSet**.

7.3.3 Position Counter Resets

“Position counter resets” sets the current position to a certain value. There are two options: **SetPosition** and **SetPositionToHomePreset**. The main use of these actions is when the positioner is at a well defined reference position after a **MoveToPreviouslyLatchedPosition** action.

Another use of this action is for a “soft” system start by Referencing a group to a known set position, without executing a home search process, for example. In this case, a suggested sequence of functions follows:

GroupReferencingStart(GroupName)

GroupReferencingActionExecute(PositionerName, “SetPosition”, “None”, KnownCurrentPosition)

GroupReferencingStop(GroupName)

SetPosition sets the current position to a value defined by a parameter. **SetPositionToHomePreset** sets the current position to the **HomePreset** value stored in the stages.ini configuration file. It is equivalent to a **SetPosition** of the same positioner to the **HomePreset** value.

It is important that all positioners of a motion group are referenced to a position using the **SetPosition** or **SetPositionToHomePreset** before leaving the Referencing state (see example on page 94).

7.3.4 State Diagram

The Referencing state is a parallel state to the homing state. It is between the NotReferenced state and the Ready state. Please see the state diagram below:

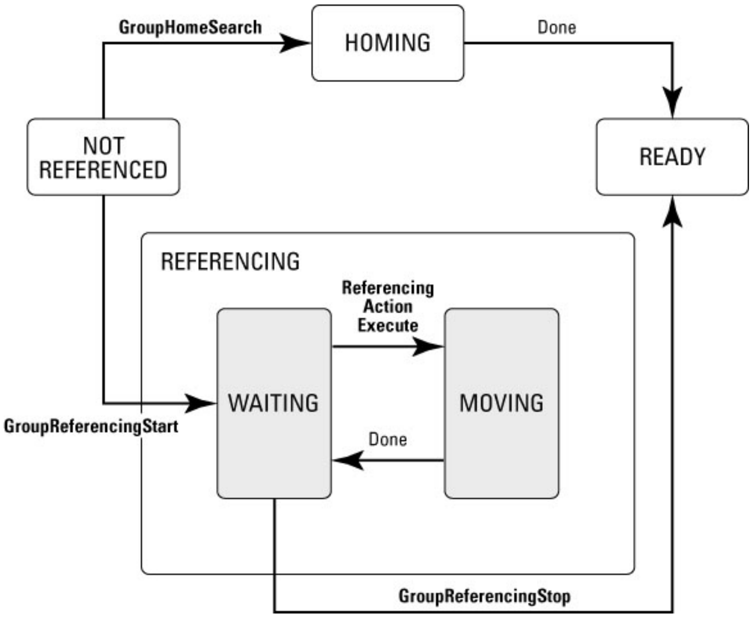


Figure 21: State diagram.

7.3.5 Example: MechanicalZeroAndIndexHomeSearch

The following sequence of functions has the same effect as the MechanicalZeroAndIndexHomeSearch:

```
GroupReferencingStart(GroupName)
PositionerHardwareStatusGet (PositionerName, &status)
if ((status & 4) == 0) { // 4 is the Mechanical zero mask on the hardware status
GroupReferencingActionExecute(PositionerName,
“LatchOnLowToHighTransition”, “MechanicalZero, -10) }
GroupReferencingActionExecute(PositionerName,
“LatchOnHighToLowTransition”, “MechanicalZero”, 10)
GroupReferencingActionExecute(PositionerName,
“LatchOnLowToHighTransition”, “MechanicalZero”, -5)
GroupReferencingActionExecute(PositionerName,
“LatchOnIndexAfterSensorHighToLow”, “MechanicalZero”, 5)
GroupReferencingActionExecute(PositionerName,
“MoveToPreviouslyLatchedPosition”, “None”, 5)
GroupReferencingActionExecute(PositionerName,
“SetPositionToHomepreset”, “None”, 0)
GroupReferencingStop(GroupName)
```

7.4 Move

A move is a point-to-point motion. On execution of a move command, the motion device moves from a current position to a desired destination (absolute move) or by a defined increment (relative move). During motion, the controller is monitoring the feedback of the positioner and is updating the output based upon the following error. The XPS controller’s position servo is updated at servo cycle rate, default value 8 kHz, and the profile generator at the profile generator rate, default value 2 kHz. These defaults values provide highly accurate closed loop positioning. Between the profiler and the corrector, there is a time-based linear interpolation to accommodate the different frequencies.

There are two types of moves that can be commanded: an absolute move and a relative move. For an absolute move, the positioner will move relative to the HomePreset position as defined in the stages.ini file. In most cases the HomePreset is 0, which makes the home position equal to the zero position of the positioner. For a relative move, the positioner will move relative to the current TargetPosition. In relative moves, it is possible to make successive moves that are not equal to a multiple of an encoder step without accumulating errors.

Absolute and relative moves can be commanded to positioners and to motion groups. When commanding a move to a positioner, only the position parameter for that positioner must be provided. When commanding a move to a motion group, the appropriate number of position parameters must be provided with the move command. For instance for a move command to an XYZ group, 3 position parameters must be defined.

When commanding a move to a motion group, all positioners of that group will move synchronously. For any move, the controller will always determine the shortest time within the positioner's parameters setup. All positioners will start and stop their motion at the same time. This type of motion is also known as linear interpolation.

The functions for absolute and relative motions are **GroupMoveAbsolute**() and **GroupMoveRelative**() respectively.

Example

A motion system consisting of one XY group called ScanTable and one SingleAxis group called FocusStage. ScanTable has two positioners, called ScanAxis and StepAxis.

…

GroupHomeSearch (ScanTable)

GroupHomeSearch (FocusStage)

After homing is completed…

GroupPositionCurrentGet (ScanTable, Pos1, Pos2)

… will return 0 to Pos1 and 0 to Pos2, assuming PresetHome = 0.

GroupPositionCurrentGet (FocusStage, Pos3)

Will return 0 to Pos3, assuming HomePreset = 0.

GroupMoveAbsolute (ScanTable, 100, 50)

GroupMoveAbsolute (ScanTable.StepAxis, -20)

The second move is only for one positioner of that group and can be only executed after the first move is completed. After all moves are completed…

GroupPositionCurrentGet (ScanTable, Pos1, Pos2)

… will return 100 to Pos1 and -20 to Pos2.

GroupMoveRelative (FocusStage, 1)

GroupMoveRelative (FocusStage, 1)

The second move can be only executed after the first move is completed. After all moves are completed…

GroupPositionCurrentGet (FocusStage, Pos3)

… will return 2 to Pos3.

The velocity, acceleration and jerk time parameters of a move are defined by the function PositionerSGammaParametersSet() (see also section 7.1). When the controller receives new values for these parameters during the execution of a move, it will not take these new values into account on the current move, but only on the following moves. To change the velocity or acceleration of a positioner during the motion, use the Jogging mode (see section 7.5).

A move can be stopped at any time with the function **GroupMoveAbort**() that accepts GroupNames and PositionerNames. It is important to note, however, that the function **GroupMoveAbort(PositionerNames)** is accepted when the motion was commanded to the positioner, and not to the group. In the previous example, the function **GroupMoveAbort(ScanTable.ScanAxis)** is rejected for a motion that has been launched with **GroupMoveRelative(ScanTable, 100, 50)**. To stop this motion, send the function **GroupMoveAbort(ScanTable)**.

With XPS firmware 1.5.0 and higher, the XPS controller supports also asynchronous moves of several positioners belonging to the same motion group. The individual motion, however, needs to be managed by separate threads (see also section 17.3: “Running Processes in Parallel” for details).

7.5 Motion Done

The XPS controller supports two methods that define when a motion is completed (MotionDone): the theoretical MotionDone and the VelocityAndPositionWindow MotionDone. The method used is set in the stages.ini file. In theory, MotionDone is completed as defined by the profiler. However, it does not take into account the settling of the positioner at the end of the move. So depending on the precision and stability requirements at the end of the move, the theoretical MotionDone might not always be the same as the physical end of the motion. The VelocityAndPositionWindow MotionDone allows a more precise definition by specifying the end of the move with a number of parameters that take the settling of the positioner into account. In the VelocityAndPositionWindow MotionDone, the motion is completed when:

> | PositionErrorMeanValue | < | MotionDonePositionThreshold | AND | VelocityMeanValue | < | MotionDoneVelocityThreshold | is verified during the MotionDoneCheckingTime period.

The different parameters have the following meaning:

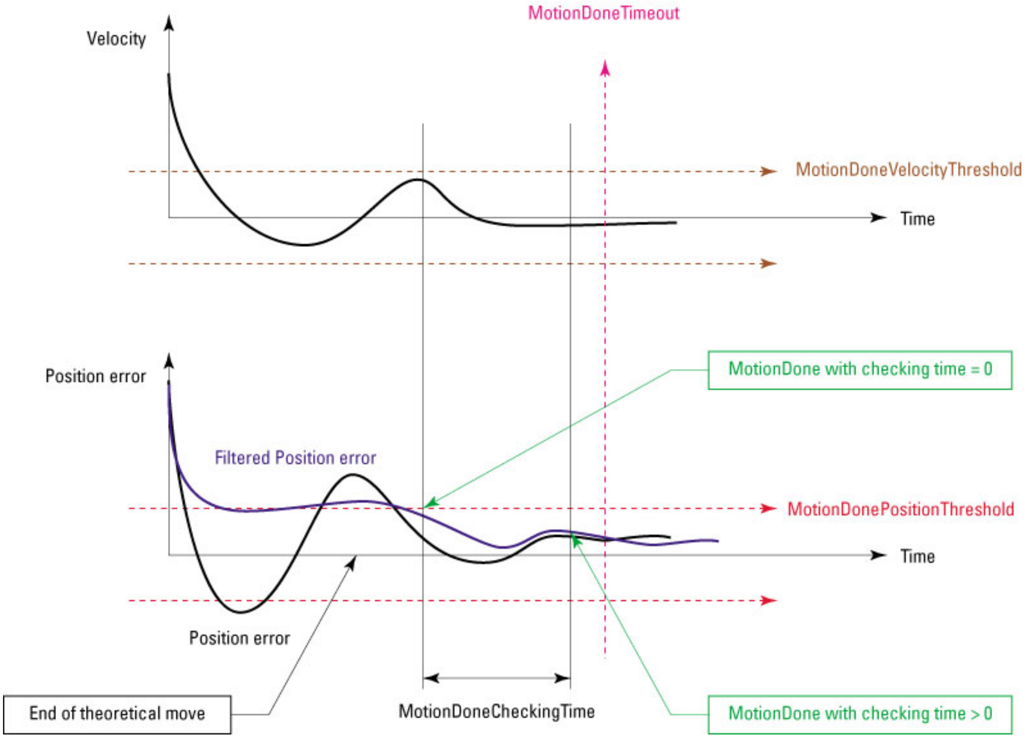


Figure 22: Motion done.

- **MotionDonePositionThreshold**: This parameter defines the position error window. The position error has to be within ± of this value for a period of MotionDoneCheckingTime to validate this condition.
- **MotionDoneVelocityThreshold**: This parameter defines the velocity window. The velocity at the end of the motion has to be within ± of this value for a period of MotionDoneCheckingTime to validate this condition.
- **MotionDoneCheckingTime**: This parameter defines the period during which the conditions for the MotionDonePositionThreshold and the MotionDoneVelocityThreshold must be true before setting the motion done.
- **MotionDoneMeanPeriod**: A sliding mean filter is used to attenuate the noise for the position and velocity parameters. The MotionDoneMeanPeriod defines the duration for calculating the sliding mean position and velocity. The mean position and velocity values are compared to the threshold values as defined above. This parameter is not illustrated on the graph.
- **MotionDoneTimeout**: This parameter defines the maximum time the controller will wait from the end of the theoretical move for the MotionDone condition, before sending a MotionDone time-out.

Important:

The XPS controller can only execute a new move on the same positioner or on the same motion group when the previous move is completed (MotionDone) and when the positioner or the motion group is again in the ready state.

The XPS controller allows triggering an action when the motion is completed (MotionDone) by using the event MotionEnd. For further details see chapter 11.0.

The functions **PositionerMotionDoneGet**() and **PositionerMotionDoneSet**() allow reading and modifying the parameters for the VelocityAndPositionWindow MotionDone. These parameters are only taken into account when the MotionDoneMode is set to VelocityAndPositionWindow in the stages.ini.

Example

Modifications of the MotionDoneMode can be made only manually in the stages.ini file. The stages.ini file is located in the config folder of the XPS controller, see section 4.32: “FTP (File Transfer Protocol) Connection” for details. Stage parameters can also be modified from the website, in Administrator mode, STAGES menu, Modify submenu.

Make a copy of the stages.ini file to the PC. Open the file with any text editor and modify the MotionDoneMode parameter of the appropriate stage to VelocityAndPositionWindow, and set the following parameters:

```
;--- Motion done
MotionDoneMode = VelocityAndPositionWindow  ; instead of Theoretical
MotionDonePositionThreshold = 4             ; units
MotionDoneVelocityThreshold = 100           ; units/s
MotionDoneCheckingTime = 0.1                ; seconds
MotionDoneMeanPeriod = 0.001                ; seconds
MotionDoneTimeout = 0.5                     ; seconds
```

Replace the current stages.ini file on the XPS controller with this modified version (make a copy of the old .ini file first). Reboot the controller. To apply any changes to the stages.ini or system.ini, the controller has to reboot.

Use the following functions:

GroupInitialize(MyGroup)

GroupHomeSearch(MyGroup)

PositionerMotionDoneGet(MyGroup.MyPositioner)

This function returns the parameters for the VelocityAndPositionWindow Motion done previously set in the stages.ini file, so 4, 100, 0.1, 0.001 and 0.5.

PositionerMotionDoneSet(MyGroup.MyPositioner, PositionThresholdNewValue, VelocityThresholdNewValue, CheckingTimeNewValue, MeanPeriodNewValue, TimeoutNewValue)

This function replaces the parameters with the newly entered values. If this function is not executed, the default setting from the .ini file is used.

7.6 JOG

Jog is an indeterminate motion defined by velocity and acceleration. Unlike a **GroupMoveAbsolute**() or a **GroupMoveRelative**(), the end of the motion is not defined by a target position. It can be best described by a “go”-command with a definition how fast, but not how far.

In Jog mode, the speed and acceleration of a motion group can be changed on-the-fly to accommodate varying situations. This is not possible with a **GroupMoveAbsolute**() or a **GroupMoveRelative**() which are defined moves. Practical examples for Jog are with tracking systems or coordinate transformations where the speed or acceleration of the jogging group is modified depending on the position or speed of the other motion groups or based on an analog input value.

The Jog mode can be enabled using the function **GroupJogModeEnable**() and is available to all motion groups. Once this mode is enabled, the motion parameters can be set using the command **GroupJogParameterSet**() which is applicable to positioners and to motion groups. To query the maximum jog velocity and acceleration values for a positioner, use **PositionerJogMaximumVelocityAndAccerationGet**(). To exit the Jog mode, first set the velocity to zero and then send the function **GroupJogModeDisable**().

Examples

For a single axis group:

GroupJogModeEnable (MySingleGroup)

Enables the Jog mode.

GroupJogParameterSet (MySingleGroup, 5, 20)

The single stage starts moving with a velocity of 5 units per second and an acceleration of 20 units per second^2.

GroupJogParameterSet (MySingleGroup, -5, 20)

The single stage starts moving in the reverse direction with the same velocity and same acceleration.

GroupJogParameterSet (MySingleGroup, 0, 20)

The single stage stops moving, its velocity being 0 units per second.

GroupJogModeDisable (MySingleGroup)

Disables the Jog mode.

For an XY group:

GroupJogModeEnable (MyXYGroup)

Enables the Jog mode.

GroupJogParameterSet (MyXYGroup, 5, 20, 10, 40)

The X axis and Y axis start moving with a velocity of 5 and 10 units per second and an acceleration of 20 and 40 units per second^2 respectively.

GroupJogParameterSet (MyXYGroup, 0, 20, 0, 40)

Both stages stop moving, their velocities being 0 units per second.

To apply new parameters to only one stage, use the following function:

GroupJogParameterSet (MyXYGroup.XPositioner, 5, 20)

Only the X axis starts moving with a velocity of 5 units per second and an acceleration of 20 units per second^2.

GroupJogParameterSet (MyXYGroup.XPositioner, 0, 20)

The X axis stage stops moving, its velocity being 0 units per second.

GroupJogModeDisable (MyXYGroup)

Disables the Jog mode.

In Jog mode, the profiler uses the CurrentPosition and the defined velocity and acceleration to calculate a new Setpoint position every 0.4 ms. These new Setpoint positions are then transferred to the corrector loop which runs every 0.1 ms. To accommodate the different frequencies between the profiler and the corrector, a linear interpolation between the new Setpoint and the previous Setpoint is done. Worst case, a new velocity and acceleration can be executed only every 0.4 ms. In Jog mode, the profiler uses a trapezoidal motion profile (see also section 7.1 for further details on motion profiles).

7.7 Master Slave

In master slave mode, any motion axis can be electronically geared to another motion axes, or a single master with multiple slaves. The gear ratio between the master and the slave is user defined. During motion, all axes compensations of the master and the slave are taken into account.

The slave must be a SingleAxis group. The master can be a positioner from any group. The Master slave relation is set by the function **SingleAxisSlaveParametersSet().**

The Master slave mode is enabled by the function **SingleAxisSlaveModeEnable**(). To enable the Master slave mode, the Slave group must be in the ready state. The Master group can be in the not-referenced or ready state.

Example 1

This example shows the sequence of functions used to set-up a master-slave relation between two axes that are not mechanically joined (meaning the two axis can move independently):

GroupInitialize (SlaveGroup)

GroupHomeSearch (SlaveGroup)

GroupInitialize (MasterGroup)

GroupHomeSearch (MasterGroup)

…

SingleAxisSlaveParametersSet (SlaveGroup, MasterGroup.Positioner, Ratio)

SingleAxisSlaveModeEnable (SlaveGroup)

GroupMoveRelative (MasterGroup.Positioner, Displacement)

…

SingleAxisSlaveModeDisable (SlaveGroup)

Example 2

This example shows the sequence of functions used to set-up a Master slave relation **between two axes that are mechanically joined**. Different from example 1, all motions, including the motion done during the home search routine, are performed synchronously.

Important: First, set the HomeSearchSequenceType of the Slave group's positioner to CurrentPositionAsHome in the stages.ini and reboot the XPS controller.

GroupInitialize (SlaveGroup)

GroupHomeSearch (SlaveGroup)

GroupInitialize (MasterGroup)

SingleAxisSlaveParametersSet (SlaveGroup, MasterGroup.Positioner, Ratio)

SingleAxisSlaveModeEnable (SlaveGroup)

GroupHomeSearch (MasterGroup)

…

GroupMoveRelative (MasterGroup.Positioner, Displacement)

NOTE

The slave positioners should have similar capabilities as the master positioner in terms of velocity and acceleration. Otherwise the full capabilities of the master or the slave positioners may not be utilized.

7.8 Analog Tracking

Analog tracking controls the position or velocity of a motion group via external analog inputs. Analog tracking is available with all motion groups. To enable this mode, first set the tracking parameters of the positioners belonging to that motion group. Then enable tracking while the motion group is homed (in ready state after homing). In analog tracking mode, the analog inputs are filtered by a first order low-pass filter. Its cut-off frequency is defined by the parameter "TrackingCutOffFrequency" given in the section "profiler" of the stage.ini parameter file.

To set or get the tracking parameters, use the following functions:

PositionerAnalogTrackingPositionParametersSet()

PositionerAnalogTrackingPositionParametersGet()

…

PositionerAnalogTrackingVelocityParametersSet()

PositionerAnalogTrackingVelocityParametersGet()

The functions PositionerAnalogTrackingPositionParametersSet() and PositionerAnalogTrackingVelocityParametersSet() define the maximum velocity and acceleration used during analog tracking.

7.8.1 Analog Position Tracking

The parameters that can be set for analog position tracking are the GPIO Name, scale and offset. The GPIO Name denotes which connector and pin number the analog signal will be input. The scale and the offset are used to calibrate the output position in the following way:

Position = InitialPosition + (AnalogValue - Offset) * Scale

Typical applications of analog position tracking are for beam stabilization, tracking systems, auto focusing sensors or alignment systems. When connecting a function generator to the GPIO input, analog tracking provides an easy way to make cyclical or sinusoidal motion, for example.

Example

Following is an example that shows the sequence of functions used to setup Analog Position Tracking:

GroupInitialize (Group)

GroupHomeSearch (Group)

…

PositionerAnalogTrackingPositionParameterSet (Group.Positioner, GPIO2.ADC1, Offset, Scale, Velocity, Acceleration)

GroupAnalogTrackingModeEnable (Group, “Position”)

…

GroupAnalogTrackingModeDisable (Group)

7.8.2 Analog Velocity Tracking

The parameters that can be set for analog velocity tracking are the GPIO Name, offset, scale, deadband threshold and order. The relationship among offset, scale, deadband and order is illustrated in Figure 23.

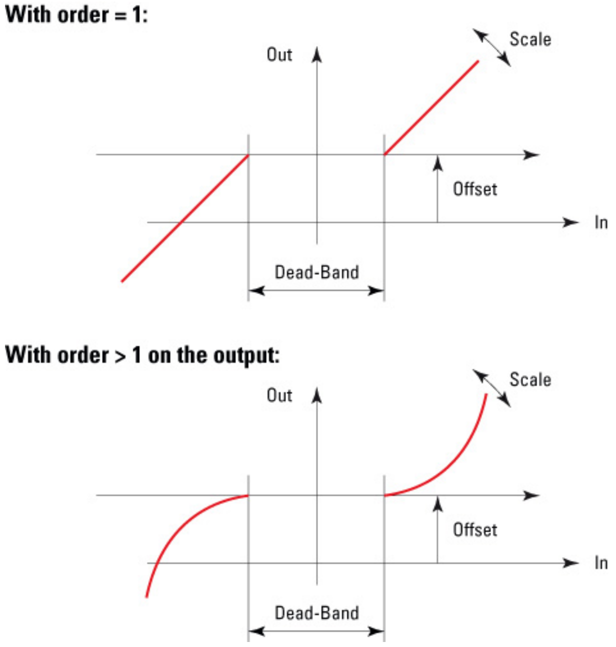


Figure 23: The relationship among Offset, Scale, Dead Band & Order.

The tracking velocity calculates as follows:

- AnalogInput is the voltage input at the GPIO
- AnalogGain refers to the AnalogGain setting of the analog input
- Offset, Order, DeadBandThreshold, and scale are defined with the function PositionerAnalogTrackingVelocityParametersSet
- MaxADCAmplitude, InputValue, OutputValue are internally-used parameters only

```
InputValue = AnalogInput - Offset
if (InputValue >= 0) then
InputValue = InputValue - DeadBandThreshold
    if (InputValue < 0) then InputValue = 0
else
InputValue = InputValue + DeadBandThreshold
    if (InputValue > 0) then InputValue = 0
OutputValue = (|InputValue|/ MaxADCAmplitude) * Order
Velocity = Sign(InputValue) * OutputValue * Scale * MaxADCAmplitude
```

In the dead band region there is no motion. If the order is set to 1, then the velocity is linear with respect to the input voltage.

If order is set greater than 1, then the velocity response is polynomial with respect to the input voltage. This makes the change in velocity more gradual and more sensitive in relation to the change in voltage.

A good example for using analog velocity tracking is for an analog joystick.

Example

Following is an example that shows the sequence of functions used to set-up Analog Velocity Tracking:

GroupInitialize (Group)

GroupHomeSearch (Group)

…

PositionerAnalogTrackingVelocityParameterSet (Group.Positioner, GPIO2.ADC1, Offset, Scale, DeadBandThreshold, Order, Velocity, Acceleration)

GroupAnalogTrackingModeEnable (Group, “Velocity”)

…

GroupAnalogTrackingModeDisable (Group)

8.0 Trajectories

The XPS controller supports 4 different types of trajectories:

The Line-arc trajectory is a trajectory defined by a combination of straight and curved segments. It is available only for positioners in XY groups. The major benefit of a Line-arc trajectory is the ability to maintain constant speed (speed being the scalar of the trajectory velocity) throughout the entire path, excluding the acceleration and deceleration periods. The trajectory is user defined in a text file that is sent to the controller via FTP. Once defined, the user executes a function to begin the trajectory and the XPS automatically calculates and executes the motion, including precise monitoring of the speed and acceleration all along the trajectory. Simply executing the same trajectory more than once results in continuous path contouring. A dedicated function performs a precheck of the trajectory which returns the maximum and minimum travel requirements per positioner as well as the maximum possible trajectory speed and trajectory acceleration that is compatible with the different positioner parameters.

The spline trajectory executes a Catmull-Rom spline (which is a 3rd order polynomial curve) on an XYZ group. The main requirements of a spline are to hit all points (except for the first and the last point that are only needed to define the start and the end of the trajectory) and to maintain a constant speed throughout the entire path (except during the acceleration and deceleration period). The definition and execution of the spline trajectory is similar to the Line-arc trajectory with similar functions for trajectory pre-checking.

The PVT-mode is the most complex trajectory and is only available with MultipleAxes group. In a PVT trajectory, each trajectory element is defined by the displacement and end speed of each positioner plus the move time for the element. When all elements are defined, the controller calculates the cubic function trajectory that will pass through all defined positions at the defined times and velocities. PVT is a powerful tool for any kind of trajectory with varying speeds and for trajectories with rotation stages or other nonlinear motion devices.

The PT-mode is based on a simpler definition of PVT trajectory and is only available with MultipleAxes group. In a PT trajectory, each trajectory element is defined by the displacement and move time for the element. When all elements are defined, the controller calculates the cubic function trajectory that will pass through all defined positions at the defined times. The output velocity of each element is defined by the firmware to avoid speed oscillations when successive elements are set with the same speed (DX/DT= constant).

8.1 Line-Arc Trajectories

8.1.1 Trajectory Terminology

Trajectory: defined as a continuous multidimensional motion path. Line-arc trajectories are defined in a two-dimensional XY plane. These are used with XY groups. The main requirement of a Line-arc trajectory is to maintain a constant speed (speed being the scalar of the vector velocity) throughout the entire path (except during the acceleration and deceleration periods).

Trajectory element (segment): an element of a trajectory is defined by a simple geometric shape, in this case a line or an arc segment.

Trajectory velocity: the tangential linear velocity (speed) along the trajectory during its execution.

Trajectory acceleration: the tangential linear acceleration used to start and end a trajectory. Trajectory acceleration and trajectory deceleration are equal by default.

8.1.2 Trajectory Conventions

When defining and executing a Line-arc trajectory, a number of rules must be followed:

- The motion group must be an XY group.
- All trajectories must be stored in the controller's memory under ..\public\trajectories (one file for each trajectory). Once a trajectory is started, it executes in the background allowing other groups or positioners to work independently and simultaneously.
- Each trajectory must have a defined beginning and end. Endless (infinite) trajectories are not allowed. Although, N-times (N defined by user) non-stop execution of the same trajectory is allowed. As the trajectory is stored in a file, the trajectory's maximum size (maximum elements number) is unlimited for practical purposes.
- Two types of Line-arc trajectory elements (segments) are available: lines Line(X,Y) and arcs Arc(R,A) (Radius, SweepAngle). Any Line-arc trajectory is a set of consecutive line or arc segments. The line segments are true linear interpolations y = A*x + B, the arc segments are true arcs of circles $(x - x0)^2 + (y - y0)^2 = R^2$.
- A Line-arc trajectory forms a continuous path, so each segment's final position is equal to the next segment's starting position. However, as the segment's tangential angles around the connection point of any two consecutive segments may not be continuous, there might be velocity discontinuities from one segment to next. For reference, this discontinuity is categorized as R0, wherein the position is continuous, but velocity is not. An excessive velocity discontinuity at joints can damage the stages, so the trajectory definition process must take this into account.
- Each Line-arc trajectory element is defined relative to the trajectory starting point. Every trajectory starting point has the coordinates (0,0), which has no relation to the zero position of the positioners. All trajectories physically start from the current X and Y positions of the XY group.

8.1.3 Geometric Conventions

The coordinate system of a Line-arc trajectory is an XY orthogonal system.

The X-axis of this system correlates to the XPositioner and the Y-axis correlates to the YPositioner of the XY group as defined in the system.ini.

The origin of the XY coordinate system is in the lower left corner, with positive values up and to the right.

All angles are measured in degrees, presented as floating point numbers. Angle origin and signs follow the trigonometric convention: positive angles are measured counter-clockwise.

8.1.4 Defining Line-Arc Trajectory Elements

A Line-arc trajectory is defined by a number of line and arc elements. The trajectory elements are executed in the same order as defined in the trajectory data file.

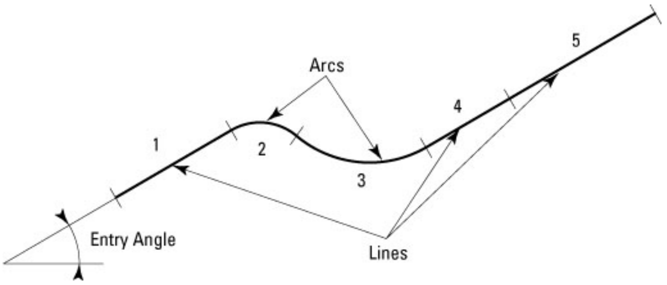


Figure 24: Line-arc trajectory example.

Figure 24 shows a trajectory example. Every trajectory must have a first element entry angle (called First Tangent) defined in the head of the trajectory data file. If the first element is a line, this parameter has no effect. If the first element is an arc, the entry angle is the tangent to the first point of the arc. Each trajectory element is identified by a number, starting from 1. The references for synchronizing external events with the trajectory execution are the starting and ending points of these elements.

Line and arc elements can be sequenced in any order. An arc is automatically placed by the controller so that its entry angle corresponds to the exit angle of the preceding element to ensure the continuity of the trajectory. But with every line segment, the user must choose the (X,Y) end-point in that way that the angle discontinuity to the previous segment does not exceed the maximum allowed angular discontinuity. The angular discontinuity is measured in degrees and is defined in the head of the trajectory data file. In theory, a trajectory can be defined only by straight lines, if two adjacent line segments have an angular difference smaller than the allowed angle of discontinuity, as shown in the Figure 25.

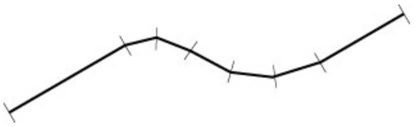


Figure 25: Contouring with linear lines only.

In practice this is not recommended since each angle of discontinuity corresponds to an instantaneous velocity change on both axes, which produces large accelerations. This can result in a shock to the stages and an increase in the following error. The larger the angle of discontinuity, the larger the shock and following error will be. Special consideration must be given to both these effects when increasing the maximum discontinuity angle from its default value.

8.1.5 Define Lines

A line element is defined by specifying the (X_i, Y_i) ending point.

The succeeding element's starting point is always the end point of the previous segment (X_{i-1}, Y_{i-1}).

Note that all line element positions are defined relative to the trajectory's starting point (0, 0).

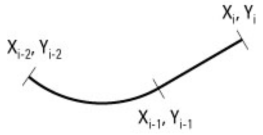


Figure 26: Line element to (X_i, Y_i) position coordinates.

As described before, when adding a new line element, the user must make sure that the discontinuity angle between the new segment and the previous one is not excessive.

8.1.6 Define Arcs

An arc is defined by specifying the radius R and the sweep angle A (Figure 27).

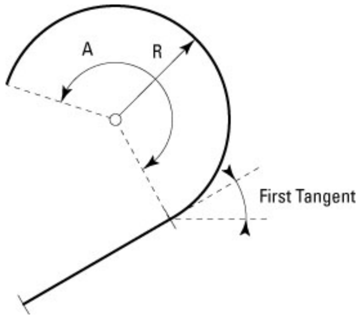


Figure 27: An arc defined with radius and angle.

Both radius and sweep angles are expressed in double precision floating point numbers. The sweep angle can range from 10^{-14} to 1.797 x 10^{308} allowing a definition of arcs from a fraction of a degree to practically an infinite number of overlapping circles.

8.1.7 Trajectory File Description

The Line-arc trajectory is defined in a file that has to be stored in the

..\public\trajectories folder of the XPS controller. This file must have the following structure:

The first line sets the "FirstTangent": Defines the tangent angle for the first point in case of an arc. This parameter has no effect if the first element is a line.

The second line sets the "DiscontinuityAngle": Defines the maximum allowed angle of discontinuity.

The third line must be empty for better readability.

The following lines define the Line-arc trajectory: Each line defines an element of the trajectory.

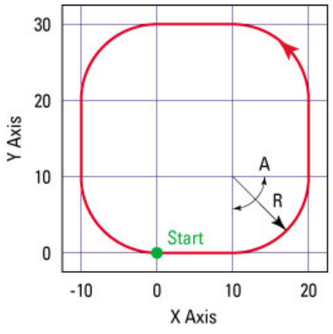
An element can be a "Line" or an "Arc":

Line: Define X and Y positions to build a linear segment Line = X, Y.

Arc: Define radius and sweep angle to build an arc of circle Arc = R, A.

8.1.8 Trajectory File Examples

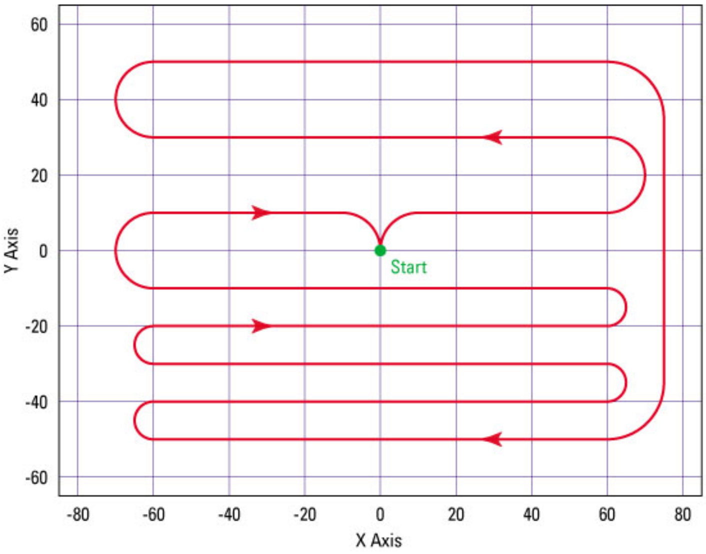
The following is an example of a trajectory file that represents a rectangle with rounded corners and with the end point equal to the starting point:



```
FirstTangent = 0 ; Degrees
DiscontinuityAngle = 0.01 ; Degrees
Line = 10, 0
Arc = 10, 90
Line = 20, 20
Arc = 10, 90
Line = 0, 30
Arc = 10, 90
Line = -10, 10
Arc = 10, 90
```

Figure 28: Graphical display of the first Line-arc trajectory data file example.

The following is an example of a trajectory file that represents a rectangle with rounded corners and with the end point equal to the starting point:



```
FirstTangent = 90 ; Degrees
DiscontinuityAngle = 0.01 ; Degrees
Arc = 10,-90
Line = 60,10
Arc = 10,180
Line = -60,30
Arc = 10,-180
Line = 60,50
Arc = 15,-90
Line = 75,-35
Arc = 15,-90
Line = -60,-50
Arc = 5,-180
Line = 60,-40
Arc = 5,180
Line = -60,-30
Arc = 5,-180
Line = 60,-20
Arc = 5,180
Line = -60,-10
Arc = 10,-180
Line = -10,10
Arc = 10,-90
```

Figure 29: Graphical display of the second Line-arc trajectory data file example.

8.1.9 Trajectory Verification and Execution

There are four functions to verify or execute a Line-arc trajectory:

- **XYLineArcVerification**()**:** Verifies a Line-arc trajectory data file.
- **XYLineArcVerificationResultGet**()**:** Returns the last trajectory verification results, actuator by actuator. This function works only after an XYLineArcVerification().
- **XYLineArcExecution**()**:** Executes a trajectory.
- **XYLineArcParametersGet**()**:** Returns the trajectory's current execution parameters. This function works only while executing the trajectory.

The **function XYLineArcVerification**() can be executed at any time and is independent from trajectory execution. This function performs the following:

- Checks the trajectory file for data and syntax coherence.
- Calculates the trajectory limits, which are: the required travel per positioner, the maximum possible trajectory velocity and the maximum possible trajectory acceleration. This function defines the parameters for trajectory execution.
- If all is OK, it returns an "OK" (0). Otherwise, it returns a corresponding error.

The function **XYLineArcVerificationResultGet**() can be executed only after an XYLineArcVerification() and returns the following:

- Travel requirement in positive and negative direction for each positioner.
- The maximum possible trajectory velocity (speed) that is compatible with all positioner's velocity parameters. It returns a value for the trajectory velocity, that when applied, at least one of the positioners will reach its maximum allowed speed at least once along the trajectory. So the returned value varies between Min $\{V_{max\_actuator}\}$ and $\boldsymbol{volocity} = \sqrt{\sum PositionerMaximumVelocity^2}$. However, this value does not take into account the positioners's acceleration, which can also limit the trajectory velocity. For example, the case of a Line-arc trajectory containing arc segments with a small radius.
- The maximum possible trajectory acceleration that is compatible with all positioners' parameters. This means that one of the positioners will reach its maximum allowed acceleration during the trajectory execution.

The XYLineArcVerificationResultGet() function returns the trajectory execution limits that have previously been calculated by the XYLineArcVerification function. Note about this function's result: Only the returned travel requirements are specific for each positioner. The returned velocity/acceleration values are the same for all positioners, because they represent the trajectory's velocity/acceleration.

To execute a Line-arc trajectory, send the function XYLineArcExecution() with the parameters for the trajectory velocity, and the trajectory acceleration that is used during the start and end of the trajectory. The motion profile for Line-arc trajectories is trapezoidal. The function XYLineArcExecution() does not verify the trajectory coherence or geometric conditions (exceeding any positioners, min. or max. travel, speed or acceleration) before execution, so users must pay attention when executing a trajectory and verify the trajectory relative to the maximum possible values or possible interference. In case of an error during execution, because of bad data or because of a following error (for example if the trajectory acceleration or speed was set too high) the motion group will make an emergency stop and will enter the disabled state. The parameters for trajectory velocity and trajectory acceleration can also be set to zero. In this case the controller uses executable default values which are Min{All $V_{max\_actuator}$} for trajectory velocity and Min{All $A_{max\_actuator}$} for trajectory acceleration.

A trajectory can be executed many times (up to 2^{31} times) by specifying the ExecutionNumber parameter with the XYLineArcExecution function. In this case, the second run of the trajectory is simply appended to the end of the first run, while the end position of the first run is taken as a new start position (referenced to zero) of the second run. The trajectory endpoint does not need to be the same as the start point. The total trajectory is executed without stopping between the different runs.

Finally, the function **XYLineArcParametersGet**() returns the trajectory execution status with trajectory name, trajectory velocity, trajectory acceleration and current executed trajectory element. This function returns an error if the trajectory is not executing.

8.1.10 Examples of the Use of the Functions

XYLineArcVerification (XYGroup, Linearc1.trj)

This function returns a 0 if the trajectory is executable.

XYLineArcVerificationResultGet (XYGroup.XPositioner, *Name, *NegTravel, *PosTravel, *MaxSpeed, *MaxAcceleration)

This function returns the name of the trajectory checked with the last sent function XYLineArcVerification to that motion group (Linearc1.trj), the negative or left travel required for the XYGroup.XPositioner, the positive or right travel required for the XYGroup.XPositioner, the maximum trajectory velocity and the maximum trajectory acceleration.

XYLineArcExecution (XYGroup, Linearc1.trj, 10, 100, 2)

Executes the trajectory Linearc1.trj with a trajectory velocity of 10 units/s and a trajectory acceleration of 100 units/s^2 two (2) times.

XYLineArcParametersGet (XYGroup, *FileName, *TrajectoryVelocity, *TrajectoryAcceleration, *ElementNumber)

Returns the name of the trajectory in execution (Linearc1.trj), the trajectory velocity (10), the trajectory acceleration (100) and the number of the current executed trajectory element.

8.2 Splines

8.2.1 Trajectory Terminology

Trajectory: Continuous multidimensional motion path. Spline trajectories are defined in a three-dimensional XYZ space. They are available with XYZ groups only. The major benefit provided by a spline trajectory is to hit all points (except for the first and the last point that are needed to define the start and the end) and to maintain an almost constant speed (speed being the scalar of the vector velocity) throughout the entire path (except during the acceleration and deceleration periods). Please note that the trajectory speed can vary in some areas depending on the distribution of the reference points. This is related to the spline algorithm used.

Trajectory element (segment): An element of a spline trajectory is defined by a 3rd order polynomial curve joining two consecutive control points.

Trajectory velocity: The tangential linear velocity (speed) along the trajectory during its execution.

Trajectory acceleration: The tangential linear acceleration used to start and end a trajectory. Trajectory acceleration and trajectory deceleration are always equal and by default.

8.2.2 Trajectory Conventions

When defining and executing a spline trajectory, a number of rules must be followed:

- The motion group must be an XYZ group.
- All trajectories must be stored in the controller’s memory under ..\public\trajectories (one file for each trajectory). Once a trajectory is started, it executes in the background allowing other groups or positioners to work independently and simultaneously.
- Each trajectory must have a defined beginning and end. Endless (infinite) trajectories are not allowed. Although, N-times (N defined by user) non-stop execution of a trajectory is allowed. As the trajectory is stored in a file, the trajectory’s maximum size (maximum elements number) is unlimited for practical purposes.
- Spline trajectory elements (segments) are 3rd order polynomial curve segments $S_i(u)$, joining the positions P_{i-1} (X_{i-1}, Y_{i-1}, Z_{i-1}) and P_i (X_i, Y_i, Z_i). Here “u” is the normalized time parameter that varies from 0 (corresponding to P_{i-1}) to 1 (corresponding to P_i).
- Spline trajectories form a continuous path (each segment’s output position is equal to the next segment’s input position), and the segment tangential angles at the connection point of any two consecutive segments are continuous, including its derivative. For reference, this discontinuity is categorized as R^1, wherein position and velocity are continuous, but not acceleration.

8.2.3 Geometric Conventions

The Spline trajectory's coordinate system is an XYZ orthogonal system.

The X-axis of this system correlates to the XPositioner, the Y-axis to the YPositioner, and the Z-axis to the ZPositioner of the XYZ group as defined in the stages.ini.

The origin of the XYZ coordinate system is in the lower left corner, with positive values up (Z), to the right (X) and forward (Y).

All angles are measured in degrees, presented as floating point numbers. Angle origin and sign follow the trigonometric convention: positive angles are measured counter-clockwise.

8.2.4 Catmull-Rom Interpolating Splines

To trace a smooth curve that links different predefined trajectory points, the intermediate points must be calculated following a mathematical model. For the sake of simplicity, in most cases this is done by a polynomial curve (polynomial interpolation). For motion systems, the resulting curve should hit all predefined points. This is called precise interpolation in contrast to approximate interpolation (like Bezier splines), where the predefined points act only as control points. Within this class of precise interpolation are:

- Global polynomial interpolation: One polynomial represents the whole trajectory. Examples are Lagrange polynomials or Newton polynomials.
- Local polynomial interpolation: Each segment that links two consecutive trajectory points has its own polynomial. The resulting curve is obtained by segment polynomial concatenation. To limit oscillations inside segments, the polynomial order is generally limited to 3 or less. This is called spline interpolation. If the polynomial order is equal to 3, it is called cubic spline interpolation.

The interpolation methods are also classified by the continuity criterion C^k. An interpolating curve has the continuity C^k if it and its derivatives up to k-degrees are continuous in all its points. The interpolating spline curves generally have C^1 or C^2 continuity.

Catmull-Rom splines are a family of **local cubic interpolating splines** where the tangent at each point p_i is calculated based on the previous p_{i-1} and the next point p_{i+1} on the spline. In case of the spline curve tension $\tau = 1/2$ (normal case), the **Catmull-Rom** spline is described by the following equation:

$$S(u) = \left(u^3\ u^2\ u\ ^1\right) \cdot \frac{1}{2} \cdot \left(\left(\begin{matrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{matrix}\right)\right) \cdot \left(\begin{matrix} p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \end{matrix}\right)$$

Here, p_i are the coordinates of the predefined trajectory point in x, y and z (p_{xi}, p_{yi}, p_{zi}). “u” is the normalized interpolating parameter, varying from 0 (starting at p_i) to 1 (ending at p_{i+1}).

Catmull-Rom splines have a C^1 continuity (continuity up to the first derivative), local control and interpolation. **Catmull-Rom** splines have the advantage of simple calculation without matrix inversion for on-line calculations, which is a great advantage for splines with a large number of trajectory points. For this reason, the XPS controller uses the **Catmull-Rom** spline interpolation.

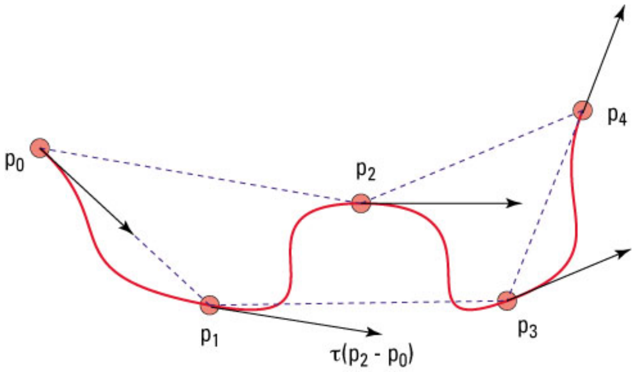


Figure 30: A Catmull-Rom spline.

8.2.5 Trajectory Elements Arc Length Calculation

Spline contouring at constant speed requires an accurate calculation of the segment’s arc length. The segment’s arc length can be expressed as follows:

$$L(u_0,u_1) = \int_{u_0}^{u_1} \sqrt{\left(\frac{d}{du}Sx(u)\right)^2 + \left(\frac{d}{du}Sy(u)\right)^2 + \left(\frac{d}{du}Sz(u)\right)^2}\, du$$

Here, $u_0 = 0$ is the segment starting point and $u_1 = 1$ is the segment ending point. Sx, Sy, Sz are x-, y-, and z-components of the segment function.

This integral can only be numerically calculated, which is done by the XPS controller using the Romberg numerical integration algorithm. This guarantees that the arc length is calculated with an error less than 10^{-7} units.

8.2.6 Trajectory File Description

The spline trajectory is described in a file in the \Admin\Public\Trajectories folder of the XPS controller. Each line of this file represents one point of the spline trajectory except for the first and the last lines that are needed only to define the start and the end of the trajectory. Two consecutive points form a trajectory segment.

The format of a line in a file is:

X-POSITION, Y-POSITION, Z-POSITION

The separator between the X-, Y-, and Z-Position is a comma.

As mentioned before, the first and last lines of the file are needed only for the interpolation of the first and the last spline segments. These define the angle the trajectory starts and ends, but the motion system will not hit these points. So the trajectory's first "real" point (starting point) is the one defined by the second line and the trajectory's real "last" point (end point) is the one defined by the second to the last line.

The position values in the data file are relative to the physical position of the motion group at the start of the trajectory. If the position in the second line of the file (starting point) is not equal to zero (0, 0, 0), the real trajectory positions (those that the motion group will hit) are shifted further by this value.

Example

The spline trajectory file has the following format:

$$\begin{array}{ccc}
x_0 & y_0 & z_0 \\
x_1 & y_1 & z_1 \\
x_2 & y_2 & z_2 \\
x_3 & y_3 & z_3 \\
x_4 & y_4 & z_4 \\
\cdots & \cdots & \cdots
\end{array}$$

At the moment the trajectory is executed, the motion group is at the position X_C, Y_C, Z_C. So the real matrix in absolute coordinates of the motion group is:

$$\begin{array}{ccc}
x_{c+x_0-x_1} & y_{c+y_0-y_1} & z_{c+z_0-z_1} \\
x_c & y_c & z_c \\
x_{c+x_2-x_1} & y_{c+y_2-y_1} & z_{c+z_2-z_1} \\
x_{c+x_3-x_1} & y_{c+y_3-y_1} & z_{c+z_3-z_1} \\
x_{c+x_4-x_1} & y_{c+y_4-y_1} & z_{c+z_4-z_1} \\
\cdots & \cdots & \cdots
\end{array}$$

8.2.7 Trajectory File Example

This trajectory example represents a spiral starting from (0, 20, 0) and ending at (0, -20, 24). As described before, the trajectory's first (-5, 19.365, -1) and last (5, -19.365, 25) points are only needed to define the start and end conditions of the trajectory. Because the second line (0, 20, 0) is not equal to zero (0, 0, 0), all points that the motion group will hit during the execution of the trajectory are reduced by this value from the physical starting position of the motion group.

The original data file is (except for the tabs that are only added for better readability):

-5,	19.365,	-1	-15,	13.229,	13
0,	20,	0	-10,	17.321,	14
5,	19.365,	1	-5,	19.365,	15
10,	17.321,	2	0,	20,	16
15,	13.229,	3	5,	19.365,	17
20,	0,	4	10,	17.321,	18
15,	-13.229,	5	15,	13.229,	19
10,	-17.321,	6	20,	0,	20
5,	-19.365,	7	15,	-13.229,	21
0,	-20,	8	10,	-17.321,	22
-5,	-19.365,	9	5,	-19.365,	23
-10,	-17.321,	10	0,	-20,	24
-15,	-13.229,	11	5,	-19.365,	25
-20,	0,	12			

With this data file, the real trajectory points relative to the physical start position of the motion group are (first and last lines are eliminated because the motion group will not hit these points and the values from the second column are reduced by 20 as the first line was (0, 20, 0)):

0,	0,	0	-15,	-6.771,	13
5,	-0.635,	1	-10,	-2.679,	14
10,	-2.679,	2	-5,	-0.635,	15
15,	-6.771,	3	0,	0,	16
20,	-20,	4	5,	-0.635,	17
15,	-33.229,	5	10,	-2.679,	18
10,	-37.321,	6	15,	-6.771,	19
5,	-39.365,	7	20,	-20,	20
0,	-40,	8	15,	-33.229,	21
-5,	-39.365,	9	10,	-37.321,	22
-10,	-37.321,	10	5,	-39.365,	23
-15,	-33.229,	11	0,	-40,	24
-20,	-20,	12			

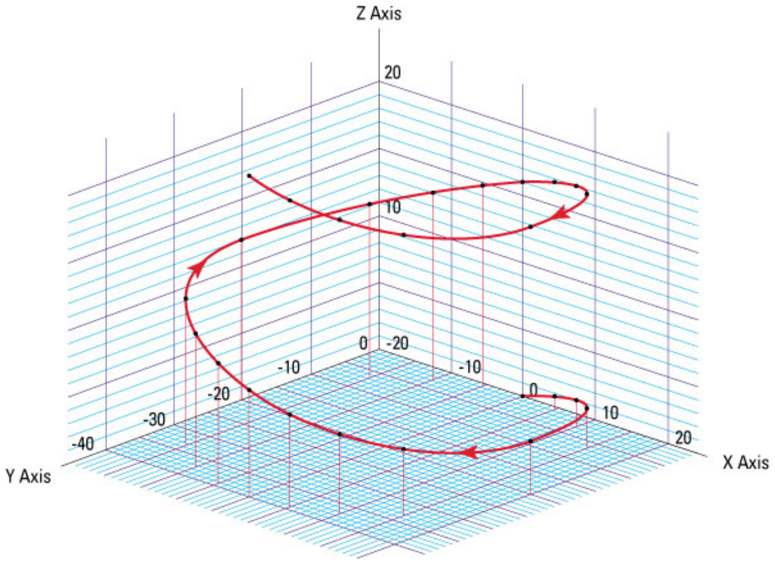


Figure 31: Executing the above normalized trajectory data file with the Catmull-Rom spline algorithm.

8.2.8 Spline Trajectory Verification and Execution

Here are four functions to verify or execute a spline trajectory:

- **XYZSplineVerification():** Verifies a spline trajectory data file.
- **XYZSplineVerificationResultGet():** Returns the last trajectory verification results, actuator by actuator. This function works only after an XYZSplineVerification().
- **XYZSplineExecution():** Executes a trajectory.
- **XYZSplineParametersGet():** Returns the trajectory current execution parameters. This function works only while executing of the trajectory.

The function **XYZSplineVerification**() can be executed at any moment and is independent from the trajectory execution. This function performs the following:

- Checks the trajectory file for data and syntax coherence.
- Calculates the trajectory limits, which are the required travel per positioner, the maximum possible trajectory velocity and the maximum possible trajectory acceleration. This function defines the parameters for trajectory execution.
- If all is OK, it returns an “OK” (0). Otherwise, it returns a corresponding error.

The function **XYZSplineVerificationResultGet**() can be executed only after an XYZSplineVerification() and returns the following:

- Travel requirement in the positive and negative directions for each positioner.
- The maximum possible trajectory velocity (speed) that is compatible with all positioners’ velocity parameters. It returns a value for the trajectory velocity, that when applied, at least one of the positioners will reach its maximum allowed speed at least once along the trajectory. So the returned value varies between Min{Vmax_actuator} and $\boldsymbol{velocity} = \sqrt{\sum \mathit{PositionerMaximumVelocity}^2}$. However, this value does not take into account that the positioners’ acceleration can limit the trajectory velocity. This is the case with splines that contain sharp curved segments.
- The maximum trajectory acceleration that is compatible with all positioner parameters. At this trajectory acceleration, one of the positioners will reach its maximum allowed acceleration during trajectory execution.

The function **XYZSplineVerificationResultGet**() returns the trajectory execution limits that have previously been calculated by the XYZSplineVerification function. Note on this function’s response: Only the returned travel requirements are specific for each positioner, the returned velocity/acceleration values are the same for all positioners, because they represent the trajectory’s velocity/acceleration.

To execute a spline trajectory, send the function **XYZSplineExecution**() with the parameters for the trajectory velocity and the trajectory acceleration (the trajectory acceleration that is used during the start and the end of the trajectory). The motion profile for spline trajectories is trapezoidal. The function XYZSplineExecution() does not verify the trajectory’s coherence or geometric conditions (exceeding any positioner’s min. or max. travel, speed or acceleration) before execution, so users must pay attention when executing a trajectory without verifying the trajectory the maximum possible values. In case of an error during execution, because of bad data or because of a following error (for example the trajectory acceleration or speed was set too high) the motion group will make an emergency stop and will go to the disabled state. The parameters for trajectory velocity and trajectory acceleration can also be set to zero. In this case the controller uses executable default values which are the Min{All $V_{max\_actuator}$} for trajectory velocity and Min{All $A_{max\_actuator}$} for trajectory acceleration.

Finally, the function **XYZSplineParametersGet**() returns the trajectory execution status with trajectory name, trajectory velocity, trajectory acceleration and current executed trajectory element. This function returns an error if the trajectory is not executing.

8.2.9 Examples

XYZSplineVerification (XYZGroup, Spline1.trj)

This function returns a 0 if the trajectory is executable.

XYZSplineVerificationResultGet (XYZGroup.XPositioner, *Name, *NegTravel, *PosTravel, *MaxSpeed, *MaxAcceleration)

This function returns the name of the trajectory checked with the last sent function XYZSplineVerification to that motion group (Spline1.trj), the negative travel required for the XYZGroup.XPositioner, the positive travel required for the XYZGroup.XPositioner, the maximum trajectory velocity and the maximum trajectory acceleration.

XYZSplineExecution (XYZGroup, Spline1.trj, 10, 100)

Executes the trajectory Spline1.trj with a trajectory velocity of 10 units/s and a trajectory acceleration of 100 units/s^2.

XYZSplineParametersGet (XYZGroup, *FileName, *TrajectoryVelocity, *TrajectoryAcceleration, *ElementNumber)

Returns the name of the trajectory being executed (Spline1.trj), the trajectory velocity (10), the trajectory acceleration (100) and the number of the currently executed trajectory element.

8.3 PVT Trajectories

8.3.1 Trajectory Terminology

Trajectory: continuous, multidimensional motion path. PVT stands for Position, Velocity, and Time. PVT trajectories are defined in an n-dimensional space (n = 1 to 4) and are only available with MultipleAxes groups. A PVT trajectory is generated with continuous movements of the group’s positioners over several time periods. For each period, each positioner must complete a defined displacement from its current position and a defined output velocity at the end of the period. By definition, there is no constant vector velocity and no definition for a vector acceleration in contrast to Line-arc trajectories or splines.

Trajectory element (segment): An element of a PVT trajectory is defined by a set of all positioner displacements and output velocities and the duration for the segment. In the PVT data file, each element is represented by a line of values:

DT, DP1, VO1, DP2, VO2, … DPn, VOn

DT:	The segment duration in seconds.
DP1, DP2,…, DPn:	Positioners’ (#1, #2,…, #n) displacements during DT.
VO1, VO2,…, VOn:	Positioners’ output velocities at the end of DT.

8.3.2 Trajectory Conventions

When defining or executing a PVT trajectory, a number of rules must be followed:

- The motion group must be a MultipleAxes group.
- All trajectories must be stored in the controller’s memory. Use the XPS webpage **Files → Trajectory files** to edit, upload or download a PVT trajectory file. Once a trajectory is started, it executes in the background allowing other groups to work independently and simultaneously.
- Each trajectory must have a beginning and an end. Endless (infinite) trajectories are not allowed. Although, execution of a trajectory file N-times (N defined by user) is allowed. Since the trajectory is stored in a file, the trajectory’s maximum size (maximum elements number) is practically not limited.

- PVT trajectory elements (segments) are 3rd order polynomial pieces for each positioner that hit the positions P_{i-1} (at time t_{i-1} with a velocity v_{i-1}) and positions P_i (at time t_i with a velocity v_i). There is no direct link between the trajectories of the different positioners in the group.
- PVT trajectories form a continuous path (each segment output position is equal to the next segment input position), and the segment tangential angles at the connection point of any two consecutive segments are continuous including its derivative. It means that the PVT trajectory continuity property is R^1.
- The input velocity of any element is equal to the output velocity of the previous element. The input velocity for the first element is always zero. The output velocity of the last element must be zero as well.

8.3.3 Geometric Conventions

- The coordinate system can be any convention, it does not need to be an orthogonal system.
- A PVT trajectory can be defined for a MultipleAxes group. The number of positioners in the PVT trajectory must match that of the linked with MultipleAxes group.

8.3.4 PVT Interpolation

For each positioner belonging to the MultipleAxes group, the PVT trajectory calculates a 3rd order polynomial curve P(u) that can be presented by the following equations:

Profile coefficient

- Acceleration jerk:

$$\mathrm{Jerk} = \frac{6 \cdot \left[\mathrm{DT} \cdot \left(\mathrm{V_{in}} + \mathrm{V_{out}}\right) - 2 \cdot \mathrm{DX}\right]}{\mathrm{DT}^3}$$

- Initial acceleration:

$$\mathrm{G_{in}} = \frac{2 \cdot \left[3 \cdot \mathrm{DX} - \mathrm{DT} \cdot \left(2 \cdot \mathrm{V_{in}} + \mathrm{V_{out}}\right)\right]}{\mathrm{DT}^2}$$

- Final acceleration:

$$\mathrm{G_{out}} = \frac{2 \cdot \left[\mathrm{DT} \cdot \left(\mathrm{V_{in}} + 2 \cdot \mathrm{V_{out}}\right) - 3 \cdot \mathrm{DX}\right]}{\mathrm{DT}^2}$$

Profile equation

- Acceleration:

$$\mathrm{Acc(t)} = \mathrm{G_{in}} + \mathrm{Jerk} \cdot \mathrm{t}$$

- Velocity:

$$\mathrm{Vel(t)} = \mathrm{V_{in}} + \mathrm{G_{in}} \cdot \mathrm{t} + \frac{\mathrm{Jerk} \cdot \mathrm{t}^2}{2}$$

- Position:

$$\mathrm{Pos(t)} = \mathrm{V_{in}} \cdot \mathrm{t} + \frac{\mathrm{G_{in}} \cdot \mathrm{t}^2}{2} + \frac{\mathrm{Jerk} \cdot \mathrm{t}^3}{6}$$

Here:

DT is the segment duration in seconds

DX is the displacement during DT

V_{in} is the output velocity of the previous segment (which is equal to the input velocity of the current segment)

V_{out} is the output velocity of the current segment.

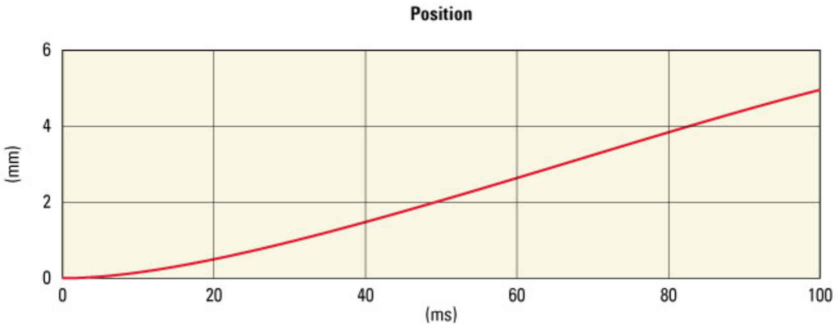
t is the time in seconds starting at 0 (entry of the current element) and ending at DT (end of the segment)

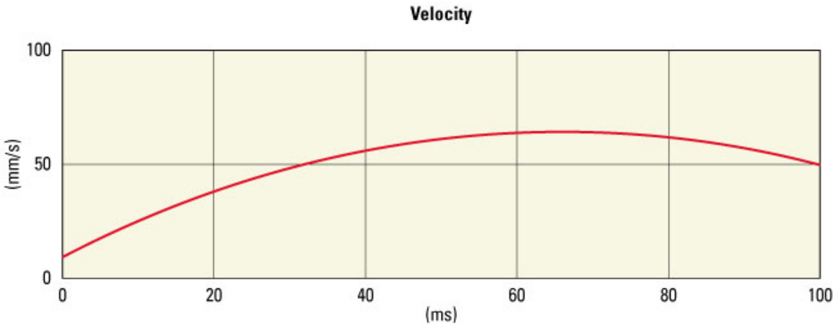
8.3.5 Influence of the Element Output Velocity to the Trajectory

The contour of each PVT trajectory element is influenced not only by the displacement, but also by the input and output velocities. As the user decides on these velocities, attention must be placed on these values to get the desired results.

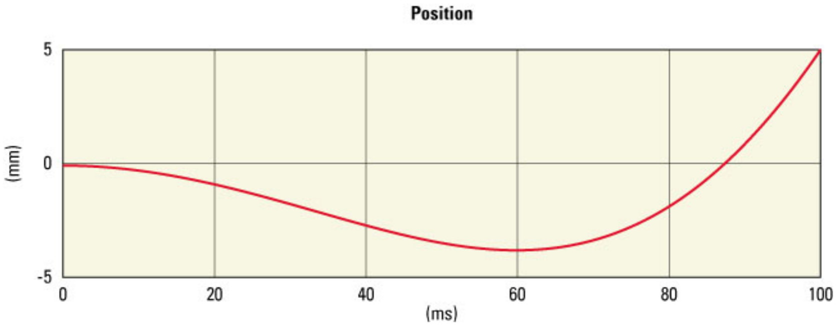
The effect of the velocity is illustrated in the following example which shows the position and velocity profiles for one segment of a PVT trajectory that has a displacement of 5 mm, a duration of 100 ms, an input velocity of 10 mm/s and an output velocity of either 50 mm/s or 500 mm/s:

- If the output velocity is equal to 50 mm/s.





- If the output velocity is equal to 500 mm/s.



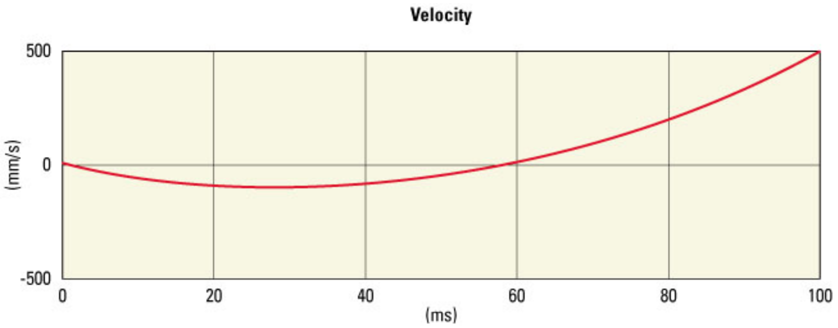


Figure 32: PVT trajectory element in execution: the comparison.

A PVT trajectory must have three parameters: position, velocity and time. With a given target displacement, output velocity and time duration, the PVT trajectory calculates intermediate positions and velocities as a function of time.

With an output velocity of 50 mm/s, the positioner has "enough" time to achieve the displacement within the assigned time (100 ms) in the forward direction. The velocity increases at the beginning and then slows down towards the end. The position always increases up to the target position (5 mm).

On the other hand, when the output velocity is set to 500 mm/s, the positioner does not have enough time to achieve the displacement and speed output required in the forward direction. So the positioner will first reverse the direction of motion to be able to approach the end position with a speed of 500 mm/s.

8.3.6 Trajectory File Description

The PVT trajectory described must be stored in the controller's memory. Use the XPS webpage **Files → Trajectory files** to edit, upload or download a PVT trajectory file. Each line of this file represents one element of the trajectory.

A line contains several data separated by a comma. The number of data in each line depends on the number of positioners in the group. The first data in each line is the duration of the element. The following data is grouped in pairs of two representing the displacement and the output velocity for each positioner of the group. Comment lines are possible, they must be preceded with a semi-colon (";") character.

So the line format is as follows:

Data #1:	Element duration (seconds).
Data #2:	1st positioner's displacement (units).
Data #3:	1st positioner's output velocity (units/s).
Data #4:	2nd positioner's displacement (units).
Data #5:	2nd positioner's output velocity (units/s).

(And so on…)

NOTE

The first positioner is always the first defined in the system.ini of the MultipleAxes group (see PositionerInUse), the second positioner is always defined as second, and so on…

Option to add GPIO outputs

To add either analog or digital outputs during the trajectory the first line of the file should specify the selected outputs and each element should define the GPIO output per plug with the following data:

- Digital Outputs

  For digital outputs, two values must be specified per element and per selected GPIO plug to define the state of the output at the end of the element, namely Mask and DigitalOutputValue.

- Analog Outputs

  Analog outputs are handled like position to output a controlled voltage profile. For analog outputs, two values per element and per selected output plug must be specified, the voltage variation (DU) during the element and the rate of change (VU) at the end of the element.

Example format:

; Comment: A MultipleAxes group with two positioners, one digital output and two analog outputs.

GPIO1.DO, GPIO2.DAC1, GPIO2.DAC2

DT1, DX1, DX2, Mask1, Value1, DU1, VU1, DU2, VU2

DT2, DX1, DX2, Mask1, Value1, DU1, VU1, DU2, VU2

DT3, DX1, DX2, Mask1, Value1, DU1, VU1, DU2, VU2

…

DTm, DX1, DX2, Mask1, Value1, DU1, VU1, DU2, VU2

8.3.7 Trajectory File Example

Following is an example of a PVT trajectory defined in a MultipleAxes group that contains two positioners. The tabs are added for better readability and are ignored in a line:

```
1.0,    0.4167,     1.25,    0,          0
1.0,    2.9167,     5,       0,          0
1.0,    7.0833,     8.75,    0,          0
1.0,    9.5833,    10,       0,          0
1.0,   10,         10,       0.4167,     1.25
1.0,   10,         10,       2.9167,     5
1.0,   10,         10,       7.0833,     8.75
1.0,   10,         10,       9.5833,    10
1.0,    9.5833,     8.75,   10,         10
1.0,    7.0833,     5,      10,         10
1.0,    2.91667,    1.25,   10,         10
1.0,    0.41667,    0,      10,         10
1.0,    0,          0,       9.5833,     8.75
1.0,    0,          0,       7.0833,     5
1.0,    0,          0,       2.91667,    1.25
1.0,    0,          0,       0.41667,    0
```

This file represents the following data:

Time Period (s)	Axis #1 Displacement	Axis #1 Velocity Out	Axis #2 Displacement	Axis #2 Velocity Out
1.0	0.4167	1.25	0	0
1.0	2.9167	5.0	0	0
1.0	7.0833	8.75	0	0
1.0	9.5833	10	0	0
1.0	10	10	0.4167	1.25
1.0	10	10	2.9167	5
1.0	10	10	7.0833	8.75
1.0	10	10	9.5833	10
1.0	9.5833	8.75	10	10
1.0	7.0833	5	10	10
1.0	2.9167	1.25	10	10
1.0	0.4167	0	10	10
1.0	0	0	9.5833	8.75
1.0	0	0	7.0833	5
1.0	0	0	2.9167	1.25
1.0	0	0	0.4167	0

Table 1: The trajectory data file.

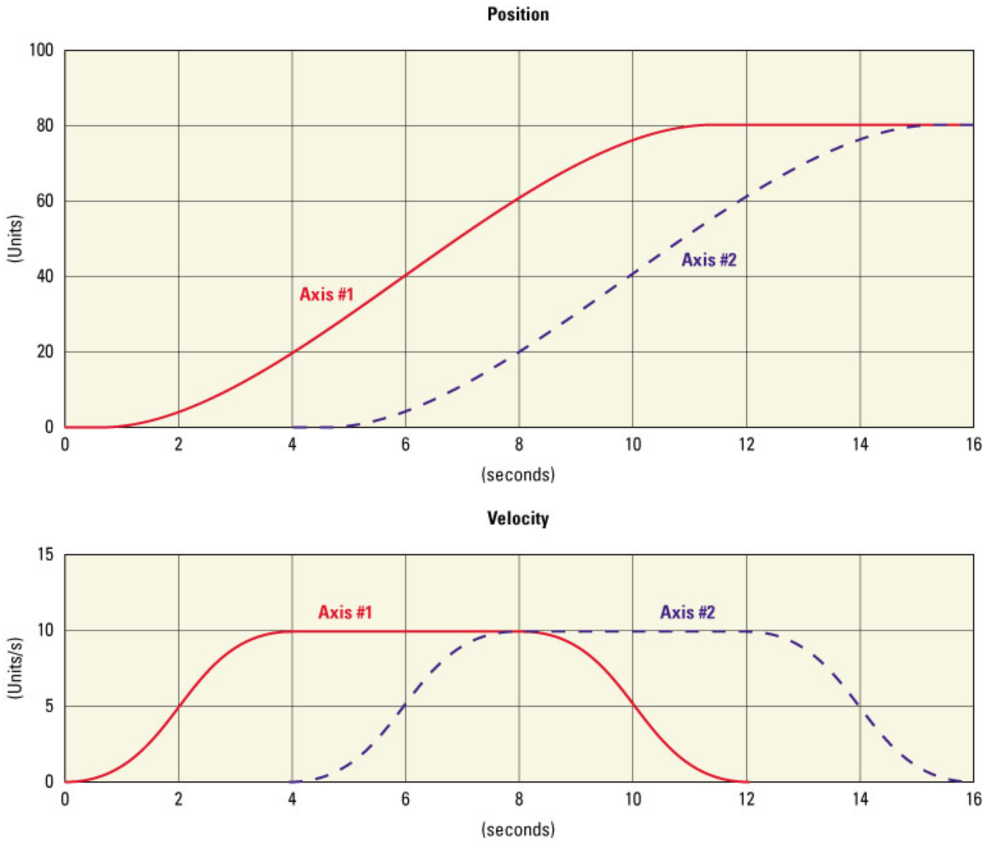


Figure 33: Executing the trajectory data file with the PVT algorithm.

8.3.8 PVT Trajectory Verification and Execution

Here are four functions to verify or execute a PVT trajectory :

- **MultipleAxesPVTVerification():** Verifies a PVT trajectory data file.
- **MultipleAxesPVTVerificationResultGet():** Returns the results of the last trajectory verification call, actuator by actuator. This function works only after a MultipleAxesPVTVerification().
- **MultipleAxesPVTExecution():** Executes a PVT trajectory.
- **MultipleAxesPVTParametersGet():** Returns the trajectory's current execution parameters. This function works only while executing a trajectory.

The function **MultipleAxesPVTVerification**() can be executed at any moment and is independent of the trajectory execution. This function does the following:

- Checks the trajectory file for data and syntax coherence.
- Simulates the trajectory to determine the positioner's travel requirements in negative and positive directions and the maximum allowed speed and acceleration for each positioner. This function determines whether the trajectory is executable.
- If all is OK, it returns an "OK" (value 0). Otherwise it returns a corresponding error. An error for instance is reported if one of the positioner's speed or acceleration reached during the trajectory exceeds the maximum allowed speed or acceleration.

The function **MultipleAxesPVTVerificationResultGet**() can be executed only after a MultipleAxesPVTVerification(). It returns the trajectory limits for each positioner, which are the travel requirements in positive and negative directions, the achieved maximum speed and acceleration.

To execute a PVT trajectory, send the function **MultipleAxesPVTExecution**() while specifying the file name and the number of cycles. This function does not verify the trajectory's coherence or geometric conditions (exceeding any positioner's min. or max. travel, speed or acceleration) before execution, so users must be careful when executing a trajectory without verifying the trajectory first. In case of an error during execution, because of bad data or because of a following error, the motion group will make an emergency stop and will go to the disabled state.

Finally, the function MultipleAxesPVTParametersGet() returns the trajectory name and the number of the trajectory element that is currently being executed. This function returns an error if the trajectory is not executing.

8.3.9 Example with a MultpleAxes Group

MultipleAxesPVTVerification (MyGroup, PVT1.trj)

This function returns a 0 if the trajectory is executable.

MultipleAxesPVTVerificationResultGet (MyGroup.MyPositioner1, *Name, *NegTravel, *PosTravel, *MaxSpeed, *MaxAcceleration)

This function returns the name of the trajectory verified with the last functions call of MultipleAxesPVTVerification to the motion group MyGroup(PVT1.trj) and the trajectory limits for the positioner MyGroup.MyPositioner1. These trajectory limits are: the negative or left travel requirement, the positive or right travel requirement, the achieved maximum speed and acceleration. Make sure that these trajectory limits (required negative and positive travel, speed and acceleration) are within the soft limits of the stages defined in the stages.ini file (section Travel: MinimumTargetPosition, MaximumTargetPosition and section Profiler: MaximumVelocity, MaximumAcceleration).

MultipleAxesPVTExecution (MyGroup, PVT1.trj, 5)

Executes the trajectory PVT1.trj five (5) times.

MultipleAxesPVTParametersGet (MyGroup, *FileName, *ElementNumber)

Returns the currently executed trajectory file name (PVT1.trj) and the number of the currently executed trajectory element.

8.4 PT Trajectrories

8.4.1 Trajectory Terminology

Trajectory: continuous, multidimensional motion path. PT stands for Position and Time. PT trajectories are defined in an n-dimensional space (n = 1 to 4) and are available only with MultipleAxes groups. A PT trajectory is generated with continuous movements of the group's positioners over several time periods. For each period, each positioner must complete a defined displacement from its current position. By definition, there is no constant vector velocity and no definition for vector acceleration compared to Line-arc trajectories or splines.

Trajectory element (segment): An element of a PT trajectory is defined by a set of all positioner displacements and the duration for the segment. In the PT data file, each element is represented by a line of values:

DT, DP1, DP2, … DPn

DT: The segment duration in seconds.

DP1, DP2,…, DPn: Positioners' (#1, #2,…, #n) displacements during DT.

8.4.2 Trajectory Conventions

When defining or executing a PT trajectory, a number of rules must be followed:

- The motion group must be a MultipleAxes group.
- All trajectories must be stored in the controller's memory. Use the XPS webpage **Files → Trajectory files** to edit, upload or download a PT trajectory file. Once a trajectory is started, it executes in the background allowing other groups to work independently and simultaneously.
- Each trajectory must have a beginning and an end. Endless (infinite) trajectories are not allowed. Although, execution of a trajectory file N-times (N defined by user) is allowed. Since the trajectory is stored in a file, the trajectory's maximum size (maximum elements number) is practically not limited.
- PT trajectory elements (segments) are 3rd order polynomial pieces for each positioner that hit the positions P_{i-1}, at time t_{i-1}, and positions P_i, at time t_i. There is no direct link between the trajectories of the different positioners in the group.
- PT trajectories form a continuous path: each segment output position is equal to the next segment input position and the input velocity of any element is equal to the

output velocity of the previous element. Hence the segment tangential angles at the connection point of any two consecutive segments are continuous including its derivative. It means that the PT trajectory continuity property is R^1.

- The input velocity for the first element is always zero. The output velocity of the last element must be zero as well.

8.4.3 Geometric Conventions

- The coordinate system can be any convention; it does not need to be an orthogonal system.
- A PT trajectory can be defined for a MultipleAxes group. The number of positioners in the PT trajectory must match that of the linked with MultipleAxes group.

8.4.4 PT Interpolation

For each positioner belonging to the MultipleAxes group, the PT trajectory calculates a 3rd order polynomial curve X(t) that can be represented by the following equations:

$$Current\ element = (DT1, DX1)$$

$$Next\ element = (DT2, DX2)$$

$$X_{in} = X_{out}\ (previous)$$

$$V_{in} = V_{out}\ (previous)$$

$$V_{out} = \frac{DX2 \cdot DT1^2 + DX1 \cdot DT2^2}{DT1 \cdot DT2 \cdot (DT1 + DT2)}$$

$$G_{in} = \frac{2 \cdot [3 \cdot DX1 - DT1 \cdot (2 \cdot V_{in} + V_{out})]}{DT1^2}$$

$$Jerk = \frac{6 \cdot (DT1 \cdot V_{in} - 2 \cdot DX1 + DT1 \cdot V_{out})}{DT1^3}$$

$$X(t) = X_{in} + V_{in} \cdot t + \frac{1}{2} \cdot G_{in} \cdot t^2 + \frac{1}{6} \cdot Jerk \cdot t^3$$

The algorithm is the same as for PVT mode except that the output velocity is not set in the trajectory file, but calculated by the firmware using the following rule:

The crossing velocity of a point is defined by taking into account the previous and the following point, as if the three points where to be crossed with a constant acceleration. This crossing velocity becomes the set output velocity of the element defined by the first two points. The result is a lower speed ripple than a path at constant acceleration.

8.4.5 Trajectory File Description

The PT trajectory described must be stored in the controller's memory. Use the XPS webpage **Files → Trajectory files** to edit, upload or download a PT trajectory file. Each line of this file represents one element of the trajectory.

A line contains several data entries separated by a comma. Comment lines are possible, and must be preceded with a semi-colon (";") character. The number of data entries in each line depends on the number of positioners in the MultipleAxes group. The first data entry in each line is the duration of the element. The following data entries represent the displacement for each positioner of the group.

So a generic format is as follows:

```
DT1,     DX1,  DX2,  …,    DXn
DT2,     DX1,  DX2,  …,    DXn
…
DTm-2,   DX1,  DX2,  …,    DXn
DTm-1,   0,    0,    …,    0
DTm,     0,    0,    …,    0
```

NOTES

- **To guarantee that the output velocity is null at the end of PT trajectory execution, two data lines with zero displacements must be present at the end of the PT trajectory.**
- **The first positioner is always the first defined in the system.ini of the MultipleAxes group (see PositionerInUse), the second positioner is always defined as second, and so on…**

Option to add GPIO outputs

To add either analog or digital outputs during the trajectory the first line of the file should specify the selected outputs and each element should define the GPIO output per plug with the following data:

- Digital Outputs

  For digital outputs, two values must be specified per element and per selected GPIO plug to define the state of the output at the end of the element, namely Mask and DigitalOutputValue.

- Analog Outputs

  Analog outputs are handled like position to output a controlled voltage profile. For analog outputs, one value per element and per output plug must be specified, the voltage variation (DU) during the element.

Example format:

; Comment: A MultipleAxes group with two positioners, one digital output and two analog outputs

```
GPIO1.DO, GPIO2.DAC1, GPIO2.DAC2
DT1, DX1, DX2, Mask1, Value1, DU1, DU2
DT2, DX1, DX2, Mask1, Value1, DU1, DU2
DT3, DX1, DX2, Mask1, Value1, DU1, DU2
…
DTM, DX1, DX2, Mask1, Value1, DU1, DU2
```

8.4.6 Trajectory File Example

Following is an example of a PT trajectory defined in a MultipleAxes group that contains two positioners and configured to output one digital and one analog output. The data entries in a line are separated by a comma (","). Comment lines are possible and must be preceded with a semi-colon (";") character. Because of the PT trajectory internal calculation of elements end velocity, two lines with zero displacements must be present at the end of the PT trajectory file to guarantee that the elements end velocity be zero at the end of trajectory execution. In the example below tabs were added for better readability:

; PT trajectory data					
GPIO1.DO,	GPIO2.DAC1,				
1.0,	0.4167,	0,	65535,	1,	0.04167
1.0,	2.9167,	0,	65535,	2,	0.29167
1.0,	7.0833,	0,	65535,	4,	0.70833
1.0,	9.5833,	0,	65535,	8,	0.95833
1.0,	10,	0.4167,	65535,	16,	1.0
1.0,	10,	2.9167,	65535,	32,	1.0
1.0,	10,	7.0833,	65535,	64,	1.0
1.0,	10,	9.5833,	65535,	128,	1.0
1.0,	9.5833,	10,	65535,	64,	0.95833
1.0,	7.0833,	10,	65535,	32,	0.70833
1.0,	2.9167,	10,	65535,	16,	0.29167
1.0,	0.4167,	10,	65535,	8,	0.04167
1.0,	0,	9.5833,	65535,	4,	0
1.0,	0,	7.0833,	65535,	2,	0
1.0,	0,	2.9167,	65535,	1,	0
1.0,	0,	0.4167,	65535,	0,	0
1.0,	-0.4167,	0,	65535,	1,	-0.04167
1.0,	-2.9167,	0,	65535,	2,	-0.29167
1.0,	-7.0833,	0,	65535,	4,	-0.70833
1.0,	-9.5833,	0,	65535,	8,	-0.95833
1.0,	-10,	-0.4167,	65535,	16,	-1.0
1.0,	-10,	-2.9167,	65535,	32,	-1.0
1.0,	-10,	-7.0833,	65535,	64,	-1.0
1.0,	-10,	-9.5833,	65535,	128,	-1.0
1.0,	-9.5833,	-10,	65535,	64,	-0.95833
1.0,	-7.0833,	-10,	65535,	32,	-0.70833
1.0,	-2.9167,	-10,	65535,	16,	-0.29167
1.0,	-0.4167,	-10,	65535,	8,	-0.04167
1.0,	0,	-9.5833,	65535,	4,	0
1.0,	0,	-7.0833,	65535,	2,	0
1.0,	0,	-2.9167,	65535,	1,	0
1.0,	0,	-0.4167,	65535,	0,	0

This file represents the following data:

GPIO #1 Digital Output	GPIO #2 Analog Output
GPIO1.DO	GPIO2.DAC1

Time Period (S)	Axis #1 Displacement	Axis #2 Displacement	GPIO #1 Mask1	GPIO #1 Value 1	GPIO#2 Analog voltage variation DU2
1.0	0.4167	0	65535	1	0.04167
1.0	2.9167	0	65535	2	0.29167
1.0	7.0833	0	65535	4	0.70833
1.0	9.5833	0	65535	8	0.95833
1.0	10	0.4167	65535	16	1.0
1.0	10	2.9167	65535	32	1.0
1.0	10	7.0833	65535	64	1.0
1.0	10	9.5833	65535	128	1.0
1.0	9.5833	10	65535	64	0.95833
1.0	7.0833	10	65535	32	0.70833
1.0	2.9167	10	65535	16	0.29167
1.0	0.4167	10	65535	8	0.04167
1.0	0	9.5833	65535	4	0
1.0	0	7.0833	65535	2	0
1.0	0	2.9167	65535	1	0
1.0	0	0.4167	65535	0	0
1.0	-0.4167	0	65535	1	-0.04167
1.0	-2.9167	0	65535	2	-0.29167
1.0	-7.0833	0	65535	4	-0.70833
1.0	-9.5833	0	65535	8	-0.95833
1.0	-10	-0.4167	65535	16	-1.0
1.0	-10	-2.9167	65535	32	-1.0
1.0	-10	-7.0833	65535	64	-1.0
1.0	-10	-9.5833	65535	128	-1.0
1.0	-9.5833	-10	65535	64	-0.95833
1.0	-7.0833	-10	65535	32	-0.70833
1.0	-2.9167	-10	65535	16	-0.29167
1.0	-0.4167	-10	65535	8	-0.04167
1.0	0	-9.5833	65535	4	0
1.0	0	-7.0833	65535	2	0
1.0	0	-2.9167	65535	1	0
1.0	0	-0.4167	65535	0	0

8.4.7 PT Trajectory Verification and Execution

Here are four functions to verify or execute a PT trajectory :

- **MultipleAxesPTVerification**(): Verifies a PT trajectory data file.
- **MultipleAxesPTVerificationResultGet**(): Returns the results of the last trajectory verification call, positioner by positioner. This function works only after a MultipleAxesPTVerification().
- **MultipleAxesPTExecution**(): Executes a PT trajectory.
- **MultipleAxesPTParametersGet**(): Returns the trajectory's current execution parameters. This function works only while executing a trajectory.

The function **MultipleAxesPTVerification**() can be executed at any moment and is independent of the trajectory execution. This function does the following:

- Checks the trajectory file for data and syntax coherence.
- Simulates the trajectory to determine the positioner's travel requirements in negative and positive directions and the maximum allowed speed and acceleration for each positioner. This function determines whether the trajectory is executable.
- If all is OK, it returns an "OK" (value 0). Otherwise it returns a corresponding error. An error for instance is reported if one of the positioner's speed or acceleration reached during the trajectory exceeds the maximum allowed speed or acceleration.

The function **MultipleAxesPTVerificationResultGet**() can be executed only after a MultipleAxesPTVerification(). It returns the trajectory limits for each positioner, which are the travel requirements in positive and negative directions, the achieved maximum speed and acceleration.

To execute a PT trajectory, send the function **MultipleAxesPTExecution**() while specifying the file name and the number of cycles. This function does not verify the trajectory's coherence or geometric conditions (exceeding any positioner's min. or max. travel, speed or acceleration) before execution, so users must be careful when executing a trajectory without verifying the trajectory first. In case of an error during execution, because of bad data or because of a following error, the motion group will make an emergency stop and will go to the disabled state.

Finally, the function **MultipleAxesPTParametersGet**() returns the trajectory name and the number of the trajectory element that is currently being executed. This function returns an error if the trajectory is not executing.

8.4.8 Example of how to use PVT functions

MultipleAxesPTVerification (MultipleGroup, PTExample.trj)

This function returns a 0 if the trajectory is executable.

MultipleAxesPTVerificationResultGet (MultipleGroup.Pos1, *Name, *NegTravel, *PosTravel, *MaxSpeed, *MaxAcceleration)

This function returns the name of the trajectory verified with the last functions call of MultipleAxesPTVerification to the motion group MultipleGroup (PTExample.trj) and the trajectory limits for the positioner MultipleGroup.Pos1. These trajectory limits are: the negative or left travel requirement, the positive or right travel requirement, the achieved maximum speed and acceleration. Make sure that these trajectory limits (required negative and positive travel, speed and acceleration) are within the soft limits of the stages defined in the stages.ini file (section Travel: MinimumTargetPosition, MaximumTargetPosition and section Profiler: MaximumVelocity, MaximumAcceleration).

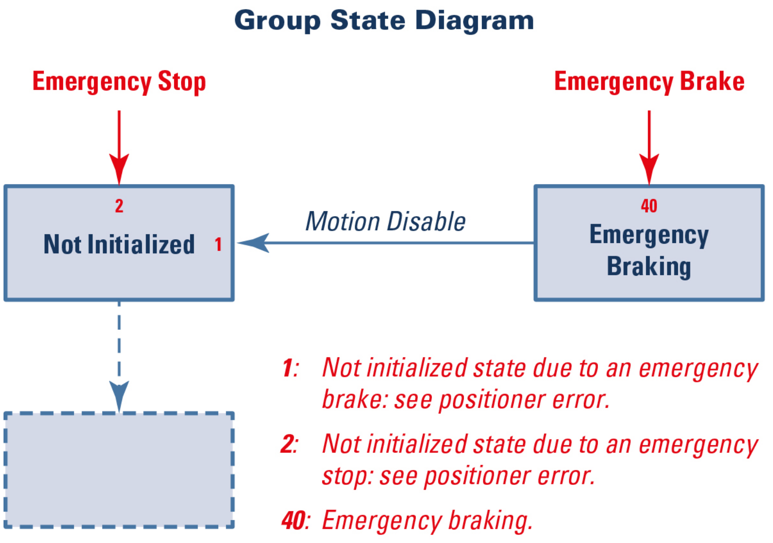
MultipleAxesPTExecution (MultipleGroup, PTExample.trj, 5)

This function executes the trajectory, PTExample.trj, five times.

MultipleAxesPTParametersGet (MultipleGroup, *FileName, *ElementNumber)

Returns the currently executed trajectory file name (PTExample.trj) and the number of the currently executed trajectory element.

9.0 Emergency Brake and Emergency Stop Cases


Group State Diagram
Emergency Stop
Emergency Brake
2
Not Initialized
1
Motion Disable
40
Emergency Braking
1: Not initialized state due to an emergency brake: see positioner error.
2: Not initialized state due to an emergency stop: see positioner error.
40: Emergency braking.

NOTE

Emergency brake brings a stage to a stop, then sets the motor power to Off. Emergency stop: sets motor power to Off only.

Emergency Brake occurs when:

Case	Error
Standard end of run driver safety supervisor Standard limit and home encoder safety supervisor Standard limit and limit encoder safety supervisor	• Plus end of run is detected • Minus end of run is detected
Line arc trajectory execution	• Error occurs when reading or getting trajectory parameters • The user target position is outside the MinimumTargetPosition and MaximumTargetPosition value • Actual positioner velocity is greater than the *MaximumVelocity* value
Spline trajectory execution	• Error occurs when reading or getting trajectory parameters • The user target position is outside the *MinimumTargetPosition* and *MaximalTargetPosition* value • Actual positioner velocity is greater than the *MaximumVelocity* value
PVT trajectory execution	• Error occurs when reading or getting trajectory parameters • Error occurs during trajectory execution • The user target position is outside the *MinimumTargetPosition* and *MaximalTargetPosition* values • Actual positioner velocity is greater than the *MaximumVelocity* value
S-gamma motion of a slave	• Group positioner is not in the home process, • **And** end of run detection is enabled • **And** the group is not a spindle group • **And** the user target position is outside the *MinimumTargetPosition* and *MaximalTargetPosition* value

Emergency Stop occurs when:

Case	Error
AquadBEncoder fault	- Quadrature error - FOC fault (over run error)
Analog interpolator encoder fault	- Quadrature error - FOC fault - Sin Cos radius error
AnalogAccelerationMotorInterface AnalogDualSinAccelerationMotorInterface AnalogPositionMotorInterface AnalogSinAccelerationMotorInterface AnalogStepperPositionMotorInterface AnalogVelocityMotorInterface AnalogVoltageMotorInterface DigitalStepperPositionMotorInterface AnalogSinAccelerationLMIMotorInterface AnalogAccelerationTZMotorInterface AnalogPositionPiezoMotorInterface	- Driver fault
Single Axis with clamping control Single Axis theta	- Unclamped state

10.0 Compensation

10.1 Definitions

The XPS controller features different compensation methods that improve the performance of a motion system namely, Backlash, Linear error, Positioner mapping, XY mapping and XYZ mapping. To understand the different compensation methods it is important to define terms used to calculated the compensation.

TargetPosition: The TargetPosition is the position where the positioner must be after the completion of a move.

SetpointPosition : The SetpointPosition is the theoretical position commanded to the servo loop. It is the position where the positioner should be, during and after the end of the move.

CurrentPosition: The CurrentPosition is the current physical position of the positioner. It is equal to the encoder position after all compensations (backlash, linear error and mapping) have been taken into account.

SetpointVelocity: The SetpointVelocity is calculated by the motion profiler and represents the “theoretical” velocity to reach during the motion.

SetpointAcceleration: The SetpointAcceleration is calculated by the motion profiler and represents the “theoretical” acceleration to reach during the motion.

FollowingError: The FollowingError is the difference between the CurrentPosition and the SetpointPosition.

A short description of the different compensation methods that improves the performance of a motion system follows:

Backlash compensation: The use of backlash compensation improves the bi-directional repeatability and accuracy of a motion device that has mechanical play. Backlash compensation is applicable to all positioners, but it is not available in all motion modes. When backlash compensation is activated, the XPS controller adds a user-defined BacklashValue to the TargetPosition to calculate a new target position whenever the direction of motion reverses. This internally used new target position is then the basis for the calculations of the motion profiler. No modification of the actual target is performed.

Linear error compensation: The linear error compensation helps improve the accuracy of a motion device by eliminating linear error sources. Linear errors can be caused by screw pitch errors, linear increasing angular deviations (abbe errors), thermal effects or cosine errors (misalignment between the feedback device and the direction of motion). Linear error compensation is applicable to all positioners. Its value is defined in the stages.ini. When set to other than zero, the encoder positions are compensated by this value. Linear error compensation can be used in conjunction with other compensation. For this reason, keep in mind the effects of using linear error compensation in addition to other compensation methods.

Positioner mapping: In contrast to the linear error compensation, positioner mapping also allows compensation for nonlinear error sources. Positioner mapping is done by sending a compensation table to the XPS controller and configuring the needed settings in the stages.ini. Positioner mapping is available with all positioners and works in parallel with other compensations except for the backlash compensation method. Better accuracy performance is achievable with linear compensation and positioner mapping combined.

XY mapping: XY mapping is only available with XY groups. It allows compensation for all errors of an XY group at any position of the XY group by sending two compensation tables to the XPS controller (x and y compensations mapped to x and y positions). The XY mapping is dynamically taken into account on the corrector loop of the XPS controller. XY mapping works in parallel to other compensation methods.

Keep in mind that the results of XY mapping may not be the same as those of Positioner mapping or linear compensation alone.

XYZ mapping: XYZ mapping is only available with XYZ groups. It compensates for all errors of an XYZ group at any position of the XYZ group by sending three compensation files to the XPS controller (x compensations mapped to x, y, and z positions, and so on). The XYZ mapping is dynamically taken into account on the corrector loop of the XPS controller. XYZ mapping works in parallel to other compensation methods. Keep in mind that the results of XYZ mapping may not be the same as those of Positioner mapping or linear compensation alone.

TargetPosition, SetpointPosition & CurrentPosition are accessible via function and Gathering (Data Collection).

SetpointVelocity, SetpointAcceleration & FollowingError are accessible via Gathering (Data Collection).

10.2 Backlash Compensation

Backlash compensation is applicable on all positioners, but works only under certain conditions:

- The “HomeSearchSequenceType” in the stages.ini must be different from “CurrentPositionAsHome”.
- Backlash compensation is not compatible with positioner mapping. So for positioners with backlash compensation, it is not allowed to have an entry for “PositionerMappingFileName” in the stages.ini.
- Backlash compensation is not compatible with trajectories (Line-Arc, Spline, PVT), jog or analog tracking. So it is not possible to execute any trajectory, to use the jog mode or to enable the analog tracking with any motion group that contains positioners with backlash compensation enabled.

After the above has been taken into consideration, a number of steps need to be taken to enable backlash compensation. First of all, there must be a value larger than 0 for “backlash” in the stages.ini. But this setting does not automatically enable backlash compensation. To do so, send the function **PositionerBacklashEnable**() while the motion group, which includes the positioner is disabled. To disable backlash compensation (for instance to execute a jog motion or to use analog tracking), use the function **PositionerBacklashDisable**(**).** The value for backlash compensation can be changed at any time with the function **PositionerBacklashSet**(**).** The new value for the backlash will be taken into account with the next following move. Finally, the function **PositionerBacklashGet**() returns the current value of the backlash and the backlash status (“enabled” or “disabled”).

For backlash setting to remain set after power down, the stages.ini file must be modified with the value desired.

Example

In the Backlash section of the stages.ini file, set a value greater than or equal to 0:

```
;--- Backlash
Backlash = 5                            ; units
```

This example shows the sequence of functions that enable backlash compensation:

PositionerBacklashEnable (MyGroup.MyPositioner)

GroupInitialize (MyGroup)

GroupHomeSearch (MyGroup)

…

PositionerBacklashSet (MyGroup.MyPositioner, 10)

PositionerBacklashGet (MyGroup.MyPositioner, *Backlash, *Status)

Returns the backlash value (10) and the backlash status (Enable).

…

PositionerBacklashDisable (MyGroup.MyPositioner)

10.3 Linear Error Correction

Linear error correction is applicable on all positioners and works in parallel with any other compensation. To use linear error correction, you need to set a value for “LinearErrorCorrection” in the stages.ini. When set, the corrected positions are calculated in the following way:

Corrected position = EncoderPosition x (1 + LinearEncoderCorrection/10^6)

The value of LinearEncoderCorrection is specified in ppm (parts per million). The correction is applied relative to the physical home position of the positioner (the Encoder position by definition is set to the HomePreset value at the home position). This hardware reference for linear error correction has the advantage of being independent of the value of the HomePreset.

<u>Example</u>

In the Encoder section of the stages.ini file, set a value other than 0,
but -0.5 x 10^6 < value < 0.5 x 10^6, in parameter LinearEncoderCorrection:

```
;--- Encoder
EncoderType =AquadB
EncoderResolution = 0.001          ; unit
LinearEncoderCorrection =5         ; ppm
```

10.4 Positioner Mapping

Positioner mapping corrects for any nonlinear errors of a positioner. Positioner mapping is applicable on all positioners and can be used with other compensations except backlash compensation. The positioner mapping is applied after linear correction is done.

The positioner mapping data is defined in a text file. Each line of that file represents one set of data. Each set of data is composed of the position and the error at this position. The separator between the two data entries in each line is a tab. All positions are relative to the physical home position of the positioner. The data file must contain the line “0 0”, which means that the error at the home position is 0. This hardware reference for positioner mapping has the advantage of being independent of the value of the HomePreset.

The following shows the general structure of such a data file:

```
PosMin          Error 0
Pos 1           Error 1
Pos 2           Error 2
…               …
0               0
…               …
PosMax          Error LineNumber-1
```

To activate positioner mapping, the mapping file must be in the ..\admin\config directory of the XPS controller and the following settings must be configured in the stages.ini:

- **PositionerMappingFileName:** Name of the mapping file.
- **PositionerMappingLineNumber:** Number of lines of the file.
- **PositionerMappingMaxPositionError:** Maximum absolute error in the file must be larger than any entry in the mapping file. To be read properly, the error entries must be in index format, see example.

PositionerMappingLineNumber and PositionerMappingMaxPositionError are only used to check for the correctness of the mapping file.

Example

The following shows an example of a positioner mapping data file:
PosMapping.txt

```
-3.00    -0.00125
-2.00    -0.00112
-1.00    -0.00137
 0.00     0.00000
 1.00     0.00140
 2.00     0.00145
 3.00     0.00154
```

Define the positioner mapping in the stages.ini file:

```
;--- Backlash
Backlash =0                          ; unit

;--- Positioner mapping
PositionerMappingFileName = PosMapping.txt
PositionerMappingLineNumber = 7
PositionerMappingMaxPositionError = 0.00154

;--- Travels
MinimumTargetPosition =-3            ; unit
HomePreset =0                        ; unit
MaximumTargetPosition =3             ; unit
```

NOTE

These travel limits must be equal to or be within the positioner's limit positions of the mapping file (+3 and -3 in the above example).

Use of the functions:

- **GroupInitialize(MyGroup)**
- **GroupHomeSearch(MyGroup)**
- **GroupMoveAbsolute(MyGroup.Positioner, 0.25)**

The mapping file must at least cover the minimum and the maximum travel of the positioner. It must cover MinimumTargetPosition and MaximumTargetPosition parameters defined in the stages.ini, section Travels. In the example above, the travel of the positioner can not be larger than ±3 units, but it can be smaller than this. The units for the data are the same as defined by EncoderResolution in the stages.ini. The data reads as follows: the corrected position at position 3.00 units is 2.99846 units (3.00 - 0.00154). Between two data points, the XPS controller performs a linear interpolation of the error. The corrected position at position 0.25 units is 0.24965 units (0.25 - 0.00140*0.25/1).

NOTE

Mapping is a function implemented within the controller to correct positioning errors. Once activated, mapping is transparent to the user. The function GroupPositionCurrentGet doesn't return 0.24965 (0.25 - 0.00140*0.25/1) but 0.25.

10.5 XY Mapping

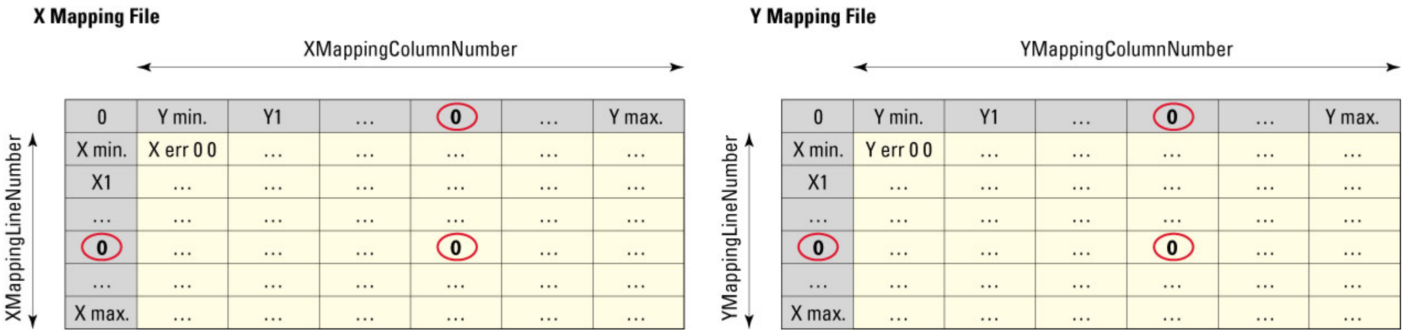
XY mapping is only available to XY groups. It compensates for all errors of an XY group at any position of that XY group. XY mapping can be used in addition to other compensations, including positioner mapping. So care must be taken about the unwanted cross-effects of using XY mapping and other compensation at the same time.

XY mapping is defined by 2 compensation tables, in text file format, each for X and Y errors. In each of these files:

- The first entry in that file must be 0 (zero).
- The **first column** specifies the **X positions** (X being the first positioner of the XY group).
- The **first row** specifies the **Y positions**.
- Each cell represents the error for that X, Y position.
- The **separator** between the different data in each row is the **tab**.
- All positions are relative to the physical home position of the XY group.
- The data files must contain the X position = 0 and the Y position = 0.
- The **error at X = Y = 0 must be 0**, which means that the error at the home position is 0.

This hardware reference for XY mapping has the advantage of being independent of the value of the HomePreset.

The following shows the structure of such mapping files:



0	Y min.	Y1	…	0	…	Y max.
X min.	X err 0 0	…	…	…	…	…
X1	…	…	…	…	…	…
…	…	…	…	…	…	…
0	…	…	…	0	…	…
…	…	…	…	…	…	…
X max.	…	…	…	…	…	…

0	Y min.	Y1	…	0	…	Y max.
X min.	Y err 0 0	…	…	…	…	…
X1	…	…	…	…	…	…
…	…	…	…	…	…	…
0	…	…	…	0	…	…
…	…	…	…	…	…	…
X max.	…	…	…	…	…	…

Figure 34: XY mapping files.

NOTE

Error in X = Y = 0 must be 0. This value in the file corresponds to the HomePreset position in the XY group reference.

To activate XY mapping, the mapping files must be in the ..\admin\config directory of the XPS controller and the following settings must be configured in the **system.ini**:

- **XMappingFileName:** Name of the mapping file.
- **XMappingLineNumber:** Total number of lines of that file.
- **XMappingColumnNumber:** Total number of columns of that file.
- **XMappingMaxPositionError:** Maximum absolute error in that file as shown in the tables below (any value larger than the actual largest value in that file will be accepted as well).
- **YMappingFileName:** Name of the mapping file.
- **YMappingLineNumber:** Total number of lines of that file.

- **YMappingColumnNumber:** Total number of columns of that file.
- **YMappingMaxPositionError:** Maximum absolute error in that file (any value larger than the actual largest value in that file will be accepted as well).

The X(Y)MappingLineNumber, X(Y)MappingColumnNumber and X(Y)MappingMaxPositionError are only used to check for the correctness of the mapping file.

Example

The following shows an example of the X and Y mapping files:

Matrix X: XYMapping_X.txt

0	-3.00	-2.00	-1.00	0.00	1.00	2.00	3.00
-3.00	-0.00192	-0.00534	-0.00254	0.00023	0.00254	0.00534	0.00192
-2.00	-0.00453	-0.00322	-0.00676	0.00049	0.00676	0.00322	0.00453
-1.00	-0.00331	-0.00845	-0.00769	0.00102	0.00769	0.00845	0.00331
0.00	-0.00787	-0.00228	-0.00787	0	0.00787	0.00228	0.00787
1.00	-0.00232	-0.00210	-0.00342	0.00089	0.00342	0.00210	0.00232
2.00	-0.00134	-0.00308	-0.00675	0.00101	0.00675	0.00308	0.00134
3.00	-0.00789	-0.00148	-0.00234	0.00121	0.00234	0.00148	0.00789

Matrix Y: XYMapping_Y.txt

0	-3.00	-2.00	-1.00	0.00	1.00	2.00	3.00
-3.00	-0.00172	-0.00434	-0.00154	0.00013	0.00204	0.00234	0.00122
-2.00	-0.00433	-0.00222	-0.00376	0.00029	0.00636	0.00222	0.00353
-1.00	-0.00311	-0.00635	-0.00569	0.00089	0.00739	0.00245	0.00231
0.00	-0.00737	-0.00128	-0.00387	0	0.00567	0.00128	0.00387
1.00	-0.00212	-0.00110	-0.00142	0.00079	0.00332	0.00310	0.00132
2.00	-0.00114	-0.00208	-0.00375	0.00089	0.00375	0.00348	0.00122
3.00	-0.00689	-0.00128	-0.00134	0.00101	0.00232	0.00138	0.00689

Verify in the stages.ini for both stages:

```
;--- Travels
MinimumTargetPosition =-3          ; unit
HomePreset =0; unit
MaximumTargetPosition =3           ; unit
```

NOTE

The limit travels must be equal or within the X and Y limit positions of the mapping files, +3 and –3, respectively in this example.

Apply the following settings in the system.ini file:

```
;--- Mapping XY
XMappingFileName = XYMapping_X.txt
XMappingLineNumber = 7
XMappingColumnNumber = 7
XMappingMaxPositionError = 0.00845

YMappingFileName = XYMapping_Y.txt
YMappingLineNumber = 7
YMappingColumnNumber = 7
YMappingMaxPositionError = 0.00739
```

Use of the functions:

- **GroupInitialize(XY)**
- **GroupHomeSearch(XY)**
- **GroupMoveAbsolute(XY, 3, 2)**

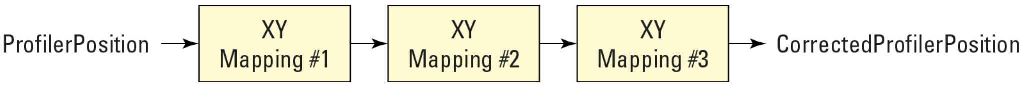
The mapping files must at least cover the minimum and the maximum travel of the XY group (they must cover the MinimumTargetPosition and the MaximumTargetPosition for the X and Y positioners, parameters defined in the stages.ini, see section Travels). So in the above example, the travel of the X and Y positioners can not be larger than ±3 units, but they can be smaller than this. The units for the data are the same as defined by the EncoderResolution in the stages.ini. The data reads as follows: at position X = 3.00 units, Y = 2.00 units the corrected X position is 2.99852 units (3.00 - 0.00148) and the corrected Y position is 1.99862 units (2.00 - 0.00138). Between two data points, the XPS controller performs a linear interpolation of the error. The two mapping files don't need to contain the same X and Y positions.

NOTE

Mapping is a function implemented within the XPS controller to correct positioning errors. When mapping is activated, it is transparent to the user. At position (X,Y) = (3.00, 2.00), the function GroupPositionCurrentGet(XY.X) doesn't return 2.99852 (3.00 - 0.00148) but 3.

10.5.1 Multiple XY Mappings in Series

Up to three XY mapping parameters are permitted for an XY group only for PP Firmware Version.



Below is the mapping section from system.ini for three XY mappings in series.

```
[GroupXY]

…

;--- Mapping XY #1
XMappingFileName =
XMappingColumnNumber=
XMappingLineNumber=
XMappingMaxPositionError=

YMappingFileName =
YMappingColumnNumber=
YMappingLineNumber=
YMappingMaxPositionError=

;--- Mapping XY #2 (PP only)
XMapping2FileName=
XMapping2ColumnNumber=
XMapping2LineNumber=
XMapping2MaxPositionError=

YMapping2FileName=
YMapping2ColumnNumber=
YMapping2LineNumber=
YMapping2MaxPositionError=

;--- Mapping XY #3 (PP only)
XMapping3FileName=
XMapping3ColumnNumber=
XMapping3LineNumber=
XMapping3MaxPositionError=

YMapping3FileName=
YMapping3ColumnNumber=
YMapping3LineNumber=
YMapping3MaxPositionError=
```

10.6 XYZ Mapping

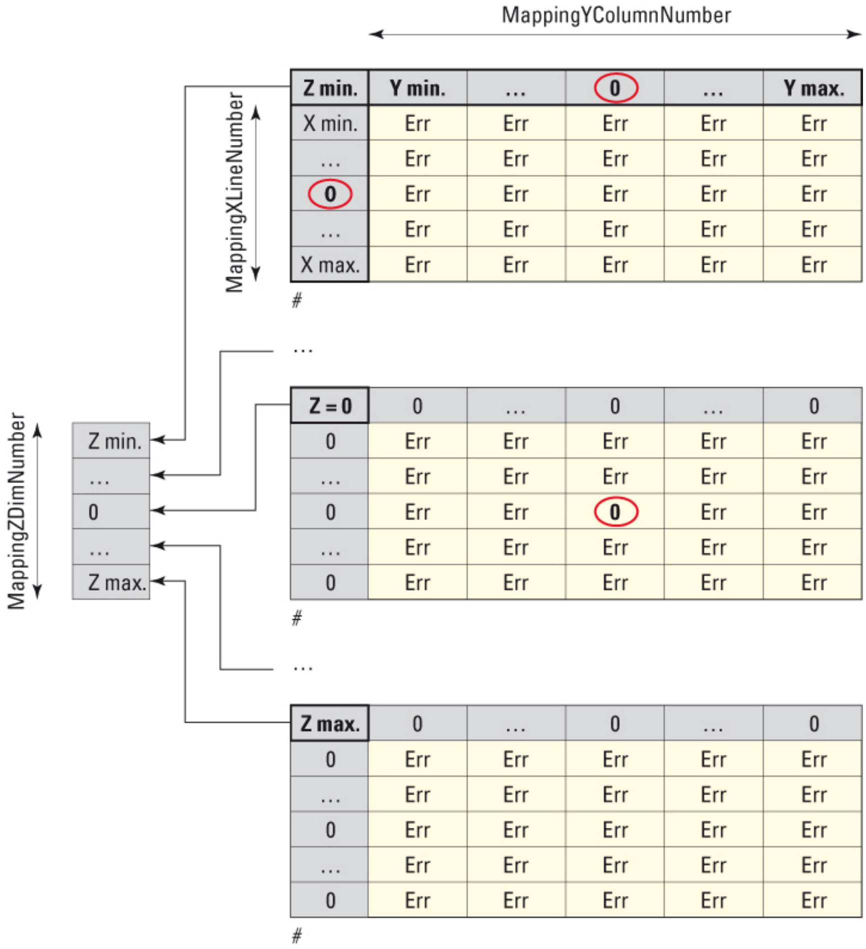
XYZ mapping is available only with XYZ groups. It compensates for all errors of an XYZ group at any position of that XYZ group. XYZ mapping can be used in conjunction with other compensations, including positioner mapping. Care must be taken to consider the effects when using XYZ mapping and other compensations at the same time.

XYZ mapping is defined by 3 compensation files (compensation for errors in X, Y or Z), in text format. Each of these files can be seen as the juxtaposition of successive tables where the first column of the first table contains the X positions; the first row of the first table contains the Y positions; and the first cell of each table contains one of the Z positions. Each table represents a plane defined by the Z position of the first cell. The separator between the different data in each row is a tab. For legibility, inserting an empty line between successive tables is recommended, but not mandatory. The other cells contain the corresponding error.

All positions are relative to the physical home position of the XYZ group. The data files must contain the X position = 0, the Y position = 0, and the Z position = 0. The error at X = Y = Z = 0 must be 0, which means that the error at the home position is 0. This hardware reference for XYZ mapping has the advantage of being independent of the value of the HomePreset.

Figure 35 shows the structure for the three mapping files for X, Y, and Z corrections:

- **XYZMappingCorrectionX.dat:** All Err entries are X errors (corrections for X).
- **XYZMappingCorrectionY.dat:** All Err entries are Y errors (corrections for Y).
- **XYZMappingCorrectionZ.dat:** All Err entries are Z errors (corrections for Z).



Z min.	Y min.	…	0	…	Y max.
X min.	Err	Err	Err	Err	Err
…	Err	Err	Err	Err	Err
0	Err	Err	Err	Err	Err
…	Err	Err	Err	Err	Err
X max.	Err	Err	Err	Err	Err

Z = 0	0	…	0	…	0
0	Err	Err	Err	Err	Err
…	Err	Err	Err	Err	Err
0	Err	Err	0	Err	Err
…	Err	Err	Err	Err	Err
0	Err	Err	Err	Err	Err

Z max.	0	…	0	…	0
0	Err	Err	Err	Err	Err
…	Err	Err	Err	Err	Err
0	Err	Err	Err	Err	Err
…	Err	Err	Err	Err	Err
0	Err	Err	Err	Err	Err

Figure 35: XYZ mapping files.
Error in each compensation file can either be Xerr, Yerr or Zerr.

NOTE

The error at X = Y = Z = 0 must be 0. This value in the file corresponds to the HomePreset positions in the XY group reference. A terminator (#) must be added at end of each table.

To activate XYZ mapping, the mapping files must be in the ..\admin\config directory of the XPS controller and the following settings must be configured in the system.ini:

- **XMappingFileName:** Name of the mapping file.
- **XMappingXLineNumber:** Total number of lines of each table including the header.
- **XMappingYColumnNumber:** Total number of columns.
- **XMappingZDimNumber:** Number of tables.
- **XMappingMaxPositionError:** Maximum absolute error in that file must be larger than any entry in the mapping file.
- **YMappingFileName:** Name of the mapping file.
- **YMappingXLineNumber:** Total number of lines of each table including header.
- **YMappingYColumnNumber:** Total number of columns.
- **YMappingZDimNumber:** Number of tables.
- **YMappingMaxPositionError:** Maximum absolute error in that file must be larger than any entry in the mapping file.
- **ZMappingFileName:** Name of the mapping file.

- **ZMappingXLineNumber:** Total number of lines of each table including header.
- **ZMappingYColumnNumber:** Total number of columns.
- **ZMappingZDimNumber:** Number of tables.
- **ZMappingMaxPositionError:** Maximum absolute error in that file must be larger than any entry in the mapping file.

The X(Y,Z)MappingXLineNumber, X(Y,Z)MappingYColumnNumber, X(Y,Z)MappingZDimNumber and X(Y,Z)MappingMaxPositionError are only used to check for the correctness of the mapping file.

Example

The following example shows the X error mapping files for an XYZ mapping. Note that it is not necessary to repeat the XY coordinates in the table, Z = -1 to the other tables, Z = 0 and Z = 1.

Matrix of X errors: XYZMapping_X.txt

```
 -1.00      -3.00      -2.00      -1.00       0.00       1.00       2.00       3.00
 -3.00      -0.00192   -0.00534    0.00254    0.00125   -0.00137    0.00110    0.00123
 -2.00       0.00453   -0.00322    0.00376   -0.00412   -0.00258   -0.00111   -0.00287
 -1.00      -0.00331    0.00445   -0.00769   -0.00126   -0.00153    0.00298    0.00487
  0.00      -0.00787    0.00228   -0.00787    0.00320    0.00154   -0.00169   -0.00369
  1.00       0.00232    0.00210   -0.00342    0.00169    0.00265    0.00169    0.00125
  2.00      -0.00134    0.00308    0.00275   -0.00369    0.00337   -0.00214   -0.00456
  3.00       0.00189   -0.00148    0.00234    0.00458   -0.00333    0.00152    0.00335
#
  0          0          0          0          0          0          0          0
  0         -0.00192   -0.00534    0.00254    0.00125   -0.00137    0.00110    0.00123
  0          0.00453   -0.00322    0.00376   -0.00412   -0.00258   -0.00111   -0.00287
  0         -0.00331    0.00445   -0.00769   -0.00126   -0.00153    0.00298    0.00487
  0         -0.00787    0.00228   -0.00787    0          0.00154   -0.00169   -0.00369
  0          0.00232    0.00210   -0.00342    0.00169    0.00265    0.00169    0.00125
  0         -0.00134    0.00308    0.00275   -0.00369    0.00337   -0.00214   -0.00456
  0          0.00189   -0.00148    0.00234    0.00458   -0.00333    0.00152    0.00335
#
  1.00       0          0          0          0          0          0          0
  0         -0.00192   -0.00534    0.00254    0.00125   -0.00137    0.00110    0.00123
  0          0.00453   -0.00322    0.00376   -0.00412   -0.00258   -0.00111   -0.00287
  0         -0.00331    0.00445   -0.00769   -0.00126   -0.00153    0.00298    0.00487
  0         -0.00787    0.00228   -0.00787    0.00320    0.00154   -0.00169   -0.00369
  0          0.00232    0.00210   -0.00342    0.00169    0.00265    0.00169    0.00125
  0         -0.00134    0.00308    0.00275   -0.00369    0.00337   -0.00214   -0.00456
  0          0.00189   -0.00148    0.00234    0.00458   -0.00333    0.00152    0.00335
#
```

Matrix of Y errors: XYZMapping_Y.txt

```
  -1.00    -3.00     -2.00     -1.00     0.00      1.00      2.00      3.00
  -3.00    -0.00190  -0.00530  0.00190   0.00125   -0.00190  0.00530   0.00190
  -2.00    -0.00190  -0.00530  0.00190   0.00125   -0.00190  0.00530   0.00190
  -1.00    -0.00190  -0.00530  0.00190   0.00125   -0.00190  0.00530   0.00190
   0.00    -0.00190  -0.00530  0.00190   0.00125   -0.00190  0.00530   0.00190
   1.00    -0.00190  -0.00530  0.00190   0.00125   -0.00190  0.00530   0.00190
   2.00    -0.00190  -0.00530  0.00190   0.00125   -0.00190  0.00530   0.00190
   3.00    -0.00190  -0.00530  0.00190   0.00125   -0.00190  0.00530   0.00190
#
   0        0         0         0         0         0         0         0
   0       -0.00192  -0.00534  0.00254   0.00125   -0.00137  0.00110   0.00123
   0       -0.00192  -0.00534  0.00254   0.00125   -0.00137  0.00110   0.00123
   0       -0.00192  -0.00534  0.00254   0.00125   -0.00137  0.00110   0.00123
   0       -0.00192  -0.00534  0.00254   0         -0.00137  0.00110   0.00123
   0       -0.00192  -0.00534  0.00254   0.00125   -0.00137  0.00110   0.00123
   0       -0.00192  -0.00534  0.00254   0.00125   -0.00137  0.00110   0.00123
   0       -0.00192  -0.00534  0.00254   0.00125   -0.00137  0.00110   0.00123
#
   1.00     0         0         0         0         0         0         0
   0       -0.00192  -0.00534  0.00254   0.00125   -0.00137  0.00110   0.00123
   0       -0.00192  -0.00534  0.00254   0.00125   -0.00137  0.00110   0.00123
   0       -0.00192  -0.00534  0.00254   0.00125   -0.00137  0.00110   0.00123
   0       -0.00192  -0.00534  0.00254   0.00125   -0.00137  0.00110   0.00123
   0       -0.00192  -0.00534  0.00254   0.00125   -0.00137  0.00110   0.00123
   0       -0.00192  -0.00534  0.00254   0.00125   -0.00137  0.00110   0.00123
   0       -0.00192  -0.00534  0.00254   0.00125   -0.00137  0.00110   0.00123
#
```

Matrix of Z errors: XYZMapping_Z.txt

```
  -1.00    -3.00     -2.00     -1.00     0.00      1.00      2.00      3.00
  -3.00    -0.0002   -0.0002   0.0002    0.0002    -0.0002   -0.0002   0.0002
  -2.00    -0.0003   -0.0003   0.0003    0.0003    -0.0003   -0.0003   0.0003
  -1.00    -0.0002   -0.0002   0.0002    0.0002    -0.0002   -0.0002   0.0002
   0.00    -0.0003   -0.0003   0.0003    0.0003    -0.0003   -0.0003   0.0003
   1.00    -0.0002   -0.0002   0.0002    0.0002    -0.0002   -0.0002   0.0002
   2.00    -0.0003   -0.0003   0.0003    0.0003    -0.0003   -0.0003   0.0003
   3.00    -0.0002   -0.0002   0.0002    0.0002    -0.0002   -0.0002   0.0002
#
   0        0         0         0         0         0         0         0
   0       -0.0002   -0.0002   0.0002    0.0002    -0.0002   -0.0002   0.0002
   0       -0.0003   -0.0003   0.0003    0.0003    -0.0003   -0.0003   0.0003
   0       -0.0002   -0.0002   0.0002    0.0002    -0.0002   -0.0002   0.0002
   0       -0.0003   -0.0003   0.0003    0         -0.0003   -0.0003   0.0003
   0       -0.0002   -0.0002   0.0002    0.0002    -0.0002   -0.0002   0.0002
   0       -0.0003   -0.0003   0.0003    0.0003    -0.0003   -0.0003   0.0003
   0       -0.0002   -0.0002   0.0002    0.0002    -0.0002   -0.0002   0.0002
#
   1.00     0         0         0         0         0         0         0
   0       -0.0002   -0.0002   0.0002    0.0002    -0.0002   -0.0002   0.0002
   0       -0.0003   -0.0003   0.0003    0.0003    -0.0003   -0.0003   0.0003
   0       -0.0002   -0.0002   0.0002    0.0002    -0.0002   -0.0002   0.0002
   0       -0.0003   -0.0003   0.0003    0.0003    -0.0003   -0.0003   0.0003
   0       -0.0002   -0.0002   0.0002    0.0002    -0.0002   -0.0002   0.0002
   0       -0.0003   -0.0003   0.0003    0.0003    -0.0003   -0.0003   0.0003
   0       -0.0002   -0.0002   0.0002    0.0002    -0.0002   -0.0002   0.0002
#
```

Verify in the corresponding sections of the stages.ini:

For the X axis:

```
;--- Travels
MinimumTargetPosition =-3          ; unit
HomePreset =0; unit
MaximumTargetPosition =3           ; unit
```

NOTE

The limit travels must be equal or within the X limit positions of the mapping files (shown here +3 and -3).

For the Y axis:

```
;--- Travels
MinimumTargetPosition =-3          ; unit
HomePreset =0; unit
MaximumTargetPosition =3           ; unit
```

NOTE

The limit travels must be equal or within the Y limit positions of the mapping files (shown here +3 and -3).

For Z axis:

```
;--- Travels
MinimumTargetPosition =-1          ; unit
HomePreset =0; unit
MaximumTargetPosition =1           ; unit
```

NOTE

The limit travels must be equal or within the Z limit positions of the mapping files (shown here +1 and -1).

In the system.ini file:

```
;--- Mapping XYZ
```

Represents the errors in the X axis.

```
XMappingFileName = XYZMapping_X.txt
XMappingXLineNumber = 7
XMappingYColumnNumber = 7
XMappingZDimNumber = 3
XMappingMaxPositionError = 0.00787
```

Represents the errors in the Y axis.

```
YMappingFileName = XYZMapping_Y.txt
YMappingXLineNumber = 7
YMappingYColumnNumber = 7
YMappingZDimNumber = 3
YMappingMaxPositionError = 0.00534
```

Represents the errors in the Z axis.

```
ZMappingFileName = XYZMapping_Z.txt
ZMappingXLineNumber = 7
ZMappingYColumnNumber = 7
ZMappingZDimNumber = 3
ZMappingMaxPositionError = 0.0003
```

Use of the functions:

- **GroupInitialize(XYZ)**
- **GroupHomeSearch(XYZ)**
- **GroupMoveAbsolute(XYZ, 3, 1, 1)**

The mapping files must at least cover the minimum and the maximum travel of the XYZ group (they must cover the MinimumTargetPosition and the MaximumTargetPosition for the X, Y and Z positioners, parameters defined in the stages.ini, see section Travels). So in the above example the travel of the X and Y positioners can not be larger than ±3 units, and the travel for the Z positioner can not be larger than ±1 unit. But the travel can be smaller than these. The unit of the data is the same as defined by EncoderResolution in the stages.ini. The data reads as follows: at position (X,Y,Z) = (3.00, 2.00, 1.00), the corrected X position is 2.99848 units (3.00 - 0.00152), the corrected Y position is 2.9989 units (3.00 - 0.00110) and the corrected Z position is 3.0002 units (3.00 + 0.0002). Between two datas, the XPS controller does a linear interpolation of the error. The three mapping files for X, Y, and Z don’t need to contain the same X, Y and Z positions.

NOTE

Mapping is a function implemented in the XPS controller to correct errors. But when mapping is activated, it is transparent to the user. At position (X,Y,Z) = (3.00, 1.00, 1.00), the function GroupPositionCurrentGet(XYZ.X) doesn’t return 3.00333 (3.00 + 0.00333) but returns 3.

11.0 Event Triggers

XPS event triggers work similar to IF/THEN statements in programming. “If” the **event** occurs, “then” an **action** is triggered. Programmers can trigger any action (from a list of possible actions, see section 11.2) at any event (from a large list of possible events, see section 11.1). It is also possible to trigger several actions with the same event. Furthermore, it is possible to link several events to an event configuration. In this case, all events must happen at the same time to trigger the action(s). It is comparable to a logic AND between the different events.

Some events are one-time events like “motion start”. They will trigger an action only once when the event occurs. Some other events have a duration like “motion state”. They will trigger the same action each time (as applicable) as long as the event occurs. For events with duration, the event can be also considered as a statement that is checked whether it is true or not. A third event category are the permanent events “Always” (always happens) and “Timer” (happens every nth servo cycle). They will trigger the action always on every nth servo cycle.

As the XPS controller provides the utmost flexibility in programming event triggers, the user must be careful and consider possible unwanted effects. Some events might have a duration although only one single action is asked. Some other events might never occur. This is especially true when linking several events to an event configuration. The different possible effects are illustrated in section 11.3 by a few examples.

To trigger an action with an event, the event and the associated action must first be configured using the functions **EventExtendedConfigurationTriggerSet**() and **EventExtendedConfigurationActionSet**(). Then, the event trigger is activated using the function **EventExtendedStart**(). When activated, the XPS controller checks for the event at each servo cycle (or each profiler cycle for those events that are motion related) and triggers the action when the event happens. Hence, the maximum latency between the event and the action is equal to the servo cycle (default value 125 µs) or equal to the profiler cycle time (default value 500 µs). For events with duration, it means that the same action is triggered at each servo cycle, i.e. every 125 µs, or at each profiler cycle, i.e. every 500 µs, as long as the event is happening.

Event triggers (and their associated actions) are automatically removed after the event configuration has happened at least once and is no longer true. The only exception is if the event configuration contains any of the permanent events “Always” or “Timer”. In this case the event trigger will always stay active. With the function **EventExtendedRemove**(), any event trigger can get removed.

The function **EventExtendedWait**() can be used to halt a process. It essentially blocks the socket until the event occurs. Once the event occurs, it is deleted. It requires a preceding function **EventExtendedConfigurationTriggerSet**() to define the event at which the process continues.

The functions **EventExtendedGet**() and **EventExtendedAllGet**() return details of the event and action configurations.

11.1 Events

General events are defined as “**Always**”, “**Immediate**” and “**Timer**”. With the event “Always”, an action is triggered each servo cycle, meaning every 125 µs for default value. For events that are defined as “Immediate”, an action is triggered once immediately (during the very next servo cycle). For the events defined as “Timer”, an action is triggered immediately and every nth servo cycle. Here, “n” corresponds to the “FrequencyTicks” defined in the function **TimerSet**(). There are five different timers available that can be selected by the actor (1…5) (Actor is the object that actions/events are linked to).

All events that are motion related (from MotionStart to TrajectoryPulseOutputState in the below table, except MotionDone) refer to the motion profiler of the XPS controller. By default, the motion profiler runs at a frequency of 2 kHz, or every 500 µs. Thus, events triggered by the motion profiler have a resolution of 500 µs. Consequently, events with duration, such as MotionState, will trigger an action every 500 µs. All motion related events, except MotionDone, have a category such as “Sgamma” or “Jog”. This category refers to the motion profiler. Here, SGamma refers to the profiler used with the function GroupMoveRelative and GroupMoveAbsolute and Jog refers to the profiler used in the Jogging state. The other event categories refer to trajectories. The separator between the category, the actor, and the event name is a dot (.).

NOTE

The table is not an exhaustive list. Refer to the Programmer's Manual for more information.

[Actor.]				[Category.]						Event Name	Parameters			
Group	Positioner	GPIO	TimerX	SGamma	Jog	XYLineArc	Spline	PVT	PT		1	2	3	4
										Immediate	0	0	0	0
										Always	0	0	0	0
			■							Timer	0	0	0	0
	■			■	■					MotionStart	0	0	0	0
	■			■	■					MotionEnd	0	0	0	0
	■			■	■					MotionState	0	0	0	0
	■			■						MotionDone	0	0	0	0
	■			■	■					ConstantVelocityStart	0	0	0	0
	■			■						ConstantVelocityEnd	0	0	0	0
	■			■	■					ConstantVelocityState	0	0	0	0
	■			■						ConstantAccelerationStart	0	0	0	0
	■			■						ConstantAccelerationEnd	0	0	0	0
	■			■						ConstantAccelerationState	0	0	0	0
	■			■						ConstantDecelerationStart	0	0	0	0
	■			■						ConstantDecelerationEnd	0	0	0	0
	■			■						ConstantDecelerationState	0	0	0	0
■						■	■	■	■	TrajectoryStart	0	0	0	0
■						■	■	■	■	TrajectoryEnd	0	0	0	0
■						■	■	■	■	TrajectoryState	0	0	0	0
■						■	■	■	■	ElementNumberStart	Element index	0	0	0
■						■	■	■	■	ElementNumberState	Element index	0	0	0
■						■	■	■	■	TrajectoryPulse	0	0	0	0
■						■	■	■	■	TrajectoryPulseState	0	0	0	0
		■								DILowState	Bit index	0	0	0
		■								DIHighState	Bit index	0	0	0
		■								DILowHigh	Bit index	0	0	0
		■								DIHighLow	Bit index	0	0	0
		■								DIToggled	Bit index	0	0	0
		■								ADCHighLimit	Value	0	0	0
		■								ADCLowLimit	Value	0	0	0
		■								ADCInWindow	min	max	0	0
		■								ADCOutWindow	min	max	0	0
	■									PositionerError	Mask	0	0	0
	■									PositionerHardwareStatus	Mask	0	0	0
	■									ExcitationSignalStart	0	0	0	0

	■									ExcitationSignalEnd	0	0	0	0
	■									WarningFollowingError	0	0	0	0
	■									WaitForPositionLeftToRight	Target position	0	0	0
	■									WaitForPositionRightToLeft	Target position	0	0	0
	■									WaitForPositionToggled	Target position	0	0	0
										DoubleGlobalArrayEqual	Global variable number	value	0	0
										DoubleGlobalArrayDifferent	Global variable number	value	0	0
										DoubleGlobalArrayInferiorOrEqual	Global variable number	value	0	0
										DoubleGlobalArraySuperiorOrEqual	Global variable number	value	0	0
										DoubleGlobalArrayInferior	Global variable number	value	0	0
										DoubleGlobalArraySuperior	Global variable number	value	0	0
										DoubleGlobalArrayInWindow	Global variable number	min	max	0
										DoubleGlobalArrayOutWindow	Global variable number	min	max	0

An event is entirely composed of:

[Actor].[Category].Event Name, Parameter1, Parameter2, Parameter3, Parameter4

Not all event names have a preceding actor and category, but all events have four parameters, even though some parameters are not needed. For these parameters, it is still required to use zero (0) as default.

To define an Event, use the function EventExtendedConfigurationTriggerSet().

Examples

EventExtendedConfigurationTriggerSet (MyGroup.MyPositioner.SGamma.MotionStart, 0, 0, 0, 0)

In this case, the actor is a positioner (MyGroup.MyPositioner) and the event has a category. The event happens when the next motion with the SGamma profiler on the positioner MyGroup.MyPositioner starts. After the motion has started, the event is removed.

EventExtendedConfigurationTriggerSet (MyGroup.XYLineArc.ElementNumberStart, 5, 0, 0, 0)

In this case, the actor is a group (MyGroup) and the event has a category. The event happens when the trajectory element number 5 on the next LineArc trajectory on this group starts.

EventExtendedConfigurationTriggerSet (GPIO2.ADC2.ADCHighLimit, 3, 0, 0, 0)

In this case, the actor is a GPIO name (GPIO2.ADC2) and the event has no category. The event happens when the voltage on the GPIO.ADC2 exceeds 3 Volts.

It is also possible to link different events to an event configuration. The same function EventExtendedConfigurationTriggerSet() is used, and the different events are just separated by a comma. The event combination happens when all individual events happen at the same time. It is comparable to a logic AND between the different events.

Examples

EventExtendedConfigurationTriggerSet (GPIO2.ADC2.ADCHighLimit, 3, 0, 0, 0, MyGroup.MyPositioner.SGamma.MotionState, 0, 0, 0, 0)

This event will happen when the voltage of the GPIO.ADC2 exceeds 3 Volts during a SGamma motion of the MyGroup.MyPositioner.

EventExtendedConfigurationTriggerSet (Always, 0, 0, 0, 0, MyGroup.MyPositioner.SGamma.MotionStart, 0, 0, 0, 0)

This event will happen during each SGamma motion starts of the positioner MyGroup.MyPositioner. The addition of the event Always has the effect of keeping the event after the next motion has been started (see differences compared to the first example above).

The exact meaning of the different events and event parameters are as follows:

Always: Triggers an action ALWAYS, means at each servo cycle.

Event parameter 1 to 4 = 0 by default.

NOTE: This event is PERMANENT until the next reboot. Call the EventExtendedRemove function to remove it.

Immediate: Triggers an action IMMEDIATELY, meaning once during the very next servo cycle:

Event parameter 1 to 4 = 0 by default.

NOTE: This event is TEMPORARY.

Timer: Triggers an action every nth servo cycle, where n is defined with the function TimerSet.

Event parameter 1 to 4 = 0 by default.

NOTE: This event is PERMANENT until the next reboot. Call the EventExtendedRemove function to remove it.

MotionDone: Triggers an action when a position is reached.

Event parameter 1 to 4 = 0 by default.

For the exact definition of MotionDone, please refer to section 7.5.

ConstantVelocityStart: Triggers an action when constant velocity is reached. Event parameter 1 to 4 = 0 by default.

ConstantVelocityEnd: Triggers an action when constant velocity is finished. Event parameter 1 to 4 = 0 by default.

ConstantVelocityState: Triggers an action during constant velocity. Event parameter 1 to 4 = 0 by default.

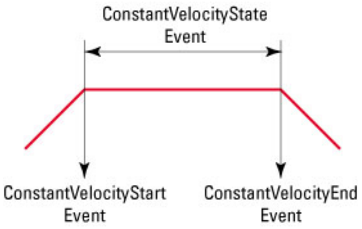


Figure 36: Constant velocity event.

ConstantAccelerationStart: Triggers an action when constant acceleration is reached. Event parameter 1 to 4 = 0 by default.

ConstantAccelerationEnd: Triggers an action when constant acceleration is finished. Event parameter 1 to 4 = 0 by default.

ConstantAccelerationState: Triggers an action during constant acceleration. Event parameter 1 to 4 = 0 by default.

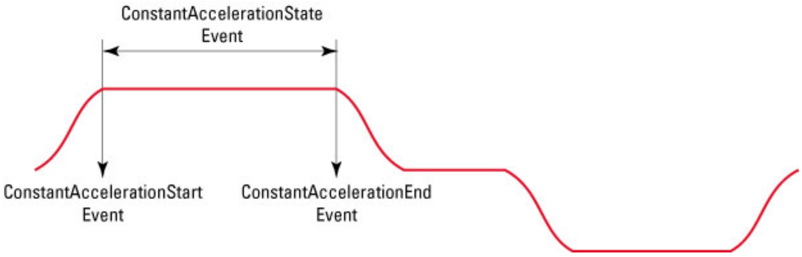


Figure 37: Constant acceleration event.

The same definition applies to ConstantDecelerationStart, ConstantDecelerationEnd and ConstantDecelerationState.

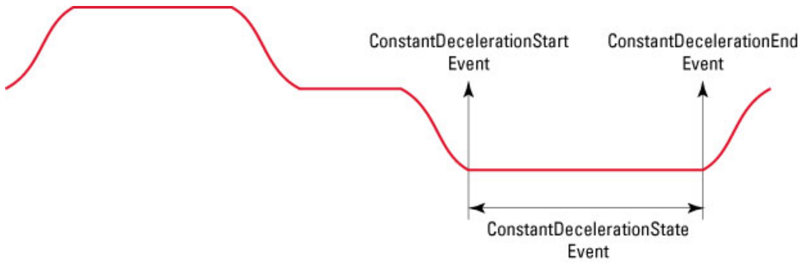


Figure 38: Constant deceleration event.

MotionStart: Triggers an action when motion starts. Event parameter 1 to 4 = 0 by default.

MotionEnd: Trigger an action when motion is ended. Event parameter 1 to 4 = 0 by default. Note, MotionEnd refers to the end of the theoretical motion which is not the same as the definition of MotionDone (see section 7.5).

MotionState: Triggers an action during motion. Event parameter 1 to 4 = 0 by default.

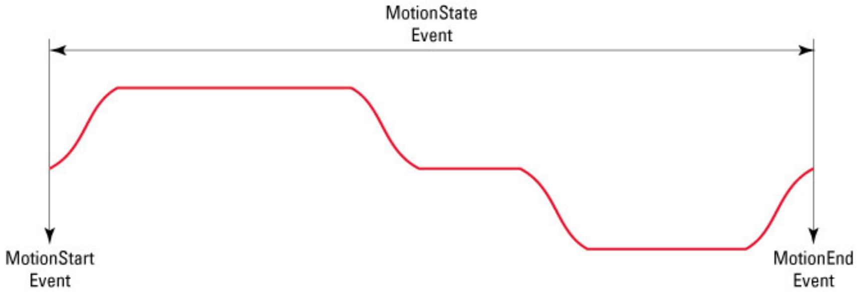


Figure 39: Motion event.

There are also several trajectory events that can be defined:

TrajectoryStart: Triggers an action when the trajectory has started. Event parameter 1 to 4 = 0 by default.

TrajectoryEnd: Triggers an action when the trajectory has stopped. Event parameter 1 to 4 = 0 by default.

TrajectoryState: Triggers an action during trajectory execution. Event parameter 1 to 4 = 0 by default.

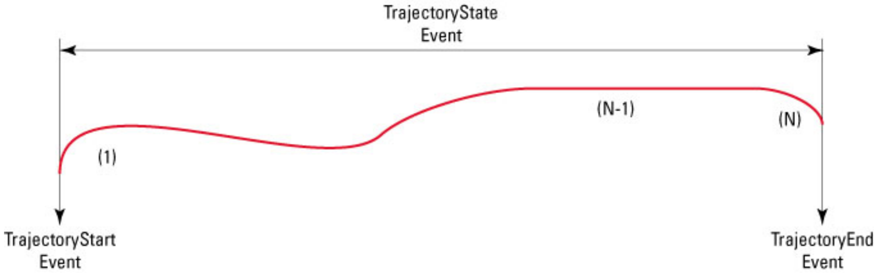


Figure 40: Trajectory event.

ElementNumberStart: Triggers an action when the trajectory element number has started. The first event parameter specifies the number of the trajectory element. The other event parameters are 0 by default.

ElementNumberState: Triggers an action during the execution of that trajectory element number. The first event parameter specifies the number of the trajectory element. The other event parameters are 0 by default.

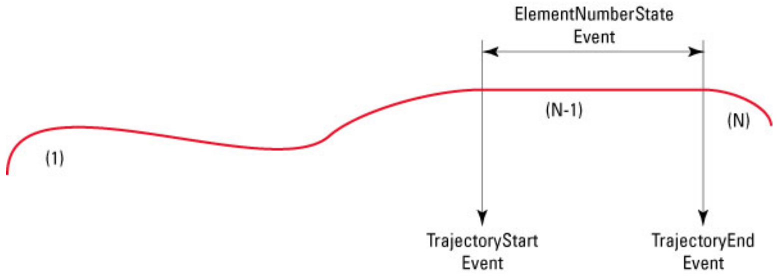


Figure 41: Element number event.

TrajectoryPulse: Triggers an action when a pulse on the trajectory is generated (see chapter 13.0: "Output Triggers" for details). All event parameters are 0 by default.

TrajectoryPulseState: Triggers an action during the trajectory pulse output state, meaning between the start and the end of the trajectory output pulses (see chapter 13.0: "Output Triggers" for details). All event parameters are 0 by default.

ILowState: Triggers an action when the digital input bit is in a low state. The first event parameter is the bit index (0 to 15). The other event parameters are 0 by default.

DILowHigh: Triggers an action when the digital input bit switches from a low state to a high state. The first event parameter is the bit index (0 to 15). The other event parameters are 0 by default.

DIHighState: Triggers an action when the digital input bit is in a high state. The first event parameter is the bit index (0 to 15). The other event parameters are 0 by default.

DIHighLow: Triggers an action when the digital input bit switches from a high to a low state. The first event parameter is the bit index (0 to 15). The other event parameters are 0 by default.

DIToggled: Triggers an action when the digital input bit switches from low to high or from high to low. The first event parameter is the bit index (0 to 15). The other event parameters are 0 by default.

ADCHighLimit: Triggers an action when the analog input value exceeds the limit. The first event parameter is the limit value in volts. The other event parameters are 0 by default.

ADCLowLimit: Triggers an action when the analog input value is below the limit. The first event parameter is the limit value in volts. The other event parameters are 0 by default.

ADCInWindow: Triggers an action when the analog input value is inside the window defined by min and max values. The first event parameter is the minimum value in volts and the second event parameter is the maximum value in volts. The other event parameters are 0 by default.

ADCOutWindow: Triggers an action when the analog input value is out of the window defined by min and max values. The first event parameter is the minimum value in volts and the second event parameter is the maximum value in volts. The other event parameters are 0 by default.

PositionerError: Triggers an action when the current positioner error applied with the error mask (for the 32 bit register) results in a value other than zero. The first event parameter specifies the error mask in a decimal format. The other event parameters are 0 by default.

NOTE

Refer to the Programmer's Manual for an error mask table.

Examples

EventExtendedConfigurationTriggerSet (MyGroup.MyPositioner.PositionerError, 2, 0, 0, 0)

This event happens when the positioner MyGroup.MyPositioner has a fatal following error.

EventExtendedConfigurationTriggerSet (MyGroup.MyPositioner.PositionerError, 12, 0, 0, 0)

This event happens when the positioner MyGroup.MyPositioner has either a home search time out or a motion done time out.

PositionerHardwareStatus: Triggers an action when the current hardware status applied with the error mask results in a value other than zero. The first event parameter specifies the status mask in decimal format. The other event parameters are 0 by default.

NOTE

For more information regarding PositionerHardwareStatus refer to the Programmer's Manual.

Example

EventExtendedConfigurationTriggerSet (MyGroup.MyPositioner.PositionerHardwareStatus, 768, 0, 0, 0)

This event happens when the positioner MyGroup.MyPositioner either the plus end of run or a minus end of run is detected.

WarningFollowingError: Triggers an action when the following error exceeds the warning following error value. In the PositionCompare mode (activated by the PositionerPositionCompareEnable() function), during a move (relative or absolute) and inside the zone set by PositionerPositionCompareSet(), if the current following error exceeds the WarningFollowingError value, the PositionCompareWarningFollowingErrorFlag is activated and the move returns a corresponding error (-120 : Warning following error during move with position compare enabled).

To reset the PositionCompareWarningFollowingErrorFlag, send the PositionerPositionCompareDisable() function.

The WarningFollowingError is set to FatalFollowingError (defined in stages.ini file) by default, but it can be modified with PositionerWarningErrorSet().

Example

EventExtendedConfigurationTriggerSet

(MyGroup.MyPositioner. WarningFollowingError, 0, 0, 0, 0)

This event happens when the positioner MyGroup.MyPositioner has a following error that exceeds the warning following error value.

DoubleGlobalArrayEqual: Triggers an action when the value of the variable in the DoubleGlobalArray and referenced by the global variable number is equal to the value to check. The variable can be modified by using the DoubleGlobalArraySet() function.

DoubleGlobalArrayDifferent: Triggers an action when the value of the variable in the DoubleGlobalArray and referenced by the global variable number is different from the value to check. The variable can be modified by using the DoubleGlobalArraySet() function.

DoubleGlobalArrayInferiorOrEqual: Triggers an action when the value of the variable in the DoubleGlobalArray and referenced by the global variable number is less than or equal to the value to check. The variable can be modified by using the DoubleGlobalArraySet() function.

DoubleGlobalArraySuperiorOrEqual: Triggers an action when the value of the variable in the DoubleGlobalArray and referenced by the global variable number is greater than or equal to the value to check. The variable can be modified by using the DoubleGlobalArraySet() function.

DoubleGlobalArrayInferior: Triggers an action when the value of the variable in the DoubleGlobalArray and referenced by the global variable number is lower than the value to check. The variable can be modified by using the DoubleGlobalArraySet() function.

DoubleGlobalArraySuperior: Triggers an action when the value of the variable in the DoubleGlobalArray and referenced by the global variable number is higher than the value to check. The variable can be modified by using the DoubleGlobalArraySet() function.

DoubleGlobalArrayInWindow: Triggers an action when the value of the variable in the DoubleGlobalArray and referenced by the global variable number is superior to MinValue and inferior to MaxValue.

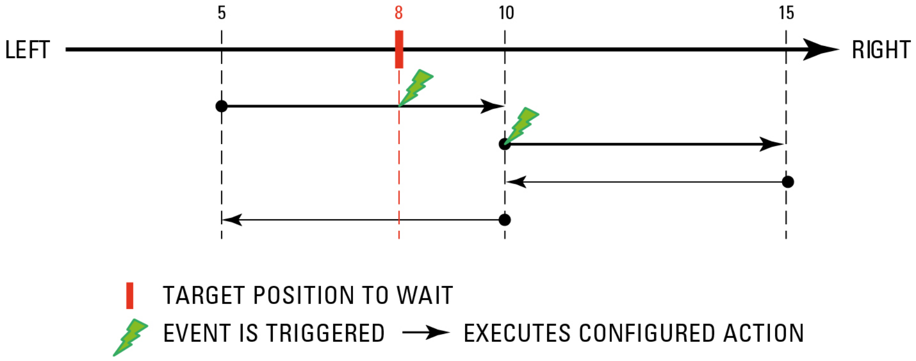
DoubleGlobalArrayOutWindow: Triggers an action when the value of the variable in the DoubleGlobalArray and referenced by the global variable number is outside the interval defined by MinValue and MaxValue.

WaitForPositionLeftToRight:

Triggers an action when the target position is detected during a displacement or if the target position has already passed, the action is triggered at the beginning of the displacement. The target position is checked only for all **positive displacements**. The first event parameter is the target position. The other event parameters are 0 by default.

NOTE

It is recommended to use this event trigger only after the home search is done. Once the event has occurred, it is deleted automatically. Reactivate the event trigger with the API function EventExtendedStart().

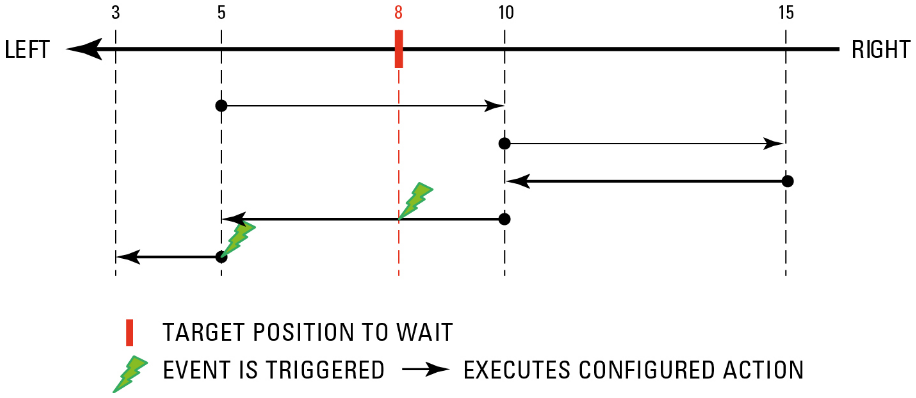


WaitForPositionRightToLeft:

Triggers an action when the target position is detected during a displacement or if the target position has already passed, the action is triggered at the beginning of the displacement. The target position is checked only for all **negative displacements**. The first event parameter is the target position. The other event parameters are 0 by default.

NOTE

It is recommended to use this event trigger only after the home search is done. Once the event has occurred, it is deleted automatically. Reactivate the event trigger with the API function EventExtendedStart().

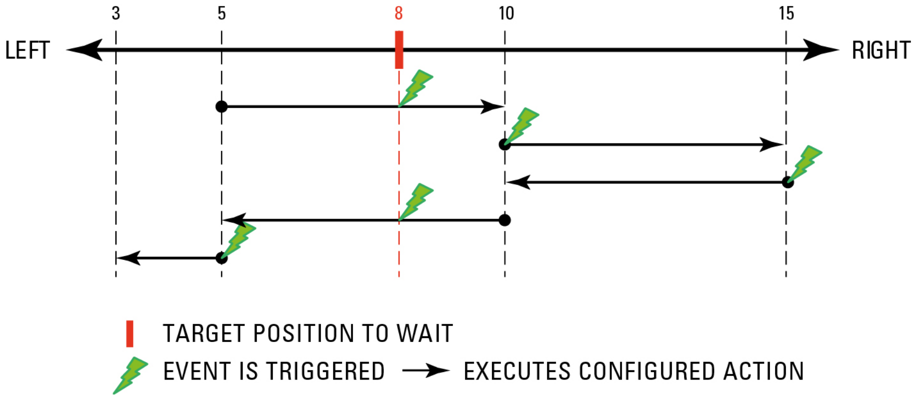


WaitForPositionToggled:

Triggers an action when the target position is detected during a displacement or if the target position has already passed, the action is triggered at the beginning of the displacement. The target position is checked for **all displacements**. The first event parameter is the target position. The other event parameters are 0 by default.

NOTE

It is recommended to use this event trigger only after the home search is done. Once the event has occurred, it is deleted automatically. Reactivate the event trigger with the API function EventExtendedStart().



Example

GroupInitialize(MyGroup)

GroupHomeSearch(MyGroup)

EventExtendedConfigurationTriggerSet(MyPositioner.**WaitForPositionToggled**, 8.0, 0, 0, 0)

EventExtendedConfigurationActionSet(GatheringRun, 10000, 1, 0, 0)

EventExtendedStart(int *ID)

The event will trigger the configured action, Gathering Run, when the target position is detected for MyPositioner or at the beginning of the displacement if the target position is already passed.

11.2 Actions

There are several actions that can be triggered by the events discussed previously. Users have the full flexibility to trigger any action (out of the list of possible actions) at any event (out of the list of possible events). It is also possible to trigger several actions at the same event by adding several sets of parameters to the function **EventExtendedConfigurationActionSet**(), similar to how it is done with events.

NOTE

The table is not an exhaustive list. Refer to the Programmer's Manual for more information.

[Actor.]				Action Name	Parameters			
Group	Positioner	GPIO	TimerX		1	2	3	4
		■		DACSet.CurrentPosition	Positioner name	Gain	Offset	0
		■		DACSet. CurrentVelocity	Positioner name	Gain	Offset	0
		■		DACSet. SetpointPosition	Positioner name	Gain	Offset	0
		■		DACSet. SetpointVelocity	Positioner name	Gain	Offset	0
		■		DACSet. SetpointAcceleration	Positioner name	Gain	Offset	0
		■		DACSet.Value	Value	0	0	0
		■		DOPulse	Mask	0	0	0
		■		DOToggle	Mask	0	0	0
		■		DOSet	Mask	Value	0	0
				EventRemove	Trigger identifier *(-1 for itself)*	0	0	0
				ExecuteCommand	Function name	Parameters *(Between {} and separator is the semi-column)*	Task name	
				ExecuteTCLScript	TCL file name	Task name	Arguments	
				ExternalGatheringRun	Nb of points	1	0	0
				GatheringOneData	0	0	0	0
				GatheringRun	Nb of points	Divisor	0	0
				GatheringRunAppend	0	0	0	0
				GatheringStop	0	0	0	0
				GlobalArrayDoubleSet	Global variable number	Double value	0	0
				GlobalArrayStringSet	Global variable number	String value	0	0
				KillTCLScript	Task name	0	0	0
■				MoveAbort	0	0	0	0
■				MoveAbortFast	Deceleration multiplier	0	0	0
				SynchronizeProfiler	0	0	0	0

CAUTION

Certain events like MotionState have a duration. These events trigger the associated action in each motion profiler cycle as long as the event is true. For example, associating the action DOToggle with the event MotionState will toggle the value of the digital output in each profiler cycle as long as the MotionState event is true.

An event doesn't reset the action after the event: For example, to set a digital output to a certain value during a constant velocity state and to set it to its previous value afterwards, two event triggers are needed: One to set to the digital output of the desired value at the event ConstantVelocityStart and another one to set it to its original value at the event ConstantVelocityEnd. The same effect CANNOT be achieved by using the event ConstantVelocityState by itself.

An action is composed entirely of:

[ACTOR].ACTION NAME, PARAMETER1, PARAMETER2, PARAMETER3, PARAMETER4.

Not all action names have a preceding actor, but all actions have four parameters. Even though all four parameters may not be defined in an action, it is still required to have an entry, with zero (0) as the default.

To define an action, use the function **EventExtendedConfigurationActionSet**().

Example:

EVENTEXTENDEDCONFIGURATIONACTIONSET (GPIO1.DO.DOTOGGLED, 4, 0, 0, 0)

In this case the actor is the digital output GPIO1.DO and the action is to toggle the output. The value 4 refers to bit #3, 00000100. Hence, this action toggles the value of bit 3 on the digital output GPIO.DO.

EVENTEXTENDEDCONFIGURATIONACTIONSET (EXECUTETCLSCRIPT, EXAMPLE.TCL, 1, 0, 0)

The action ExecuteTCLScript has no preceding actor. This action will execute the TCL script "Example.tcl". The task name is 1 and the TCL script has no arguments (a zero for the third parameter means there are no arguments).

EVENTEXTENDEDCONFIGURATIONACTIONSET (GATHERINGRUN, 1000, 8, 0, 0)

The action GatheringRun has no preceding actor. This action will start an internal data gathering. It will gather a total of 1000 data points, one data point every 8th servo cycle, meaning one data point for every 8/8000 s = 1 ms (for the default servo rate 8 kHz).

It is also possible to trigger several actions with the same event. To do so, just define another action in the SAME function. Several actions must be separated by a comma (,).

Example:

EventExtendedConfigurationTriggerSet (MyGroup.MyPositioner.PositionerError, 2, 0, 0, 0)

EventExtendedConfigurationActionSet (ExecuteTCLScript, ShutDown.tcl, 1, 0, 0, ExecuteTCLScript, ErrorDiagnostic.tcl, 2, 0, 0)

EventExtendedStart()

In this example, the TCL scripts ShutDown.tcl and ErrorDiagnostic.tcl are executed when a fatal following error is detected on the positioner MyGroup.MyPositioner.

The exact meaning of the different actions and action parameters is as follows:

DOToggle: This action is used to reverse the value of one or many bits of the Digital Output. When using this action with an event that has some duration (for example motion state) the value of the bits will be toggled at each profiler cycle as long as the event occurs.

Action Parameter #1 – Mask
The mask defines which bits on the GPIO output will be toggled (change their value). For example, if the GPIO output is an 8 bit output and the mask is set to 4 then the equivalent binary number is 00000100. So as an action, bit #3 will be toggled.

Action Parameter #2 to #4
These parameters are 0 by default.

DOPulse: This action is used to generate a positive pulse on the Digital Output. The duration of the pulse is 1 microsecond. To function, the bits on which the pulse is generated should be set to zero before. When using this action with an event that has some duration (for example motion state), a 1 µs pulse will be generated at each cycle of the motion profiler (or every 500 µs for default value) as long as the event occurs.

Action Parameter #1 – Mask
The mask defines on which bits on the GPIO output the pulse will be generated. For example, if the GPIO output is an 8 bit output and the mask is set to 6 then the equivalent binary number is 00000110. So as an action, a 1 µs pulse will be generated on bit #2 and #3 of the GPIO output.

Action Parameter #2 to #4
These parameters are 0 by default.

DOSet: This action is used to modify the value of bit(s) on a Digital Output.

Action Parameter #1 – Mask
The mask defines which bits on the GPIO output are being addressed. For example, if the GPIO output is an 8 bit output and the mask is set to 26 then the equivalent binary number is 00011010. Therefore with a Mask setting of 26, only bits # 2, #4 and #5 are being addressed on the GPIO output.

Action Parameter #2 – Value
This parameter sets the value of the bits that are being addressed according to the Mask setting. For example since a Mask setting of 26, bits #2, #4 and #5 can be modified, a value of 8 (00001000) will set the bits #2 and #5 to 0 and the bit #4 to 1.

Action parameter #3 and #4
These parameters are 0 by default.

DACSet.CurrentPosition and **DACSet.SetpointPosition:** This action sets a voltage on the Analog output in relation to the actual (current) or theoretical (Setpoint) position. The gain and offset are used to calibrate the output. This action makes the most sense with events that have some duration (always, MotionState, ElementNumberState, etc.) as the analog output will be updated at each servo cycle or at each profiler cycle as long as the event occurs. When used with events that have no duration (like MotionStart or MotionEnd), the analog output is only updated once and this value is kept until it is changed.

Action Parameter #1 – Positioner Name
This parameter defines the name of the positioner on which the position value is used.

Action Parameter #2 – Gain
The position value is multiplied by the gain value. For example, if the gain is set to 10 and the position value is 1 mm (or any other unit), then the output voltage is 10 V.

Action Parameter #3 – Offset	The offset value is used to correct for any voltage that may already be present in the Analog output.

Analog output = Position value * gain + offset

Action parameter #4	This parameter is 0 by default.

DACSet.CurrentVelocity and **DACSet.SetpointVelocity:** This action sets a voltage on the Analog output relative to the actual (current) or theoretical (Setpoint) velocity. The gain and the offset are used to calibrate the output. This action makes most sense with events that have duration (Always, MotionState, ElementNumberState, etc.) as the analog output is updated at each servo cycle or at each profiler cycle as long as the event occurs. When used with events that have no duration (like MotionStart or MotionEnd), the analog output is only updated once and this value is kept until it is changed.

Action Parameter #1 – Positioner Name	This parameter defines the name of the positioner in which the Velocity value is used.
Action Parameter #2 – Gain	The Velocity value is multiplied by the gain value. For example if the gain is set to 10 and the velocity value is 1 mm/s (or any other velocity unit), then the output voltage is 10 V.
Action Parameter #3 – Offset	The offset value is used to correct for any voltage that may initially be present in the Analog output.

Analog output = Velocity value * gain + offset

Action parameter #4	This parameter must be 0 by default.

DACSet.SetpointAcceleration: This action is used to output a voltage on the Analog output to form an image of the theoretical acceleration. The gain and the offset are used to calibrate this image. This action makes most sense with events that have duration (Always, MotionState, ElementNumberState, etc.) as the analog output will be updated at each servo cycle or at each profiler cycle as long as the event lasts. When used with events that have no duration (like MotionStart or MotionEnd), the analog output is only updated once and keep this value until it is changed.

Action Parameter #1 – Positioner Name	This parameter defines the name of the positioner in which the SetpointAcceleration is used to output in the analog output.
Action Parameter #2 – Gain	The SetpointAcceleration is multiplied by the gain value. For example, if the gain is set to 10 and the corrected SetpointAcceleration is 1 mm/s^2 then the output voltage will be 10 V.
Action Parameter #3 – Offset	The offset value is used to correct for any voltage that may initially be present in the Analog output.

Analog output = SetpointAcceleration value * gain + offset

Action parameter #4	This parameter is 0 by default.

NOTE

The gain can be any constant value used to scale the output voltage and the offset value can be any constant value used to correct for any offset voltage in the analog output.

ExecuteTCLScript: This action executes a TCL script on an event.

Action Parameter #1 – TCL File Name This parameter defines the file name of the TCL program.

Action Parameter #2 – TCL Task Name Since several TCL scripts can run simultaneously different or even the same, the TCL Task Name is used to track individual TCL programs. For example, the TCL Task Name stops a particular program without stopping all other TCL programs that are running simultaneously.

Action Parameter #3 – TCL Argument List The Argument list is used to run the TCL scripts with input parameters. For the argument parameter, any input can be used (number, string). These parameters are used inside the script. To get the number of arguments, use $tcl_argc" inside the script. To get each argument, use "$tcl_argc($i)" inside the script. For example, this parameter can be used to specify a number of loops inside the TCL script. A zero (0) for this parameter means there are no input arguments.

Action parameter #4 This parameter is 0 by default.

KillTCLScript: This action stops a TCL script on an event.

Action parameter #1 – Task name This parameter defines which TCL script is stopped. Since several TCL scripts can run simultaneously, different or even the same script, the TCL Task Name is used to track individual TCL programs.

Action parameter #2 to #4 These parameters are 0 by default.

GatheringOneData: This action acquires one data as defined by the function GatheringConfigurationSet. Different from the GatheringRun (see next action), which generates a new gathering file, the GatheringOneData appends the data to the current gathering file stored in memory. In order to store the data in a new file, first launch the function GatheringReset, which deletes the current gathering file from memory.

Action parameter #1 to #4 These parameters are 0 by default.

GatheringRun: This action starts an internal data gathering. It requires that an internal gathering was previously configured with the function GatheringConfigurationSet. The gathering must be launched by a punctual event and does not work with events that have duration.

Action Parameter #1 – NbPoints This parameter defines the number of data acquisitions. NbPoints multiplied by the number of gathered data types must be smaller than 1,000,000. For instance, if 4 types of data are collected, NbPoints can not be larger than 250,000 (4*250,000 = 1,000,000).

Action Parameter #2 This parameter is 1 by default.

Action Parameter #3 and #4 These parameters are 0 by default.

GatheringRunAppend: This action continues a gathering previously stopped with the action GatheringStop, see next action.

Action parameter #1 to #4 These parameters are 0 by default.

GatheringStop: This action halts a data gathering previously launched by the action GatheringStart. Use the action GatheringRunAppend to continue data gathering. Note that the action GatheringStop does not automatically store the gathered data from the buffer to the flash disk of the controller. To store data, use the function GatheringStopAndSave. For more details about data gathering, refer to chapter 12.0: “Data Gathering”.

Action parameter #1 to #4	These parameters are 0 by default.

ExternalGatheringRun: This action starts an external data gathering. It requires that an external data gathering was previously configured with the function GatheringExternalConfigurationSet. The gathering must be launched by a punctual event and does not work with events that have duration.

Action Parameter #1 – NbPoints	This parameter defines the number of data acquisitions. NbPoints multiplied by the number of gathered data types must be smaller than 1,000,000. For instance, if 4 types of data are collected, NbPoints can not be larger than 250,000 (4*250,000 = 1,000,000).
Action Parameter #2 – Divisor	This parameter defines every Nth number of the trigger input signal at which the gathering will take place. This parameter must be an integer and greater than or equal to 1. For example if the divisor is set to 5 then gathering will take place every 5th trigger on the trigger input signal.
Action Parameter #3 and #4	These parameters are 0 by default.

For further details on data gathering, see chapter 12.0: “Data Gathering”.

MoveAbort: This action stops (abort) a motion on an event. It is similar to sending a MoveAbort() function on the event. After stopping, the group is in the READY state.

Action Parameter #1 to #4	These parameters are 0 by default.

11.3 Functions

The following functions are related to event triggers:

- **EventExtendedConfigurationTriggerSet**(): This function configures one or several events. In the case of several events, the different events are separated by a comma (,) in the argument list. Before activating an event, one or several actions must be configured with the function EventExtendedConfigurationActionSet(). Only then, the event and the associated action(s) can be activated with the function EventExtendedStart().
- **EventExtendedConfigurationTriggerGet**(): This function returns the event configuration defined by the last EventExtendedConfigurationTriggerSet() function.
- **EventExtendedConfigurationActionSet**(): This function associates an action to the event defined by the last EventExtendedConfigurationTriggerSet() function.
- **EventExtendedConfigurationActionGet**(): This function returns the action configuration defined by the last EventExtendedConfigurationActionSet() function.
- **EventExtendedStart**(): This function launches (activates) the last configured event and the associated action(s) defined by the last EventExtendedConfigurationTriggerSet() and EventExtendedConfigurationActionSet() and returns an event identifier. When activated, the XPS controller checks for the event at each servo cycle (or at each profiler cycle for those events that are motion related) and triggers the action when

the event occurs. Hence, the maximum latency between the event and the action is equal to the servo cycle time (default value 125 µs) or equal to the profiler cycle time (default value 500 µs) for motion related events. For events with duration, it also means that the same action is triggered at each servo cycle, meaning every 125 µs, or at each profiler cycle, which is every 500 µs as long as the event occurs.

Event triggers (and their associated action) are automatically removed after the event configuration has happened at least once and is no longer true. The only exception is if the event configuration contains any of the permanent events "Always" or "Trigger". In this case the event trigger will always stay active. With the function EventExtendedRemove(), any event trigger can get removed.

- **EventExtendedWait**(): This function halts a process (essentially by blocking the socket) until the event defined by the last EventExtendedConfigurationTriggerSet() occurs.
- **EventExtendedRemove**(): This function removes the event trigger associated with the defined event identifier.
- **EventExtendedGet**(): This function returns the event configuration and the action configuration associated with the defined event identifier.
- **EventExtendedAllGet**(): This function returns, for all active event triggers, the event identifier, the event configuration and the action configuration. The details of the different event triggers are separated by a comma (,).

11.4 Examples

Below is a table that shows possible events that can be associated with possible actions. Some of these examples however, may have unwanted results. Since the XPS controller provides great flexibility to trigger almost any action at any event, the user must be aware of the possible unwanted effects.

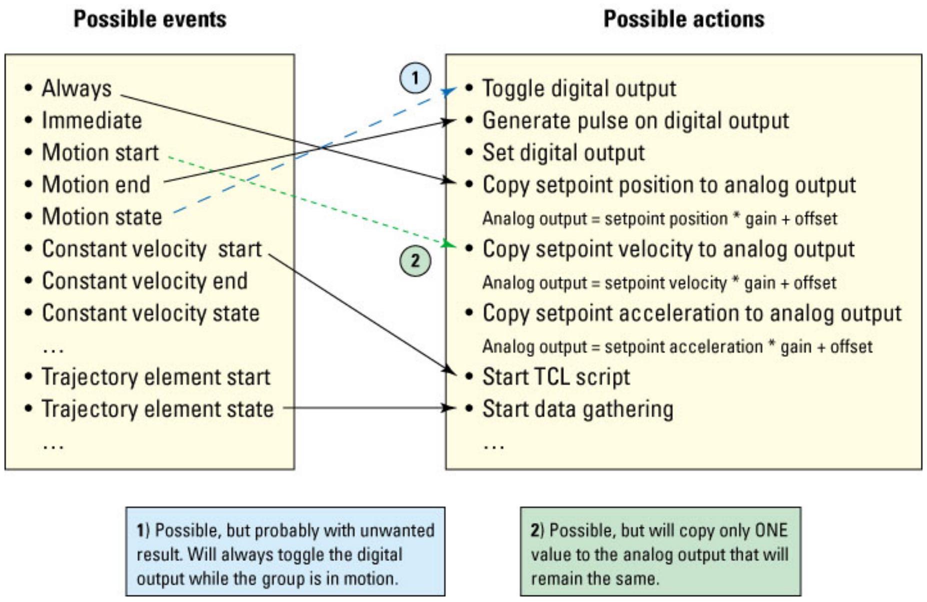

Possible events
• Always
• Immediate
• Motion start
• Motion end
• Motion state
• Constant velocity start
• Constant velocity end
• Constant velocity state
…
• Trajectory element start
• Trajectory element state
…
Possible actions
• Toggle digital output
• Generate pulse on digital output
• Set digital output
• Copy setpoint position to analog output
Analog output = setpoint position * gain + offset
• Copy setpoint velocity to analog output
Analog output = setpoint velocity * gain + offset
• Copy setpoint acceleration to analog output
Analog output = setpoint acceleration * gain + offset
• Start TCL script
• Start data gathering
…
1
2
1) Possible, but probably with unwanted result. Will always toggle the digital output while the group is in motion.
2) Possible, but will copy only ONE value to the analog output that will remain the same.

Figure 42: Possible events.

Examples

1. **EventExtendedConfigurationTriggerSet (G1.P1.SGamma.ConstantVelocityStart, 0, 0, 0, 0)**

   EventExtendedConfigurationActionSet (GPIO1.DO.DOSet, 4, 4, 0, 0)

   EventExtendedStart()

   GroupMoveAbsolute (G1.P1, 50)

   In this example, when positioner G1.P1 reaches constant velocity, bit #3 on the digital output on connector number 1 is set to 1 (Note: 4 = 00000100). Note, that the state of the bit will not change when the constant velocity of the positioner has ended. In order to do so, a second event trigger would be required (see next example).

2. **EventExtendedConfigurationTriggerSet (G1.P1.SGamma.ConstantVelocityStart, 0, 0, 0, 0)**

   EventExtendedConfigurationActionSet (GPIO1.DO.DOSet, 4, 4, 0, 0)

   EventExtendedStart()

   EventExtendedConfigurationTriggerSet (G1.P1.SGamma.ConstantVelocityEnd, 0, 0, 0, 0)

   EventExtendedConfigurationActionSet (GPIO1.DO.DOSet, 4, 0, 0, 0)

   EventExtendedStart()

   GroupMoveAbsolute (G1.P1, 50)

   In this example, when positioner G1.P1 reaches constant velocity, bit #3 on the digital output on connector number 1 is set to 1 (Note: 4 = 00000100) and when the constant velocity of the positioner G1.P1 is over, bit #3 will be set to zero. Note, that the same effect can not be reached with the event name ConstantVelocityState. After both events have happened, the event triggers will get automatically removed. In order to trigger the same action at each motion, it is required to link the events with the event "Always" (see next example). This link will avoid that the event trigger gets removed after it is not happening anymore.

3. **EventExtendedConfigurationTriggerSet (Always, 0, 0, 0, 0, G1.P1.SGamma.ConstantVelocityStart, 0, 0, 0, 0)**

   EventExtendedConfigurationActionSet (GPIO1.DO.DOSet, 4, 4, 0, 0)

   EventExtendedStart()

   EventExtendedConfigurationTriggerSet (Always, 0, 0, 0, 0, G1.P1.SGamma.ConstantVelocityEnd, 0, 0, 0, 0)

   EventExtendedConfigurationActionSet (GPIO1.DO.DOSet, 4, 0, 0, 0)

   EventExtendedStart()

   GroupMoveAbsolute (G1.P1, 50)

   GroupMoveAbsolute (G1.P1, -50)

   In this example, when positioner G1.P1 reaches constant velocity, bit #3 on the digital output on connector number 1 is set to 1 (Note: 4 = 00000100) and when the constant velocity of the positioner G1.P1 is over, bit #3 will be set to zero. Different from the previous example, adding the event "Always" avoids the event trigger being removed after the event is over. Hence, the state of the bit #3 will change with every beginning and with every end of the constant velocity state of a motion.

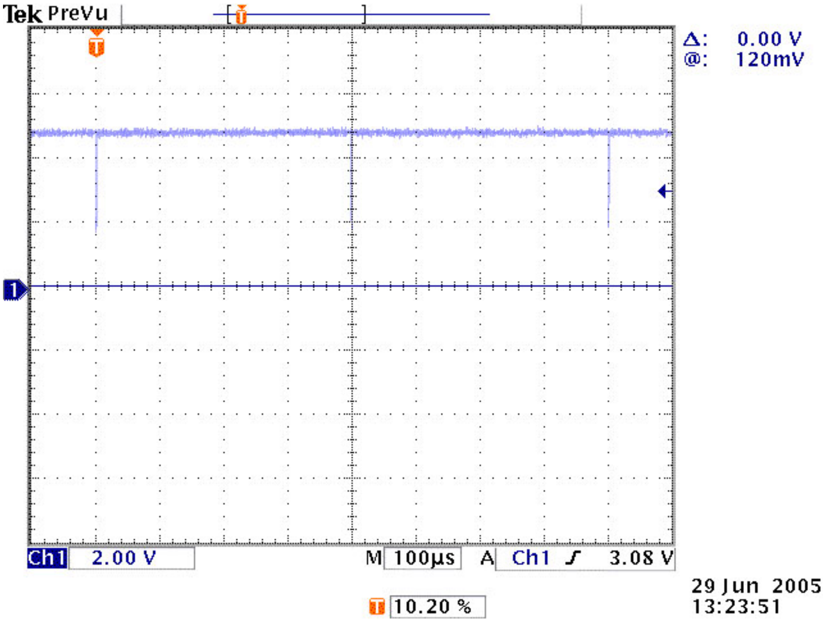
4. **EventExtendedConfigurationTriggerSet (G1.P1.SGamma.ConstantVelocityState, 0, 0, 0, 0)**

   EventExtendedConfigurationActionSet (GPIO1.DO.DOSet, 255, 0, 0, 0)

   EventExtendedStart()

   GroupMoveAbsolute (G1.P1, 50)

   In this example, during the constant velocity state of the positioner G1.P1, 1 µs pulses are generated on all 8 bits in the digital output on connector number 1, at every cycle of the motion profiler (Note: 255 = 11111111). The cycle time of the motion profiler is 500 µs for default value, so pulses are generated every 500 µs (see picture below).



5. **EventExtendedConfigurationTriggerSet (Always, 0, 0, 0, 0)**

   EventExtendedConfigurationActionSet (GPIO2.DAC1.DACSet.SetpointPosition, G1.P1, 0.1, -10, 0, GPIO2.DAC2.DACSet.SetpointVelocity, G1.P1, 0.5, 0, 0)

   EventExtendedStart()

   In this example, the analog output #1 on GPIO2 will always output a voltage in relation to the SetpointPosition of the positioner G1.P1, and the output #2 on GPIO2 will always output a voltage in relation to the SetpointVelocity of the same positioner. The gain on output #1 is set to 0.1 V/unit and the offset to -10 V. This means that when the stage is at the position 0 units, a voltage of -10 V will be sent. When the stage is at the position 10 units, a voltage of -9 V will be sent.
   Here, the event “Always” means that these values will be updated every servo cycle, means every 0.125 ms for the default servo cycle rate. If instead of the event “Always”, the event “Immediate” will be used, only the most recent values will be sent and kept. If instead of the event “Always”, a motion related event such as MotionState is used, the update will only happen at every profiler cycle, or every 0.5 ms for the default profile generator rate.

6. **TimerSet(Timer1,8000)**

   EventExtendedConfigurationTriggerSet (Timer1.Timer, 0, 0, 0, 0)

   EventExtendedConfigurationActionSet (GPIO1.DO.DOToggle, 255, 0, 0, 0)

   EventExtendedStart()

   EventExtendedRemove(1)

   The function Timer() sets the Timer1 at every 8,000th servo cycle (for the default servo cycle 8 kHz), or at one second. Hence, in this example, every second all bits in the digital output on connector number 1 will be toggled (Note: 255 = 11111111). The event Timer is permanent. In order to remove the event trigger, use the function EventExtendedRemove() with the associated event identifier (1 in this case).

7. **MultipleAxesPVTPulseOutputSet(G1,2,20,1)**

   GatheringConfigurationSet(G1.P1.CurrentPosition)

   EventExtendedConfigurationTriggerSet(Always, 0,0,0,0,G1.PVT.TrajectoryPulse,0,0,0,0)

   EventExtendedConfigurationActionSet(GatheringOneData,0,0,0,0)

   EventExtendedStart()

   MultipleAxesPVTExecution(G1,Traj.trj,1)

   In this example, the generation of an output pulse every one second between the 2nd and the 20th element in the next PVT trajectory executed on the group G1 is first defined (function MultipleAxisPVTPulseOutputSet). Then, data gathering is defined (CurrentPosition of positioner G1.P1).

   Hence, in this example, with every trajectory pulse, one data point is gathered and appended to the current gathering file in memory. Here, adding the event TrajectoryPulse with the permanent event Always makes sure that the event trigger is always active. Without the event Always, only one data point will be gathered.

   This is because any event is automatically removed once it happens and does not happening in the next servo or profiler cycle (which is the case here as a pulse is only generated every one second).

   Please note that the action GatheringOneData appends data to the current data file. In order to store the data in a new file it is required to first launch the function GatheringReset() which deletes the current data file from memory.

8. **GatheringConfigurationSet(G1.P1.CurrentPosition)**

   EventExtendedConfigurationTriggerSet (G1.P1.SGamma.MotionStart,0,0,0,0)

   EventExtendedConfigurationActionSet(GatheringRun,20,1000,0,0)

   EventExtendedStart()

   GroupMoveAbsolute (G1.P1, 50)

   GatheringStopAndSave()

   In this example, an internal data gathering of 20 data points every 0.1 second (every 800th servo cycle) is launched with the start of the next motion of the positioner G1.P1. The type of data that gathered is defined with the function GatheringConfigurationSet (CurrentPosition of positioner G1.P1). To store the data from internal memory to the flash disk in the XPS controller, send the function GatheringStopAndSave(). The GatheringRun deletes the current data file in internal memory (in contrast to the GatheringOneData which appends data to the current file). Also, the function GatheringStopAndSave() stores the data file under a default name Gathering.dat on the flash disk of the XPS controller and will overwrite any older file of the same name in the same folder. Hence, make sure to store valuable data files under a different name before a GatheringStopAndSave().

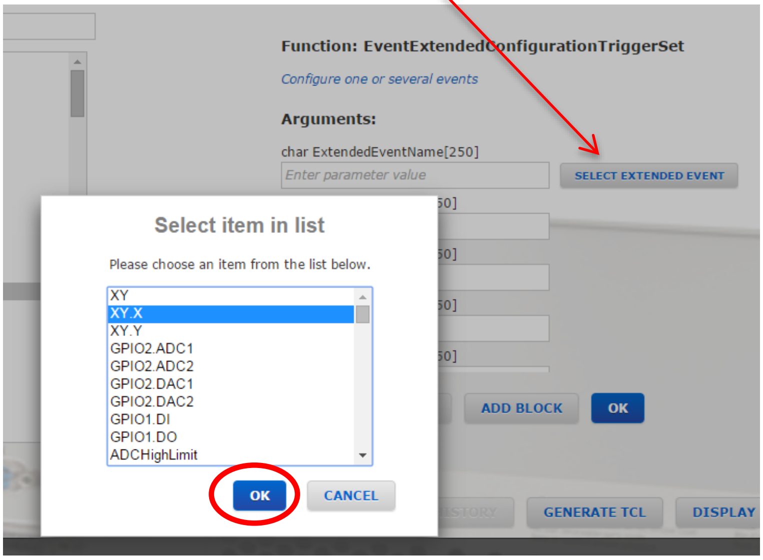
NOTE

When using the function EventExtendedConfigurationTriggerSet() or EventExtendedConfigurationActionSet() from the terminal screen of the XPS utility, the syntax for one parameter is not directly accessible. For instance, for the event XY.X.SGamma.MotionStart, first select XY.X from the choice list. Then, click on the choice field again and select SGammaMotionStart. See also screen shots below.

For specifying more than one data type, use the "ADD BLOCK" button. Select the next parameter as described above.

Step 1:

Click **"SELECT EXTENDED EVENT"** then select the positioner name and click **"OK"**.



Function: EventExtendedConfigurationTriggerSet

Configure one or several events

Arguments:

char ExtendedEventName[250]

Enter parameter value

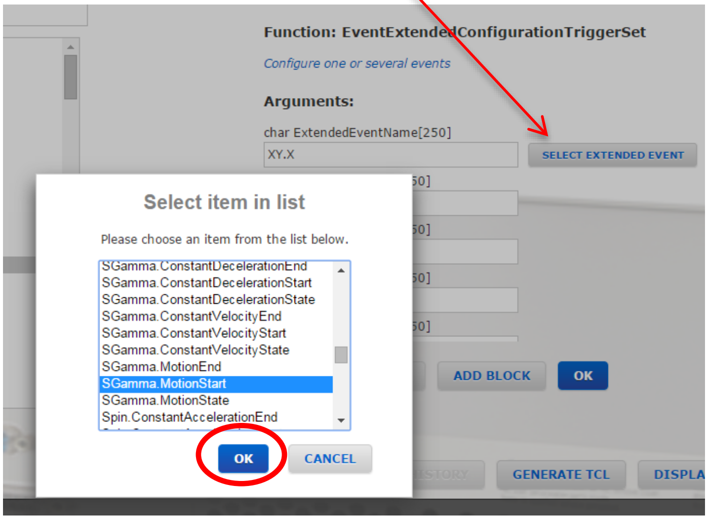
Select item in list

Please choose an item from the list below.

XY
XY.X
XY.Y
GPIO2.ADC1
GPIO2.ADC2
GPIO2.DAC1
GPIO2.DAC2
GPIO1.DI
GPIO1.DO
ADCHighLimit

Step 2:

Click **"SELECT EXTENDED EVENT"** again then select the parameter name and click **"OK"**.



Function: EventExtendedConfigurationTriggerSet

Configure one or several events

Arguments:

char ExtendedEventName[250]

XY.X

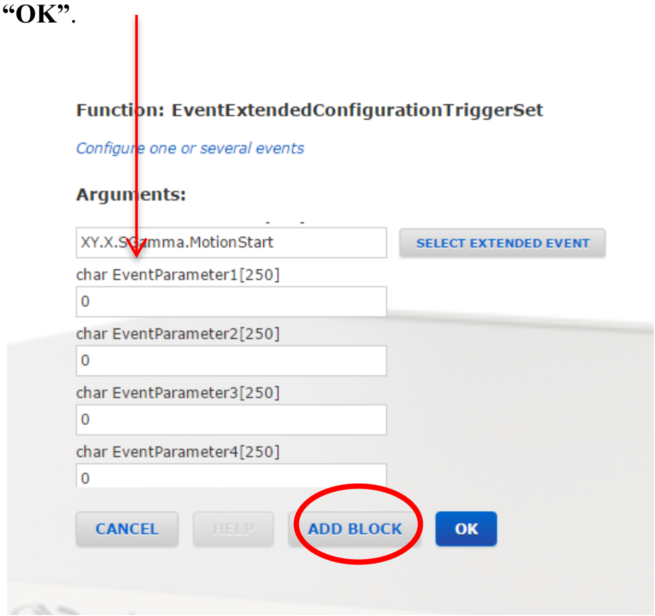
Select item in list

Please choose an item from the list below.

SGamma.ConstantDecelerationEnd
SGamma.ConstantDecelerationStart
SGamma.ConstantDecelerationState
SGamma.ConstantVelocityEnd
SGamma.ConstantVelocityStart
SGamma.ConstantVelocityState
SGamma.MotionEnd
SGamma.MotionStart
SGamma.MotionState
Spin.ConstantAccelerationEnd

Step 3:

Define event parameters. To add another event **"ADD BLOCK".** Repeat Step 1 and Step 2, or else click **"OK"**.



Function: EventExtendedConfigurationTriggerSet

Configure one or several events

Arguments:

XY.X.SGamma.MotionStart

char EventParameter1[250]: 0

char EventParameter2[250]: 0

char EventParameter3[250]: 0

char EventParameter4[250]: 0

12.0 Data Gathering

The XPS controller provides four methods for data gathering:

1. Time-based (internal) data gathering. With this method one data set is gathered for every nth servo cycle.
2. Event-based (internal) data gathering. With this method one data set is gathered at an event.
3. Function-based (internal) data gathering. With this method one data set is gathered by a function.
4. Trigger-based (external) data gathering. With this method one data set is gathered for every nth external trigger input (see also chapter 13.0: “Output Triggers“).

Method 1, 2, and 3, these are also referred to as internal or servo cycle synchronous data gathering. With the trigger-based data gathering, this is also referred to as an external data gathering, as the event that triggers the data gathering or the receipt of a trigger input, is asynchronous to the servo cycle.

The time-based, the event-based and the function-based data gathering store the data in a common file called gathering.dat. The trigger-based (external) data gathering stores the data in a different file, called ExternalGathering.dat. The type of data that can be gathered differs also between the internal and the external data gathering.

Before starting any data gathering, the type of data to be gathered needs to be defined using the functions GatheringConfigurationSet() (in case of an internal data gathering) or ExternalGatheringConfigurationSet() (in case of an external data gathering). Refer to the Programmer's Manual and the Gathering functions for a complete list of data types.

During data gathering, new data is appended to a buffer. With the functions GatheringCurrentNumberGet() and GatheringExternalCurrentNumberGet(), the current number of data sets in this buffer and the maximum possible number of data sets that fits into this buffer can be recalled. The maximum possible number of data sets equals 1,000,000 divided by the number of data types belonging to one data set.

The function GatheringDataGet(index) returns one set of data from the buffer. Here, the index 0 refers to the 1st data set, the index (n-1) to the n-th data set. When using this function from the Terminal screen of the XPS utility, the different data types belonging to one data line are separated by a semicolon (;).

To save the data from the buffer to the flash disk of the XPS controller, use the functions GatheringStopAndSave() and GatheringExternalStopAndSave(). These functions will store the gathering files in the XPS controller under the name Gathering.dat (with function GatheringStopAndSave() for internal gathering) or GatheringExternal.dat (with function GatheringExternalStopAndSave() for external gathering). To rename the gathering file use the API function **FileGatheringRename**().

CAUTION

The functions GatheringStopAndSave() and GatheringExternalStopAndSave() overwrite any older files with the same name. After a data gathering, it is required to rename (use the API function FileGatheringRename() to rename the gathering file) or better, to download to a PC using the XPS webpage Files -> Gathering files.

A gathering file can have a maximum of 1,000,000 data entries and a maximum of 25 different data types. The first line of the data file contains the sample period in seconds (minimum period = CorrectorISRPeriod), the second line contains the names of the data type(s) and the other lines contain the acquired data. A sample file is shown below.

Gathering.dat

```
SamplePeriod           0                  0
GatheringTypeA     GatheringTypeB     GatheringTypeC
ValueA1               ValueB1            ValueC1
ValueA2               ValueB2            ValueC2
…                      …                  …
ValueAN               ValueBN            ValueCN
```

NOTE

Refer to Programmer's manual to get a list of gathering data types and a list of external gathering data types.

12.1 Time-Based (Internal) Data Gathering

The data for time-based gathering are latched by an internal interrupt related to the servo cycle of the XPS (example 8 kHz). The function GatheringConfigurationSet() defines the type of data that will be stored in the data file. The following is a list of all the data type(s) that can be collected:

PositionerName.CurrentPosition

PositionerName.SetpointPosition

PositionerName.FollowingError

PositionerName.CurrentVelocity

PositionerName.SetpointVelocity

PositionerName.CurrentAcceleration

PositionerName.SetpointAcceleration

PositionerName.CorrectorOutput

GPIO (ADC, DAC, DI, DO) See the Programmer's Manual for a list of all the GPIO Names of the Analog and Digital I/O.

The Setpoint values refer to the theoretical values from the profiler whereas the current values refer to the actual or real values of position, velocity and acceleration.

It is possible to start the gathering either by a function call or at an event. The following sequence of functions is used for a time-based data gathering started by a function call:

GatheringConfigurationSet()

GatheringRun()

The following sequence of functions is used to start a time-based data gathering at an event:

GatheringConfigurationSet()

EventExtendedConfigurationTriggerSet()

EventExtendedConfigurationActionSet()

EventExtendedStart()

A function triggers the action, for instance, a GroupMoveRelative().

When all data is gathered, use the function **GatheringStopAndSave**() to save the data from the buffer to the flash disk of the XPS controller. To rename the gathering file use the API function **FileGatheringRename**().

Other functions associated with internal Gathering are:

GatheringConfigurationGet()

GatheringCurrentNumberGet()

GatheringDataGet()

GatheringDataMultipleLinesGet()

GatheringStop()

GatheringRunAppend()

See the Programmer's Manual for details about these functions.

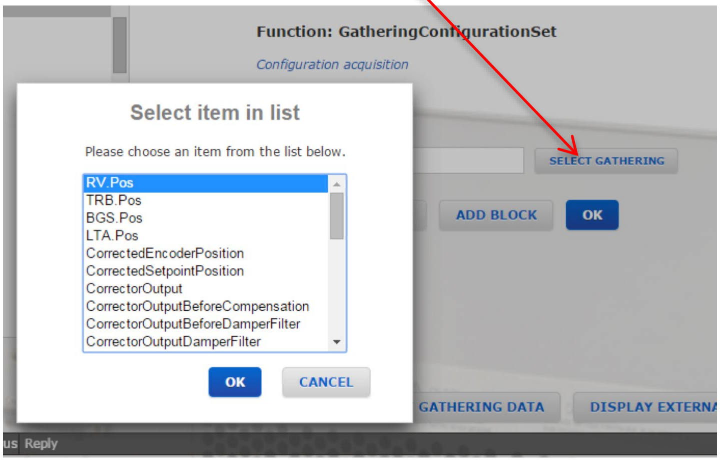
NOTE

When using the function GatheringConfigurationSet() from the terminal screen of the XPS utility, the syntax for one parameter is not directly accessible. For instance, for the parameter XY.X.SetpointPosition, first select XY.X from the choice list. Then, click on the choice field again and select SetpointPosition. See also screen shots on the next page.

For specifying more than one data type, use the ADD button. Select the next parameter as described below.

Step 1:

Click **"SELECT GATHERING"** then select the positioner name and click **"OK"**.



Function: GatheringConfigurationSet

Configuration acquisition

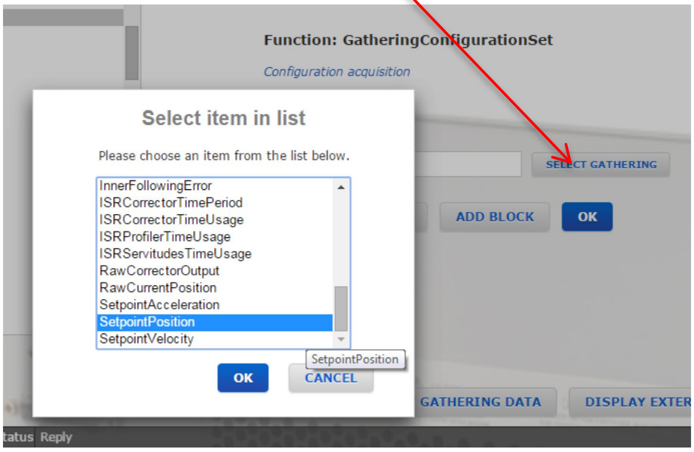
Select item in list

Please choose an item from the list below.

RV.Pos
TRB.Pos
BGS.Pos
LTA.Pos
CorrectedEncoderPosition
CorrectedSetpointPosition
CorrectorOutput
CorrectorOutputBeforeCompensation
CorrectorOutputBeforeDamperFilter
CorrectorOutputDamperFilter

Step 2:

Click **"SELECT GATHERING"** again then select the parameter name and click **"OK"**.



Function: GatheringConfigurationSet

Configuration acquisition

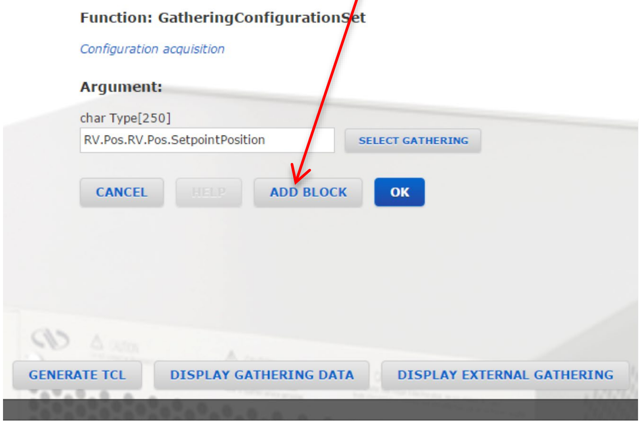
Select item in list

Please choose an item from the list below.

InnerFollowingError
ISRCorrectorTimePeriod
ISRCorrectorTimeUsage
ISRProfilerTimeUsage
ISRServitudesTimeUsage
RawCorrectorOutput
RawCurrentPosition
SetpointAcceleration
SetpointPosition
SetpointVelocity

Step 3:

To add another parameter, click **"ADD BLOCK".**
Repeat Step 1 and Step 2.



Function: GatheringConfigurationSet

Configuration acquisition

Argument:

char Type[250]: RV.Pos.RV.Pos.SetpointPosition

Example 1

Using the terminal screen of the XPS utility, this example shows the sequence of functions to accomplish a time-based data gathering triggered at an event.

GroupInitialize(XY)

GroupHomeSearch(XY)

GatheringConfigurationSet(XY.X.SetpointPosition, XY.X.CurrentVelocity, XY.X.SetpointAcceleration)

The 3 data XY.X.SetpointPosition, XY.X.CurrentVelocity and XY.X.SetpointAcceleration will be gathered.

EventExtendedConfigurationTriggerSet (XY.X.SGamma.MotionStart,0,0,0,0)

EventExtendedConfigurationActionSet(GatheringRun,4000,8,0,0)

EventExtendedStart()

GroupMoveRelative(XY.X, 50)

GatheringStopAndSave()

In this example, gathering is started when the positioner XY.X starts its next motion using the Sgamma profiler, in this case with GroupMoveRelative() or possibly with GroupMoveAbsolute(). The types of data being collected are the Setpoint Position, Current Velocity and Setpoint Acceleration for the positioner XY.X. A total of 4000 data sets is collected, one data point every 8th servo cycles (for a servo cycle rate defined at 8 kHz), or one data point every 8/8000 s = 0.001 s.

Example 2

Using the terminal screen of the XPS utility, this example shows the sequence of functions to accomplish a time-based data gathering started by a function call.

GroupInitialize(X)

GroupHomeSearch(X)

GatheringConfigurationSet(X.X.SetpointPosition, X.X.FollowingError)

GatheringRun (5000,8)

GroupMoveRelative (X, 10)

GatheringStop()

GatheringStopAndSave()

In this example, gathering is started by a function call. The SetpointPosition and FollowingError of the positioner XY.X are gathered at a rate of 1 kHz (every 8th servo cycle, for the default 8 kHz servo cycle rate). Data gathering is stopped after the relative move is completed.

Gathering will stop automatically once the number of points specified has been collected. However, data will not be saved automatically to a file. The function **GatheringStopAndSave**() must be used to save the data to a file.

It is also possible to halt data gathering at an event. To do so, define another event trigger and assign the action GatheringStop to that event. Use another event trigger and assign the action GatheringRunAppend to continue with gathering. For details, see chapter 11.0: “Event Triggers“.

NOTE

The function GatheringRun() always starts a new internal data gathering and deletes any previous internal gathering data hold in the buffer. If you want to append data to the file, use the function GatheringRunAppend() instead.

12.2 Event-Based (Internal) Data Gathering

The event-based gathering provides a method to gather data at an event. For instance, gathering data at a certain value of a digital or analog input, during a constant velocity state of a motion or on a trajectory pulse.

The event-based data gathering uses the same file as the time-based and the function based data gathering (see sections 12.1 and 12.3). However, unlike the time-based gathering, the event-based gathering appends data to the existing file in memory. This allows gathering of data during several periods or even with different methods in one common file, see examples. To start data gathering in a new file, use the function GatheringReset(), which deletes the current gathering file from memory.

The data type(s) that can be collected with event-based gathering are the same as data for time-based and function-based gathering:

PositionerName.CurrentPosition

PositionerName.SetpointPosition

PositionerName.FollowingError

PositionerName.CurrentVelocity

PositionerName.SetpointVelocity

PositonerName.CurrentAcceleration

PositionerName.SetpointAcceleration

PositionerName.CorrectorOutput

GPIO (ADC, DAC, DI, DO) See Programmer’s manual for a list of all the GPIO Names for the Analog and Digital I/O.

The Setpoint values refer to the theoretical values from the profiler where as the current values refer to the actual or real values of position, velocity and acceleration.

The following sequence of functions is used in event-based data gathering:

GatheringReset()

GatheringConfigurationSet()

EventExtendedConfigurationTriggerSet()

EventExtendedConfigurationActionSet(GatheringOneData,0,0,0,0)

EventExtendedStart()

…

Use the function GatheringStopAndSave() to store the gathered file from the buffer to the flash disk of the XPS controller.

Other functions associated with the event-based gathering are:

GatheringConfigurationGet()

GatheringCurrentNumberGet()

GatheringDataGet()

FileGatheringRename()

Please refer to the programmer's manual for details.

Example 1

GatheringReset()

Deletes gathering buffer in memory.

GatheringConfigurationSet(XY.X.CurrentPosition, XY.Y.CurrentPosition, GPIO2.ADC1)

The 3 data XY.X.CurrentPosition, XY.Y.CurrentPosition and GPIO2.ADC1 will be gathered.

EventExtendedConfigurationTriggerSet(GPIO2.ADC1.ADCHighLimit, 5,0,0,0)

EventExtendedConfigurationActionSet(GatheringOneData,0,0,0,0)

EventExtendedStart()

Data gathering starts when the value of the GPIO2.ADC1 exceeds 5 Volts. One set of data will be gathered at each servo cycle (as the event is checked at each servo cycle). Data gathering automatically stops when the value of the GPIO2.ADC1 falls below 5 V again and the event is automatically removed (see chapter 11.0: "Event Triggers" for details).

Example 2

TimerSet(Timer1, 8)

Sets the timer 1 to 8 servo ticks, means every 1 ms for a servo rate defined at 8 kHz.

GatheringReset()

Deletes gathering buffer from memory.

GatheringConfigurationSet(XY.X.CurrentPosition, XY.Y.CurrentPosition, GPIO2.ADC1)

The 3 data XY.X.CurrentPosition, XY.Y.CurrentPosition and GPIO2.ADC1 will be gathered.

EventExtendedConfigurationTriggerSet(Timer1,0,0,0,0, GPIO2.ADC1.ADCHighLimit,5,0,0,0)

EventExtendedConfigurationActionSet(GatheringOneData,0,0,0,0)

EventExtendedStart()

Different from the previous example, here the event ADCHighLimit is linked to the event Timer1. This has two effects. First, the event becomes permanent as the event timer is permanent. Second, one set of data is gathered only every 1 ms (combination of events must be true). For details on the event definition, please see chapter 11.0: "Event Triggers".

As a result, one set of data is gathered every 1 ms whenever the value of the GPIO2.ADC1 exceeds 5 V for a servo rate defined at 8 kHz.

<u>**Example 3**</u>

TimerSet(Timer1, 8)

Sets the timer 1 to 8 servo ticks, means every 1 ms for a servo rate defined at 8 kHz.

GatheringReset()

Deletes gathering buffer from memory.

GatheringConfigurationSet(XYZ.X.CurrentPosition, XYZ.Y.CurrentPosition, XYZ.Z.CurrentPosition)

EventExtendedConfigurationTriggerSet(Timer1,0,0,0,0, XYZ.Spline.TrajectoryState,0,0,0,0)

EventExtendedConfigurationActionSet(GatheringOneData,0,0,0,0)

EventExtendedStart()

In this example, during the execution of the next spline trajectory on the group XYZ, one set of data will be gathered every 1 ms. In contrast to time-based gathering, which allows programming of a similar function, data gathering will automatically stop at the end of the trajectory. Also, it is not needed to define the total number of data sets that will be gathered.

12.3 Function-Based (Internal) Data Gathering

Function-based gathering provides a method to gather one set of data using a function. It uses the same data file as the time-based and the event-based data gathering, see chapter 13.1 for details. At receipt of the function, one set of data is appended to the gathering file in memory.

The data type(s) that can be collected with the event-based gathering are the same as for the time based and the event-based gathering, see section 12.1 and 12.2 for details.

<u>**Example**</u>

GatheringReset()

Deletes gathering buffer.

GatheringConfigurationSet(XY.X.CurrentPosition, XY.Y.CurrentPosition)

The 2 data XY.X.CurrentPosition and XY.Y.CurrentPosition will be gathered.

GatheringDataAcquire()

Gathers one set of data.

GatheringCurrentNumberGet()

This function will return 1, 500000; 1 set of data acquired, max. 500 000 sets of data can be acquired.

GatheringDataAcquire()

GatheringDataAcquire()

GatheringCurrentNumberGet()

This function will return 3, 500000; 3 sets of data acquired, max. 500,000 sets of data can be acquired.

12.4 Trigger-Based (External) Data Gathering

The trigger-based data gathering allows acquiring position and analog input data at receipt of an external trigger input (Extended GPIO synchronization inputs GPIO5.DI_15 and GPIO5.DI_16 of the XPS, see Appendix B: General I/O Description for more details).

The position data is latched by dedicated hardware. The jitter between the trigger signal and the acquisition of the position data is less than 50 ns. The analog inputs, however, are only latched by an internal interrupt at the servo rate and the XPS will store the most recent value. Hence, the acquired analog input data might be as old as the CorrectorISRPeriod (125 µs for a servo rate of 8 kHz).

NOTE

There must be a minimum time one Corrector Period between two successive trigger inputs.

The data of the trigger-based (external) data gathering is stored in a file named ExternalGathering.dat, which is different from the file used for the internal data gathering (Gathering.dat). Hence, internal and external data gathering can be used at the same time.

The function GatheringExternalConfigurationSet() defines which type of data will be gathered and stored in the data file. The following data types can be collected:

PositionerName.ExternalLatchPosition

These positions refer to the uncorrected encoder position, meaning no error corrections are taken into account. For devices with RS422 differential encoders, the resolution of the position information is equal to the encoder resolution.

For devices with sine/cosine 1 Vpp analog encoder interface, the resolution is equal to the encoder scale pitch divided by the value of the positioner hard interpolator, see function PositionerHardInterpolatorFactorGet(). Its value is set to 20 by default; the maximum allowed value is 200. Please refer to the Programmer's Manual for details.

The external latch positions require that the device has an encoder. No position data can be latched with this method for devices that have no encoder.

GPIO2.ADC1 to GPIO2.ADC2 (referring to the 2 analog input channels on GPIO2 for the Basic GPIO option of the XPS)

GPIO4.ADC1 to GPIO4.ADC8 (referring to the 8 analog input channels on GPIO4 for the Advanced GPIO option of the XPS)

The following sequence of functions is used for a trigger-based data gathering:

GatheringExternalConfigurationSet()

EventExtendedConfigurationTriggerSet()

EventExtendedConfigurationActionSet()

EventExtendedStart()

Other functions associated with trigger-based gathering are:

GatheringConfigurationGet()

GatheringCurrentNumberGet()

GatheringExternalDataGet()

FileGatheringRename()

Please refer to the Programmer's Manual for details.

Example

```
GatheringExternalConfigurationSet(XY.X.ExternalLatchPosition,
GPIO2.ADC1)
EventExtendedConfigurationTriggerSet(Immediate,0,0,0,0)
EventExtendedConfigurationActionSet(ExternalGatheringRun,100,1,0,0)
EventExtendedStart()
```

In this example, a trigger-based (external) gathering is started immediately (with the function EventExtendedStart()). The types of data being collected are the XY.X encoder position and the value of the GPIO2.ADC1. A total of 100 data sets are collected; one set of data at each trigger input. Gathering will stop automatically after the 100th data acquisition. Use the function **GatheringExternalStopAndSave**() to save the data to a file. The file format is the same as for internal data gathering. To rename the gathering file use the API function **FileGatheringRename**().

13.0 Output Triggers

External data acquisition tools, lasers, and other devices can be synchronized to the motion. For this purpose, the XPS features one dedicated trigger output for Axis 1 and Axis 2, see Appendix D: PCO Connector for details. The XPS can be configured to either output distance spaced pulses, AquadB encoder signals, or time spaced pulses on this connector.

In the distance spaced configuration, one output pulse is generated when crossing a defined position and a new pulse is generated at every defined distance until a maximum position has been reached. In most cases, this mode provides the most precise synchronization of the motion to an external tool.

In the AquadB configuration, AquadB encoder signals are output on the PCO connector. These signals can be provided either always or only if the positioner is within a defined position window. When used with stages that feature a digital encoder (AquadB) as opposed to a SinCos encoder (AnalogInterpolated), the AquadB configuration essentially provides an image of the encoder signals on the PCO connector.

In the time flasher configuration, an output pulse is generated when crossing a defined position and a new pulse is generated at a defined time interval until a maximum position has been reached. In some cases, this mode can provide an even more precise synchronization of the motion to an external tool, in particular if the variation of the speed multiplied with the time interval is smaller than the error of the encoder signals during the same period.

Dedicated hardware is used to check the position crossing and the time interval to attain less than 50 ns latency between the position crossing and the trigger output.

For the distance spaced pulses configuration, time flasher configuration or AquadB signals on PCO connector configuration, it is recommended to calibrate the position compare before all PCO pulses generation. It is also recommended to set the position compare hardware to the scanning range you intend to use to get the best performances (*refer to section 13.4: "*Distance, Time Spaced Pulses or AquadB Position Compare*" for details*).

In addition and independent from the above, the XPS controller can output distance spaced pulses on Line-arc trajectories and time spaced pulses on PT and PVT trajectories. In these cases, the distances/time intervals are checked on the servo cycle.

13.1 Triggers on Line-Arc Trajectories

This capability outputs pulses at constant trajectory length intervals on Line-Arc-Trajectories. The pulses are generated between a start length and an end length. All lengths are calculated in an orthogonal XY plane. The StartLength, EndLength, and PathLengthInterval refer to the Setpoint positions.

The trajectory length is calculated at the servo rate. This means that the resolution of the trajectory length for an example servo rate of 8 kHz is 0.000125 * trajectory velocity. For a trajectory velocity of 100 mm/s for instance, the resolution of the trajectory length is 12.5 µm. If the programmed PathLengthInterval is not a multiple of this resolution, the pulses can be off from the ideal positions by a maximum ± half of this resolution.

Two signals are provided for each XPS-RL GPIO option:

GPIO Basic:

GPIO1.DO6, Window: A constant 5 V signal is sent between the StartLength and the EndLength.

GPIO1.DO7, Pulse: A 1µs pulse with 5 V peak voltage is sent every PathLengthInterval.

GPIO Extended:

GPIO5.DO14, Window: A constant 5 V signal is sent between the StartLength and the EndLength.

GPIO5.DO15, Pulse: A 1µs pulse with 5 V peak voltage is sent every PathLengthInterval.

For details about the XPS I/O connectors, see Appendix B: General I/O Description.

To define the StartLength, EndLength, and PathLengthInterval, use the function **XYLineArcPulseOutputSet**().

Example

XYLineArcPulseOutputSet(XY, 10, 30, 0.01)

One pulse will be generated every 10 µm on the next Line-Arc Trajectory between 10 mm and 30 mm.

XYLineArcVerification(XY, Traj.trj)

Loads and verifies the trajectory Traj.trj

XYLineArcExecution(XY, Traj.trj, 10, 100, 1)

Executes the trajectory at a trajectory speed of 10 mm/s and with a trajectory acceleration of 100 mm/s one time.

Please note, that the pulse output settings are automatically removed when the trajectory is over. Hence, with the execution of every new trajectory, it is also required to define the pulse output settings again.

It is also possible to use the trajectory pulses and the pulse window state as events in the event triggers (see chapter 11.0: "Event Triggers" for details). This allows the gathering of data on a trajectory at constant length intervals.

Example

XYLineArcPulseOutputSet(XY, 10, 30, 0.01)

One pulse every 10 µm will be generated on the Line-Arc Trajectory between 10 mm and 30 mm.

XYLineArcVerification(XY, Traj.trj)

Loads and verifies the trajectory Traj.trj

GatheringConfigurationSet(XY.X.CurrentPosition, XY.Y.CurrentPosition, GPIO2.ADC1)

Configures data gathering to capture the current positions of the XY.X and the XY.Y and the analog input GPIO2.ADC1

EventExtendedConfigurationTriggerSet(Always, 0,0,0,0,XY.LineArc.TrajectoryPulse,0,0,0,0)

Triggers an action for every trajectory pulse. The link of the event TrajectorPulse with the event Always is important to make the event permanent. Otherwise, the event will be removed after the first pulse.

EventExtendedConfigurationActionSet(GatheringOneData,0,0,0,0)

Defines the action; gathers one set of data each trajectory pulse.

EventExtendedStart()

Starts the event trigger.

XYLineArcExecution(XY, Traj.trj, 10, 100, 1)

Executes the trajectory at a trajectory speed of 10 mm/s and a trajectory acceleration of 100 mm/s one time.

GatheringStopAndSave()

Saves the gathering data from memory into a file gathering.dat in the ..Admin/Public/Gathering folder of the XPS.

In this example, one set of data will be gathered on the trajectory between length 10 mm and 30 mm at constant trajectory length intervals of 10 µm.

13.2 Triggers on PVT Trajectories

This capability outputs pulses at constant time intervals on a PVT trajectory. The pulses are generated between a first and a last trajectory element (see section 8.3: PVT Trajectories for details). The minimum possible time interval is one servo cycle.

Two signals are provided for each XPS-RL GPIO option:

GPIO Basic:

GPIO1.DO6, Window:	A constant 5 V signal is sent between the StartLength and the EndLength.
GPIO1.DO7, Pulse:	A 1µs pulse with 5 V peak voltage is sent every PathLengthInterval.

GPIO Extended:

GPIO5.DO14, Window:	A constant 5 V signal is sent between the StartLength and the EndLength.
GPIO5.DO15, Pulse:	A 1µs pulse with 5 V peak voltage is sent every PathLengthInterval.

For details about the XPS I/O connectors, see Appendix B: General I/O Description.

To define the first element, the last element and the time interval, use the function **MultipleAxesPVTPulseOutputSet**().

Example 1

MultipleAxesPVTPulseOutputSet(Group1, 3, 5, 0.01)

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

MultipleAxesPVTVerification(Group1, Traj.trj)

Loads and verifies the trajectory Traj.trj

MultipleAxesPVTExecution(Group1, Traj.trj, 1)

Executes the trajectory Traj.trj one time.

Note that the pulse output settings are automatically removed when the trajectory is over. Hence, with the execution of every new trajectory, the pulse output settings must be defined again.

It is also possible to use the trajectory pulses and the pulse window state as events in the event triggers (see chapter 11.0: “Event Triggers” for details). This allows the gathering of data on a trajectory.

Example 2

MultipleAxesPVTPulseOutputSet(Group1, 3, 5, 0.01)

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

MultipleAxesPVTVerification(Group1, Traj.trj)

Loads and verifies the trajectory Traj.trj

GatheringConfigurationSet(Group1.P.CurrentPosition, GPIO2.ADC1)

Configures data gathering to capture the current position of the Group1.P positioner and the analog input GPIO2.ADC1

EventExtendedConfigurationTriggerSet(Always, 0,0,0,0,Group1.PVT.TrajectoryPulse,0,0,0,0)

Triggers an action for every trajectory pulse. The link of the event TrajectorPulse with the event Always is important to make the event permanent. Otherwise, the event will be removed after the first pulse.

EventExtendedConfigurationActionSet(GatheringOneData,0,0,0,0)

Defines the action; gathers one set of data each trajectory pulse.

EventExtendedStart()

Starts the event trigger

MultipleAxesPVTExecution(XY, Traj.trj, 1)

Executes the trajectory Traj.trj one time.

GatheringStopAndSave()

Saves the gathering data from memory in a file gathering.dat in the ..Admin/Public/Gathering folder of the XPS.

In this example, one set of data will be gathered every 10 ms on the trajectory between the start of the 3rd and the end of the 5th element.

13.3 Triggers on PT Trajectories

This capability outputs pulses at constant time intervals on a PT trajectory. The pulses are generated between a first and a last trajectory element (see section 8.4: PT Trajectories for details). The minimum possible time interval is one servo cycle.

Provided signals are:

GPIO Extended:

GPIO1.DO6, Window:	A constant 5 V signal is sent between the StartLength and the EndLength.
GPIO1.DO7, Pulse:	A 1µs pulse with 5 V peak voltage is sent every PathLengthInterval.

GPIO Basic:

GPIO5.DO14, Window:	A constant 5 V signal is sent between the StartLength and the EndLength.
GPIO5.DO15, Pulse:	A 1µs pulse with 5 V peak voltage is sent every PathLengthInterval.

For details about the XPS I/O connectors, see Appendix B: General I/O Description.

To define the first element, the last element and the time interval, use the function **MultipleAxesPTPulseOutputSet**().

<u>**Example 1**</u>

MultipleAxesPTPulseOutputSet (MultipleGroup, 3, 5, 0.01)

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

MultipleAxesPTVerification(MultipleGroup, Traj.trj)

Loads and verifies the trajectory Traj.trj

MultipleAxesPTExecution(MultipleGroup, Traj.trj, 1)

Executes the trajectory Traj.trj one time.

Note that the pulse output settings are automatically removed when the trajectory is over. Hence, with the execution of every new trajectory, the pulse output settings must be defined again.

It is also possible to use the trajectory pulses and the pulse window state as events in the event triggers (see chapter 11.0: “Event Triggers“ for details). This allows the gathering of data on a trajectory.

Example 2

MultipleAxesPTPulseOutputSet(MultipleGroup, 3, 5, 0.01)

One pulse will be generated every 10 ms between the start of the 3rd element and the end of the 5th element.

MultipleAxesPTVerification(Group1, Traj.trj)

Loads and verifies the trajectory Traj.trj

GatheringConfigurationSet(MultipleGroup.Pos1.CurrentPosition, GPIO2.ADC1)

Configures data gathering to capture the current position of the MultipleGroup.Pos1 positioner and the analog input GPIO2.ADC1

EventExtendedConfigurationTriggerSet(Always, 0,0,0,0, MultipleGroup.PT.TrajectoryPulse,0,0,0,0)

Triggers an action for every trajectory pulse. The link of the event TrajectorPulse with the event “Always” is important to make the event permanent. Otherwise, the event will be removed after the first pulse.

EventExtendedConfigurationActionSet(GatheringOneData,0,0,0,0)

Defines the action; gathers one set of data each trajectory pulse.

EventExtendedStart()

Starts the event trigger

MultipleAxesPTExecution(MultipleGroup, Traj.trj, 1)

Executes the trajectory Traj.trj one time.

GatheringStopAndSave()

Saves the gathering data from memory in a file “gathering.dat” in the “Admin/Public/Gathering” folder of the XPS.

In this example, one set of data will be gathered every 10 ms on the trajectory between the start of the 3rd and the end of the 5th element.

13.4 Distance, Time Spaced Pulses or AquadB Position Compare

13.4.1 Even Distance Spaced Pulses Position Compare

In the even distance spaced pulse configuration, one first output pulse is generated when the positioner enters the defined position window. This is independent of the positioner entering the window from the minimum position or from the maximum position. From this first pulse position, a new pulse is generated at every position step until the stage exits the window.

NOTE

To make sure that the trigger pulses are always at the same positions independent of the positioner entering the window from the minimum or from the maximum window position, the difference between the minimum and the maximum window position should be an integer multiple of the position step.

The duration of the trigger pulse is 2 µs by default and can be modified using the function **PositionerPositionComparePulseParametersSet (PositionerName, PCOPulseWidth)**. Possible values for PCOPulseWidth are: 35 ns to 327.68 µs (5 ns resolution). Successive trigger pulses should have a minimum time lag of 625 ns (200 ns for less than 4096 pulses).

The following functions are used to configure the distance spaced pulses:

PositionerPositionCompareSet

PositionerPositionCompareGet

PositionerPositionCompareEnable

PositionerPositionCompareDisable

The function PositonerPositonCompareSet() defines the position window and the distance for the trigger pulses. It has four input parameters:

Positioner Name

Minimum Position

Maximum Position

Position Step

To enable the distance spaced pulses, the function PositionerPositionCompareEnable() must be sent.

Example

GroupInitialize(MyStage)

GroupHomeSearch(MyStage)

PositionerPositionCompareSet(MyStage.X,5, 25, 0.002)

PositonerPositionCompareEnable(MyStage.X)

PositionerPositionCompareGet(MyStage, &MinimumPosition, &MaximumPosition, &PositionStep, &EnableState)

This function returns the parameters previously defined, the minimum position 5, the maximum position 25, the position step 0.002 and the enabled state (1=enabled, 0 =disabled).

GroupMoveAbsolute(MyStage,30)

PositionerPositionCompareDisable(MyStage.X)

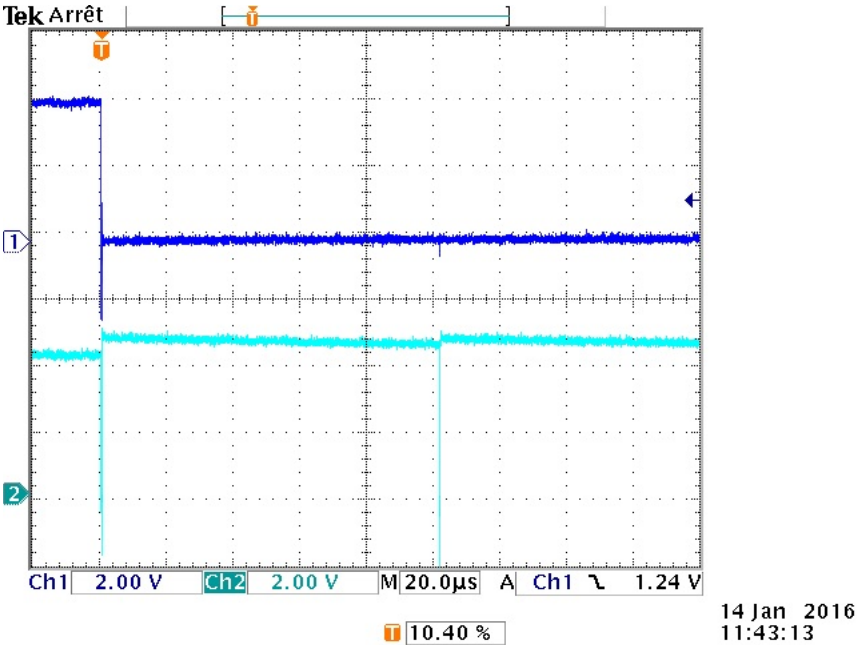
The group has to be in a READY state for the position compare to be enabled. Also, the PositionerPositionCompareSet() function must be completed before the PositionerPositionCompareEnable() function. In this example, one trigger pulse is generated every 0.002 mm between the minimum position of 5 mm and the maximum position of 25 mm. The first trigger pulse will be at 5 mm and the last trigger pulse will be at 25 mm.

The output pulses are accessible from the PCO connector at the back of the XPS controller, see Appendix D: PCO Connector for details.

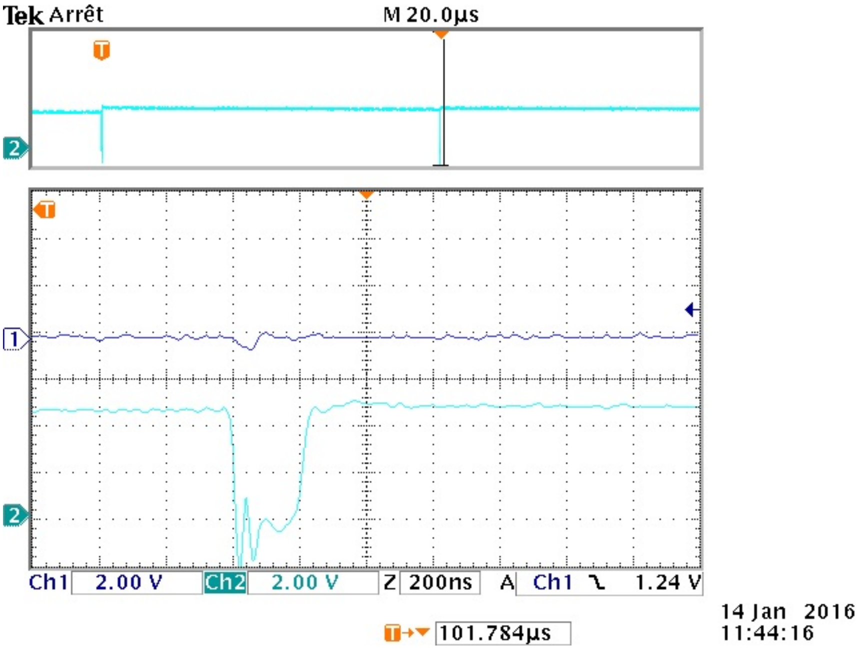
This table summarizes the results of the example above:

Position of the stage	Pulse enable 1 state	Pulse 1 activation	Explanation
0	0	No	Position compare not enabled
5	1	Yes	Position compare enabled, first pulse
5…25	1	Yes	One pulse every 0.002 mm
25	1	Yes	Last pulse
25.002	0	No	Position compare disabled
30	0	No	Position compare disabled

The figure below shows actual screen shots from an oscilloscope for the example above. The enable window is displayed in ch1 and the pulses in ch2:



At position 5 mm, the position compare output functionality becomes active and the first pulse is generated. Then, pulses are generated every 2 µm which equals a time span of 100 µs at a speed of 20 mm/s (2 µm/20 mm/s = 100 µs).



This second picture shows a zoom of the second pulse. The duration of the pulse should be 200 ns.

NOTE

The parameters PositionStep, MinimumPosition, and MaximumPosition (specified with the function PositionerPositionCompareSet) are rounded to the nearest detectable trigger position. When using the Position Compare function with AquadB encoders, the trigger resolution is equal to the EncoderResolution of the positioner specified in the stages.ini. When using the Position Compare function with AnalogInterpolated encoders, the trigger resolution is equal to the EncoderScalePitch defined in the stages.ini divided by 256 (basic PCO) or 65536 (extended PCO).

AnalogInterpolated encoder

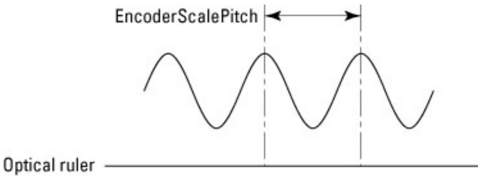


Figure 43: Analog interpolated encoder.

Trigger resolution = $\frac{\text{EncoderScalePitch}}{\text{InterpolationFactor}}$

InterpolationFactor = 256 with basic PCO or 65536 with extended PCO.

Trigger pulses

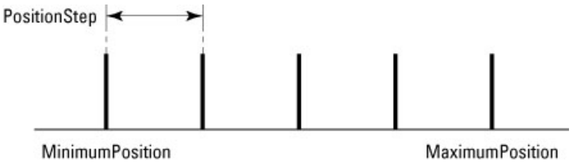


Figure 44: Trigger pulses.

MinimumPosition, MaximumPosition, and PositionStep should be multiples of the Trigger resolution. If not, rounding to the nearest multiple value is made.

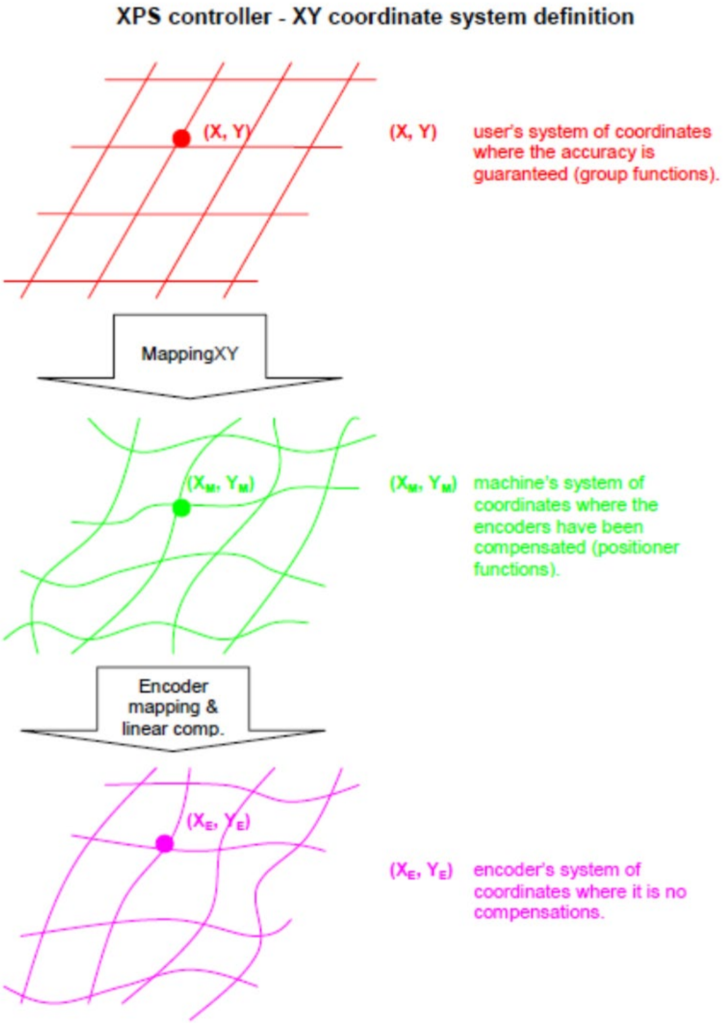
13.4.2 Compensated Position Compare

This feature is used to output a pulse each time the stage moves over user predefined positions compensated through error mapping. Available only for a XPS with the Extended PCO feature.

13.4.2.1 XPS System of Coordinates

To explore the details of the XPS coordinate system, use the example of the XY group but the same is true for the other groups.

The firing positions are defined in the called user's system of coordinates (X, Y). The controller will convert the (X, Y) coordinates to raw encoder positions (X_E, Y_E) to take into account the group mapping, the encoder mapping and the encoder linear compensation to accurately fire the pulses at the requested positions.


XPS controller - XY coordinate system definition
(X, Y)
(X, Y)
user's system of coordinates where the accuracy is guaranteed (group functions).
MappingXY
(X_M, Y_M)
(X_M, Y_M)
machine's system of coordinates where the encoders have been compensated (positioner functions).
Encoder mapping & linear comp.
(X_E, Y_E)
(X_E, Y_E)
encoder's system of coordinates where it is no compensations.

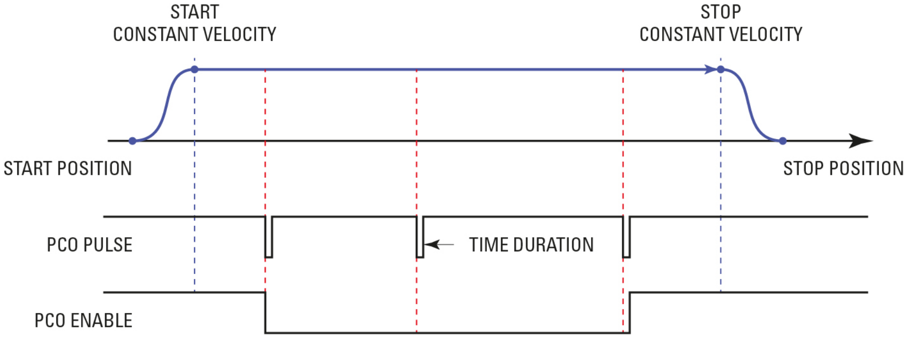
To know the positions in the different systems of coordinates, the following functions are provided:

- *GroupPositionCorrectedProfilerGet()* function has as input a (X, Y) position in the user's system of coordinates and will output the (X_M, Y_M) position in the machine's system of coordinates by applying the XY mapping compensation.
- *XYGroupPositionPCORawEncoderGet()* function has as input a (X, Y) position in the user's coordinate system and will output the (X_E, Y_E) position in the encoder's system of coordinates without any compensation.

13.4.2.2 Compensated position compare signals definition

If the compensated PCO pulses generation is activated, the PCO pulses will be generated at each predefined position with a pulse time duration that can be set with the *PositionerCompensatedFastPCOPulseParametersSet()* function *(cf. XPS Programmer's Manual for details).*

The PCO enable will be generated from the beginning of the first pulse to the end of the last pulse.



The pulse rate is limited by 2 things:

- The minimum time between 2 successive pulse : 200 ns
- The transfer rate of the target positions from CPU memory to CIE internal memory (4096 positions FIFO) : 1.6 MHz

If there are less than 4096 pulses to generate, the only limitation is the time between 2 successive points and the maximum pulses frequency is 5 MHz.

*MinimumTriggerPulseDistance > ScanningVelocity * 200 ns*

For more than 4096 pulses the limitation is the transfer frequency and the maximum frequency is 1.6 MHz.

*MinimumTriggerPulseDistance > ScanningVelocity * 625 ns*

The margin to take in account will depend on many parameters such as the speed stability.

If there are two PCO running on the same CIE board, the maximum transfer frequency of 1.6 MHz is divided by 2, but the minimum time between 2 successives pulses when there are less than 4096 points is still 200 ns.

If the transfer rate limitation is reached, the PCO error flag will be set.

If the time between two successive pulse is too short, the position will not be detected and the pulse generation will stop.

13.4.2.3 Compensated position compare scanning process description

13.4.2.3.1 Scan preparation

- Initialize and home the scanning group:

GroupInitialize(Group) *; Initialize scanning group*

GroupHomeSearch(Group) *; Search home for the scanning*

13.4.2.3.2 Scan execution

- If needed, set PCO pulse duration and polarity/type (pulse or toggle):

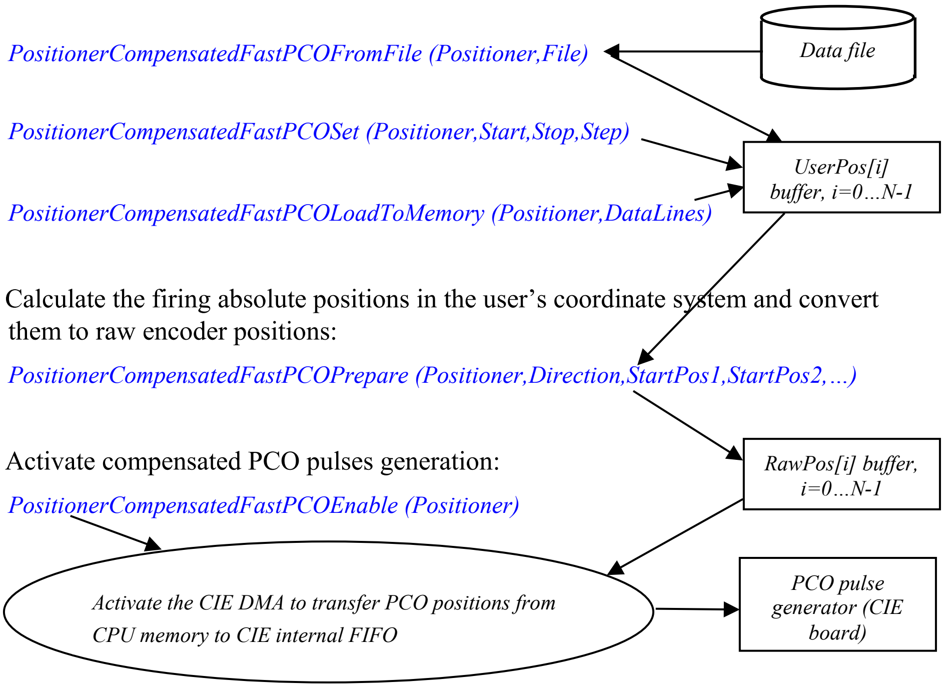
PositionerCompensatedFastPCOPulseParametersSet(Positioner,PulseDuration,PulsePolarity,PulseToggle)

- Move the scanning group to the scan start position (*outside of the scanning zone*):

GroupMoveAbsolute (Group, Position1, Position2, …)

- Set the firing positions by reading data from file or loading to controller's memory or with a "set" function.

Note that the firing positions defined with the following functions are only the offsets relative to the scanning positioner start position, that will be specified with the *PositionerCompensatedFastPCOPrepare()* function.



PositionerCompensatedFastPCOFromFile (Positioner,File)

PositionerCompensatedFastPCOSet (Positioner,Start,Stop,Step)

PositionerCompensatedFastPCOLoadToMemory (Positioner,DataLines)

- Calculate the firing absolute positions in the user's coordinate system and convert them to raw encoder positions:

PositionerCompensatedFastPCOPrepare (Positioner,Direction,StartPos1,StartPos2,…)

- Activate compensated PCO pulses generation:

PositionerCompensatedFastPCOEnable (Positioner)

Data file
UserPos[i] buffer, i=0…N-1
RawPos[i] buffer, i=0…N-1
Activate the CIE DMA to transfer PCO positions from CPU memory to CIE internal FIFO
PCO pulse generator (CIE board)

- Set motion parameters for scan:

  PositionerSGammaParametersSet(Positioner, ScanVelocity, ScanAcceleration, MinimumJerkTime,MaximumJerkTime)

- Move the scanning positioner across the scanning zone, during this move the firing pulses will be generated:

  GroupMoveRelative (Positioner, ScanDistance)

13.4.2.4 Compensated position compare related functions

Here is the list of the associated functions with a brief description. For detailed information, refer to the *XPS Programmer's Manual*.

13.4.2.4.1 Firing positions definition

There are three ways to enter the firing positions: reading from file, writing directly to the controller's memory or calculating with a "set" function.

Firing positions definition from a data file

Function *PositionerCompensatedFastPCOFromFile(Positioner, FileName)* reads firing positions from a data file to the controller's memory.

Firing positions definition from a "load to memory" function

Function *PositionerCompensatedFastPCOFromFile(Positioner, DataLines)* appends firing positions to the controller's memory from *DataLines* parameter.

To reset the controller's memory, the *PositionerCompensatedFastPCOMemoryReset()* function is provided.

Firing positions definition from a "set" function

Function *PositionerCompensatedFastPCOSet (Positioner,Start,Stop,Step)* calculates a set of evenly spaced firing positions to the controller's memory.

13.4.2.4.2 *Firing positions preparation*

Function *PositionerCompensatedFastPCOPrepare (Positioner, ScanDirection, StartPosition1, StartPosition2, ...)* calculates the firing at absolute positions, *in user's coordinate system* and converts them to firing absolute raw PCO positions, *in encoder's coordinate system*.

When mappings are enabled, the correction between the user's coordinate system position and raw encoder position will be different at each different location. For this reason, the prepare function must know the location (*positions of all positioners in the scanning group*) where the scan will be done.

13.4.2.4.3 *Associated functions*

Pulses generation enable

Function *PositionerCompensatedFastPCOEnable (Positioner)* activates the compensated PCO pulses generation *(status becomes running (value 1))*. The pulses will be generated when the scanning positioner will move across the predefined positions. When the last pulse is generated, the compensated PCO mode will become inactive (*status becomes inactive (value 0))*. To get the status of the compensated PCO pulses generation, use the *PositionerCompensatedFastPCOCurrentStatusGet()* function.

Note that only the scanning positioner positions are used to fire pulses: if you prepare a set of positions at a given location but you enable the firing pulses generation and start the move from a different location, the pulses could be generated but their accuracy will be impacted by the mapping difference between the two locations.

Pulses generation abort

Function *PositionerCompensatedFastPCOAbort (Positioner)* disables the compensated PCO pulses generation. The pulses generation is stopped immediately; no more pulse will be generated even if the scanning positioner continues to move across the predefined firing positions. To stop the scanning move, use *GroupMoveAbort()* function.

Pulses data reset

The function *PositionerCompensatedFastPCOMemoryReset (Positioner)* resets the compensated PCO data memory. This function is useful to remove the data that was previously entered with the *PositionerCompensatedFastPCOLoadToMemory()* function.

Pulses generation status get

The function *PositionerCompensatedFastPCOCurrentStatusGet (Positioner, Status)* gets the current status of compensated PCO pulses generation.

13.4.3 Time Spaced Pulses (Time Flasher)

In the time spaced configuration, a first pulse is generated when the motion axis enters the time pulse window. From this first pulse, a new pulse is generated at every time interval until the positioner exits the time pulse window.

Hardware attains less than 5 ns jitter for the trigger pulses. The duration of the pulse is 2 µs by default and can be modified using the function **PositionerPositionComparePulseParametersSet**(). Possible values for the PCOPulseWidth are: 35 ns to 327.68 µs (5 ns resolution). Successive trigger pulses should have a minimum time lag of 50 ns.

The following functions are used to generate time spaced pulses:

PositionerTimeFlasherSet

PositionerTimeFlasherGet

PositonerPositionCompareEnable

PositionerTimeFlasherEnable

PositionerTimeFlasherDisable

The function PositonerTimeFlasherSet() defines the position window and the time intervals for the trigger signals. It has four input parameters:

Position Name

Minimum Position

Maximum Position

Time Interval

The time interval must be greater than or equal to 0.00000005 seconds (50 ns) and less than or equal to 21.47483648 seconds. Furthermore, the time interval must be a multiple of 5 ns.

To enable the time spaced pulses, the function PositionerTimeFlasherEnable() must be sent.

Example 1

GroupInitialize(MyStage)

GroupHomeSearch(MyStage)

PositionerTimeFlasherSet(MyStage.X,5, 25, 0.00001)

PositonerPositionCompareEnable(MyStage.X)

PositonerTimeFlasherEnable(MyStage.X)

GroupMoveAbsolute(MyStage,30)

PositionerTimeFlasherDisable(MyStage.X)

The group has to be in a READY state for the time flasher to be enabled. Also, the PositionerTimeFlasherSet() function must be completed before the PositionerTimeFlasherEnable() function. In this example, one trigger pulse is generated every 0.00001 seconds or at a rate of 100 kHz between the minimum position of 5 mm and the maximum position of 25 mm. The first trigger pulse will be at 5 mm and the last trigger pulse will be at 25 mm or before.

The output pulses are accessible from the PCO connector at the back of the XPS controller, See Appendix D: PCO Connector for details.

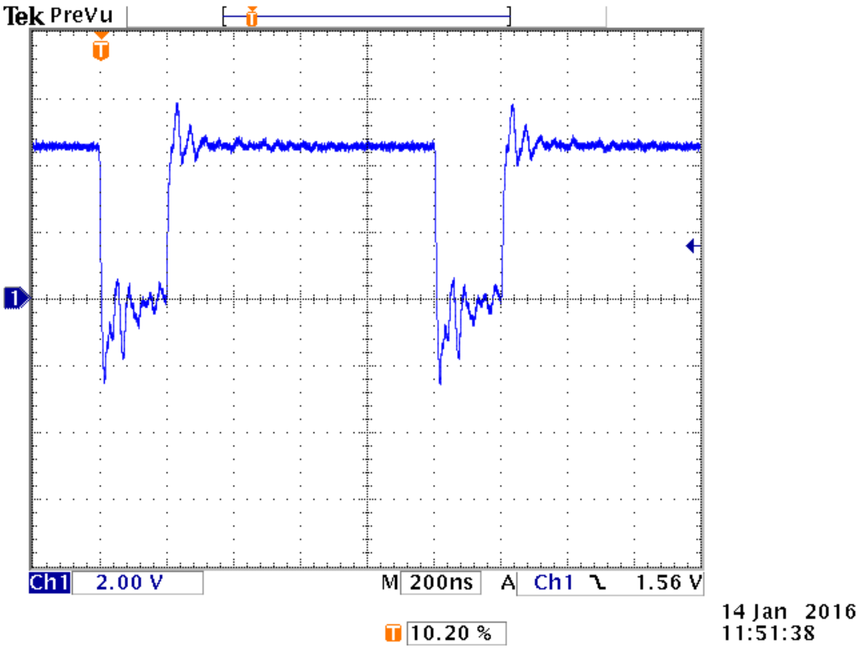


Figure 45: Temporal resolution of time spaced pulses in oscilloscope view.

Example 2

The time flasher function is of particular use with high precision (direct drive) stages. At high speeds, these stages typically provide very good speed stability. In other words, the position change over a short time interval is highly consistent and repeatable. Hence, time spaced pulses can be used for synchronization with similar, in some cases even higher precision as distance spaced pulses. The time spaced pulse configuration, however, provides some further flexibility with regards to the nominal distance between successive triggers.
Consider an XM stage for instance. XM stages feature an analog encoder with 4 µm signal period. The max. resolution of the distance spaced pulses is 15.625 nm (256x interpolation). If the goal is to get pulses at a nominal distance of 272.5 nm at a speed of 200 mm/s speed, this is not possible using the distance spaced pulse configuration. Either 265.625 nm or 281.25 nm are possible, but not 272.5 nm. With some minor adjustments to the target speed, however, this is possible using the time spaced pulse configuration:

- The target speed is 200 mm/s, the desired distance between successive pulses is 272.5 nm. So the nominal time interval between successive pulses is:
  272.5 nm/200 mm/s = 1.3625 µs
- Round this nominal value to the next possible time interval, means to the next integer multiple of 5 ns : 1.365 µs
  Use this rounded time interval to calculate a corrected velocity:
  272.5 nm/1.365 µs = 199.6337 mm/s

PositionerSGammaVelocityAndAccelerationSet(MyStage.X,199.6337,2500)
PositionerTimeFlasherSet(MyStage.X, -30, 30, 0.000001365)
PositionerTimeFlasherEnable(MyStage.X)
GroupMoveAbsolute(MyStage.X)
PositionerTimeFlasherDisable(MyStage.X)

In this example, a first pulse is generated when the stage crosses the position -30 mm. Further pulses are generated every 1.350 µs until the stage reaches the maximum position of +30 mm. Since the stage moves at a speed of 199.6337 mm/s, the nominal distance between successive pulses is: 199.6337 mm/s * 1.365 µs = 272.5 nm.

13.4.4 AquadB Signals on PCO Connector

In the AquadB signals configuration, AquadB encoder signals are provided on the PCO connector, see Appendix E, PCO connector for details and pinning. These signals are either output always (Always configuration), or only when the positioner is within a defined position window (Windowed configuration).

When used with stages that feature a digital encoder (AquadB), the AquadB signals are the same as the encoder signals of the stage. When used with SinCos encoders (AnalogInterpolated), the resolution of the AquadB signal is defined by the signal period of the encoder and the settings of the prescaler by the function PositionerPositionCompareAquadBPrescalerSet().

Example

XM stages feature an analog encoder with a signal period of 4 µm. With the setting PositionerPositionCompareAquadBPrescalerSet(MyStage.X,256) the post-quadrature resolution of the AquadB signals is: 4 µm/256 = 0.015625 µm. In this case one full period of the AquadB signals equals 0.0625 µm.

The following functions are used to configure AquadB signals:

PositionerPositionCompareAquadBWindowedSet

PositionerPositionCompareAquadBWindowedGet

PositionerPositionCompareEnable

PositionerPositionCompareAquadBAlwaysEnable

PositionerPositionCompareDisable

PositionerPositionCompareAquadBPrescalerSet

The function PositonerPositonCompareAquadBAlwaysEnable() has only one input parameter, the positioner name. When sent, AquadB signals are generated always. To disable this mode use the function PositionerPositionCompareDisable().

The function PositonerPositonCompareAquadBWindowedSet() has three input parameters.

Positioner name

Minimum Position

Maximum Position

To enable the AquadB signals, the function PositionerPositionCompareEnable() must be sent.

Example

GroupInitialize(MyStage)

GroupHomeSearch(MyStage)

PositionerPositionCompareAquadBWindowedSet(MyStage.X, 10, 20)

PositonerPositionCompareEnable(MyStage.X)

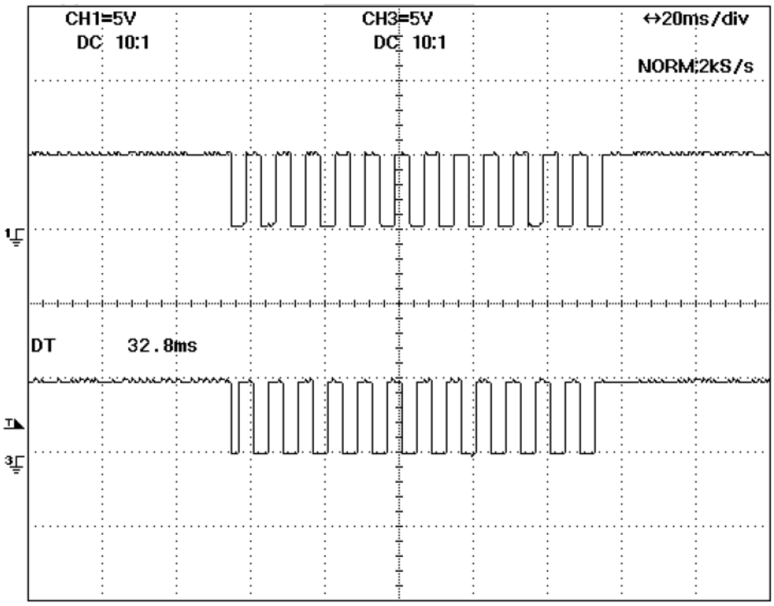
PositionerPositionCompareGet(MyStage, &MinimumPosition, &MaximumPosition, &EnableState)

This function returns the parameters previously defined, the minimum position 10, the maximum position 20 and the enabled state (1=enabled, 0 =disabled).

GroupMoveAbsolute(MyStage,30)

PositionerPositionCompareDisable(MyStage.X)

The figure below shows a screen shots from an oscilloscope for the example above.



The group has to be in a READY state for the position compare to be enabled. Also, the PositionerPositionCompareAquadBWindowedSet() function must be completed before the PositionerPositionCompareEnable() function. In this example, AquadB signals are generated when the positioner is between the minimum position of 10 mm and the maximum position of 20 mm.

NOTE

The AquadB signal configuration is only available with positioners that have an encoder (AquadB or AnalogInterpolated).

The AquadB signals can not be provided at the same time as the distance spaced pulses (PCO) or the time spaced pulses.

The function PositionerPositionCompareEnable() enables always the last configuration sent, either distance spaced pulses defined with the function PositionerPositionCompareSet() or AquadB pulses defined with the function PositionerPositionCompareAquadBWindowedSet().

14.0 Control Loops

14.1 XPS Servo Loops

14.1.1 Servo structure and Basics

The XPS controller can be used to control a wide range of motion devices, which are categorized by the XPS as “positioners”. Within the structure of the XPS' firmware, a “positioner” is defined as an object with an associated profile (trajectory), a PID corrector, a motor interface, a driver, a stage and an encoder.

The general schematic of a positioner servo loop is below.

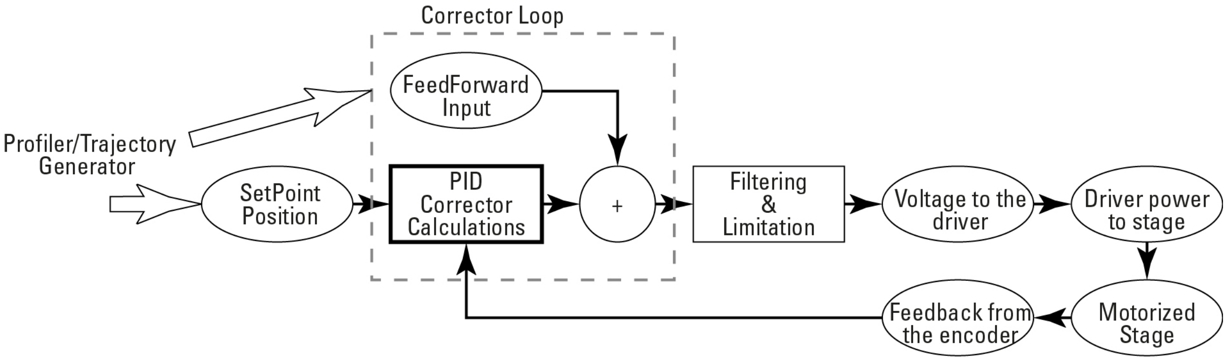


Figure 46: Servo structure and basics.

The calculations done by the “servo loop” result in a voltage output from the controller that is applied to the driver, which can be either any of Newport's Universal drive modules or to an external driver through the XPS pass-through module. Depending on the corrector loop type selected, the level of this output voltage can be the result of two gain factors, the PID corrector and the FeedForward loop. The XPS has imbedded configuration files that provide optimized corrector loop settings for all Newport stages. Non-Newport stages may need to be assigned a specific corrector loop setting during the set-up process. In addition to the two main gain loops the XPS also adds filtering and error compensation parameters to this servo loop to improve system response and reliability.

The profiler (Trajectory Generator) within the controller calculates in real time, the position, velocity, and acceleration/deceleration that the positioner must follow to reach its commanded position (Setpoint Position). This profile is updated at the ProfileGenerator rate (default is 2 kHz). The ProfileGenerator rate is defined in relation to the servo rate given by the expression:

ProfileGenatorISRRatio = Servo Rate/ ProfileGenerator Rate.

The ProfileGenatorISRRatio and the CorrectorISRPeriod (CorrectorISRPeriod = Servo Rate^{-1}) values are found in the system.ref file.

Example system.ref file

```
[GENERAL]
OptionalModuleNames =
CorrectorISRPeriod = 125e-6                ; seconds
IRQDelay = 6e-6                            ; seconds
DACUpdateDelay = 110e-6                    ; seconds
ProfileGeneratorISRRatio = 4
ServitudesISRRatio = 10
GatheringBufferSize = 1000000              ; data count
DelayBeforeStartup = 0                     ; seconds
DebugTraceCommunicationBufferSize = 0      ; characters, if 0 => no trace
```

The PID corrector then compares the SetpointPosition, as defined by the profiler, and the current position, as reported by the positioner's encoder, to determine the current following error. The PID corrector then outputs a value that the controller uses to maintain, increase or decrease the output voltage, which is applied to the driver. This loop is updated at servo rate (default value is 8 kHz). The adjustment of the PID parameters allows users to optimize the performance of their positioner or system by increasing or decreasing the responsiveness of the output to increasing or decreasing following errors. Refer to the section 14.3 on PID tuning for more information and tips on PID tuning. The PID corrector loop and trajectory generation loop default rates have been optimized to provide the highest level of precision. In most applications the critical control loop is the PID corrector since it has the most significant impact on positioning performance. Because of this with default values, the PID loop is updated 4 times (8/2) during each profiler cycle to improve profile execution and minimize following errors.

The Feed-Forward gain generates a voltage output to the driver that is directly proportional to the input. The purpose of this gain is to generate a movement of the positioner as close as possible to the desired move that is independent of the encoder feedback loop. Adding this Feed-Forward gain can help reduce any encountered following errors and thus requires less compensation by the PID gain corrector. For example, if a driver and positioner respond to a constant voltage by moving at a constant speed, then feed forward input would be dictated by the SetpointSpeed.

The XPS stores standard Newport stage configuration files that can be used to quickly and easily develop the stage and system initialization (.ini) files. Below is an example of a typical stage and the type of DriverName, MotorDriverInterface and CorrectorType each is assigned. These standard Newport settings will be optimal for virtually every application and users would only need to modify their corrector loop parameters (Kp, Kd, Ki) to optimize positioner performance. Similar configurations can be adopted for non-Newport stages that are of similar motor driver types.

◆ Stages with high current (> 3 A) DC motor (RV, IMS) (with tachometer or back-emf estimation):

  DriverName: XPS-DRV01, 03

  ◇ ±10 V Input gives ±ScalingVelocity (stage velocity).

  ◇ Speed loop & Current loop configured by hardware.

  MotorDriverInterface: AnalogVelocity

  CorrectorType: PIDFFVelocity for Speed loop and PIDFFAcceleration for current loop.

◆ Stages with DC motor driven through a current loop (RGV) (no tachometer):

  DriverName: XPS-DRV02

  ◇ ±10 V Input gives ±ScalingAcceleration (stage acceleration).

  ◇ Current loop configured by hardware.

  MotorDriverInterface: AnalogAcceleration

  CorrectorType: PIDFFAcceleration

◆ Stages with low current (< 3 A) DC motor & tachometer (VP):

  DriverName: XPS-DRV01 in velocity mode.

  ◇ Input 1: ±10 V results in ±ScalingVelocity (theoretical stage velocity).

  ◇ Input 2: ±10 V results in ±ScalingCurrent (3 A).

  ◇ Speed loop programmable.

  MotorDriverInterface: AnalogVelocity

  CorrectorType: PIDFFVelocity

◆ Stages with low current (<3 A) DC motor, without tachometer (ILSCC type):

  DriverName: XPS-DRV01 in voltage mode.

  ◇ Input 1: ±10 V results in ±ScalingVoltage (48 V).

  - ◇ Input 2: ±10 V results in ±ScalingCurrent (3 A).

  MotorDriverInterface: AnalogVoltage

  CorrectorType: PIDDualFFVoltage

- ◆ Stages with Stepper motor & Encoder (UTSPP, RVPE, ILSPP…):

  DriverName: XPS-DRV01 in stepper mode.

  - ◇ Input 1: ±10 V results in ±ScalingCurrent in motor winding 1.
  - ◇ Input 2: ±10 V results in ±ScalingCurrent in motor winding 2.

  MotorDriverInterface: AnalogStepperPosition

  CorrectorType: PIPosition

- ◆ Stages with Stepper motor & no encoder (TRA, SR50PP, PR50PP, MFAPP):

  DriverName: XPS-DRV01 in stepper mode.

  - ◇ Input 1: ±10 V results in ±ScalingCurrent in motor winding 1.
  - ◇ Input 2: ±10 V results in ±ScalingCurrent in motor winding 2.

  MotorDriverInterface: AnalogStepperPosition

  CorrectorType: NoEncoderPosition

These are just examples of available positioner associations in the XPS. The flexibility of positioner associations allows many other configurations to be developed to drive non-Newport positioners or other products. Before developing other configurations, the user must be aware that the main goal of creating these associations is to match the servo loop output to the appropriate driver input as stated by the manufacturer. For instance:

- The Corrector PIPosition is used when a constant voltage applied to a driver results in a constant position of the positioner (stepper motor, piezo, electrostrictive, etc.).
- Corrector PIDFFVelocity is used when a constant voltage applied to a driver results in a constant speed of the positioner (DC motor and driver board in speed loop mode).
- Corrector PIDFFAcceleration is used when a constant voltage applied to a driver results in a constant acceleration of the positioner (DC motor and driver board in current loop mode).
- Corrector PIDDualFFVoltage is used when a constant voltage applied to a driver results in a constant voltage applied to the motor (DC motor and driver board with direct PWM command).

14.1.2 XPS PIDFF Architecture

Corrector loops PIDFFVelocity, PIDFFAcceleration and PIDFFDualVoltage all use the same architecture as the PID corrector that is detailed below. PIPosition is a simplified version of this loop that is used to provide closed loop positioning via encoder feedback to stepper motor positioners.

14.1.2.1 PID Corrector Architecture

The PID corrector uses the following error (SetpointPosition – EncoderPosition) as its input and applies the sum of three correction terms (Kp, Kd and Ki) to determine the output.

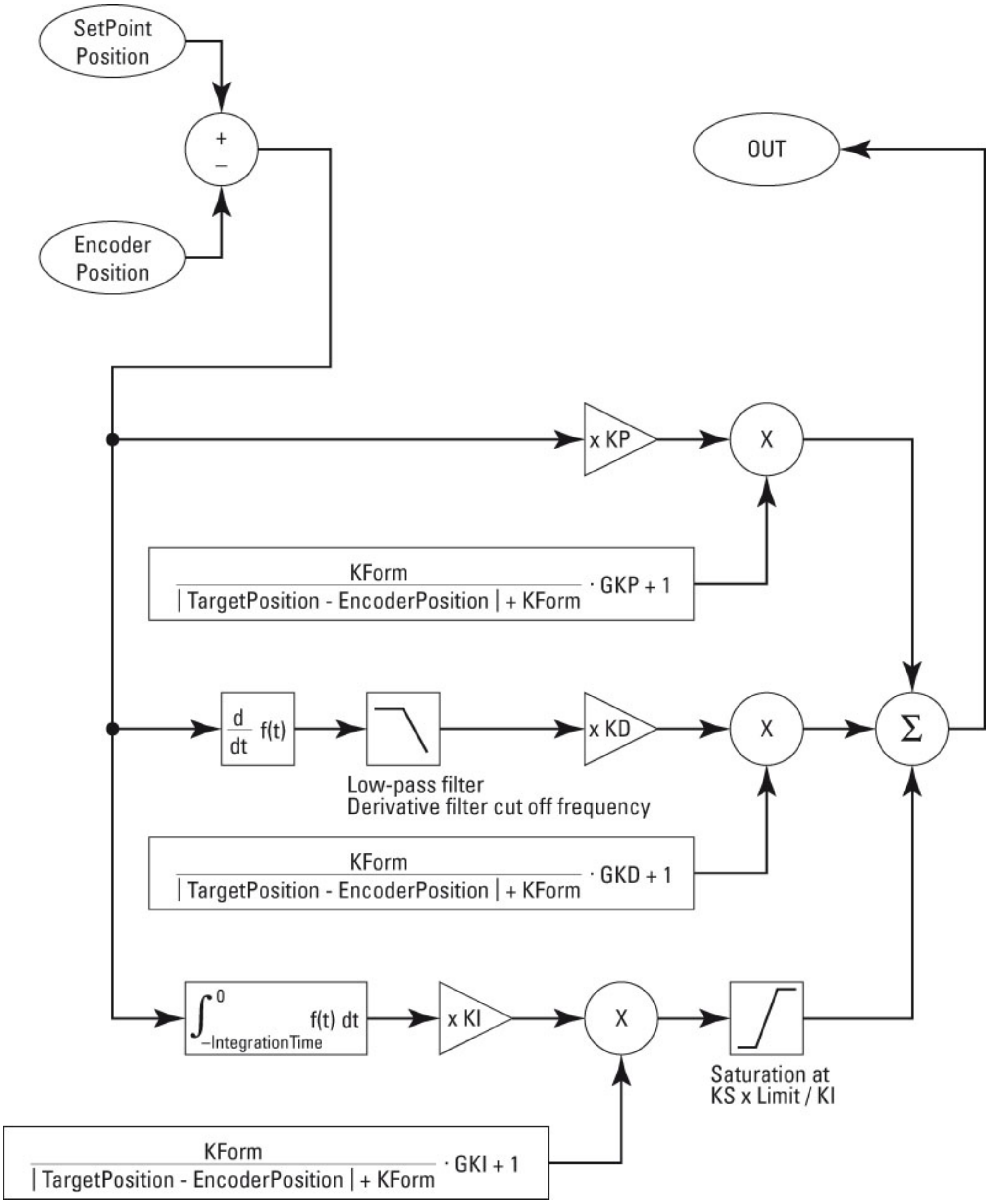

SetPoint Position
+
–
Encoder Position
OUT
x KP
X
KForm / | TargetPosition - EncoderPosition | + KForm · GKP + 1
d/dt f(t)
x KD
X
Σ
Low-pass filter
Derivative filter cut off frequency
KForm / | TargetPosition - EncoderPosition | + KForm · GKD + 1
∫ 0 −IntegrationTime f(t) dt
x KI
X
Saturation at
KS x Limit / KI
KForm / | TargetPosition - EncoderPosition | + KForm · GKI + 1

Figure 47: PID corrector architecture.

14.1.2.2 Proportional Term

The **Kp**, or proportional gain, multiplies the current following error of that servo cycle by the proportional gain value (Kp). The effect is to react immediately to the following error and attempt to correct it. Changes in position generally occur during commanded acceleration, deceleration, and in moves where velocity changes occur in the system dynamics during motion. As Kp is increased, the PID corrector will respond with a increased output and the error is more quickly corrected. For instance, if a positioner or group of positioners is expected to have small following errors, as is the case for small moves where overcoming static friction of the system is predominant, then the Kp may need to be increased to produce sufficient output to the driver. For larger moves, the following errors are generally larger and require lower Kp values to produce the desired output. Also note that for larger moves the kinetic friction of the system is generally much lower than static friction and would generally require less correction gain than smaller moves. However, if Kp becomes too large, the mechanical system may begin to overshoot (encoder position > SetpointPosition), and at some point, it may begin to oscillate, becoming unstable if it does not have sufficient damping.

Kp cannot completely eliminate errors. However, since as the following error e, approaches zero, the proportional correction element, Kp x e, also approaches zero and results in some amount of steady-state error. For this reason other gain factors like Kd and Ki are required.

14.1.2.3 Derivative Term

The **Kd**, or derivative gain, multiplies the differential between the previous and current following error by the derivative gain value (Kd). The result of this gain is to stabilize the transient response of a system and can also be thought of as electronic damping of the Kp. The derivative acts as a gain that increases with the frequency of the variations of the following error:

$$\frac{\mathbf{d}}{\mathbf{dt}}=\left[\mathbf{sin}\left(2\pi\ \mathbf{Fr\ t}\right)\right]=2\pi\ \mathbf{Fr\ cos}\left(2\pi\ \mathbf{Fr\ t}\right)$$

The result is that the derived term becomes dominant at high frequencies, compared to the proportional and integral terms. For the same reason, the value of Kd is in most cases limited by high frequency resonance of the mechanics. This is why a low pass filter (cut off frequency = DerivativeFilterCutOffFrequency) is implemented in the derivative branch to limit excitation at high frequencies. Increasing the value of Kd increases the stability of the system. The steady-state error, however, is unaffected since the derivative of the steady-state error is zero.

These two gains alone can provide stable positioning and motion for the system. However to eliminate the steady state errors, an additional gain value must be used.

14.1.2.4 Integral Term

The Integral term **Ki** acts as a gain that increases when the frequency of the variations of the following error decrease:

$$\int\left[\mathbf{sin}\left(2\pi\ \mathbf{Fr\ t}\right)\right]=-\left(\frac{1}{2}\pi\ \mathbf{Fr}\right)\mathbf{sin}\left(2\pi\ \mathbf{Fr\ t}\right)$$

The result is that the integral term becomes dominant at low frequencies, compared to the proportional and derivative terms. The gain becomes infinite when frequency = 0. Even a very small following error will generate an infinite value of the integral term. The advantage of the integral term is that it will eliminate any steady-state following error. However, the disadvantage is that the integral term can reach values where the corrector is saturated causing the system to become unstable at the end of a move and cause the positioner to hunt or dither. To reduce this effect, two additional parameters are included in the PID corrector to help prevent these instabilities, Ks and Integration Time.

Ks

The saturation limit factor Ks permits users to limit the maximum value of Ki that is applied to the total PID corrector output. The Ks saturation limit can be set between 0 and 1, a typical setting is 0.5. As an example, at a setting of 0.5, the maximum output generated by the Ki term applied to the PID output would be 0.5 x the maximum set output. However, if the Ki gain factor output is less than 0.5 x the maximum set output, then the entire gain will be applied to the PID corrector. This maximum output is set within the section MotorDriverInterface in the stages.ini using the parameters AccelerationLimit, VelocityLimit or VoltageLimit.

Integration Time

The IntegrationTime is used to adjust the duration for integration of the residual errors. This can help in applications where large following errors can occur during motion. The use of a small Integration Time value will limit the integration range to the latter parts of the move, avoiding the need of a large overshoot at the end of the move to clear the integrated following error value. The drawback is that the static error will be less compensated.

14.1.2.5 Variable Gains

In addition to the classical Kp, Ki, and Kd gain parameters, the XPS PID Corrector Loop also includes variable gain factors GKp, GKd, and GKi. These can be used to reduce settling time on systems that have nonlinear behavior or to tighten the control loop during the final segment of a move. For example, a positioner or stage with a high level of friction will have a response which is dependent on the size of the move: friction is negligible for a large move but becomes a predominant factor for small moves. For this reason, the required response of the system to reach the commanded position is not the same for small and large moves. The optimum value of PID parameters for small moves is very often higher than the optimum value for large moves. It is advantageous to modify PID settings depending on the move size. For users that do not need to make PID corrector adjustments (or prefer not to) benefit from the compensations provided by the variable gain correctors. This compensation is made automatically by the XPS variable gain corrector by applying a gain that is driven by the distance between the Target Position (position that must be reached at the end of the motion) and the Encoder Position. As shown in the figure below, when the distance to move completion is large, the total output gain from these parameters is fractional (the “Kform term” is fractional), but as the move size or distance to final position is small the Kform term approaches 1 and full GKx output is provided.

GKp = 10 Kp = 2
Target Position = 0
Encoder Position = -100 to 100

$$Kp_{(\text{Kform, Encoder Position})} = \left[1 + GK \cdot \left(\frac{\text{Kform}}{|\text{Target Position - Encoder Position}| + \text{Kform}}\right)\right] \cdot Kp$$

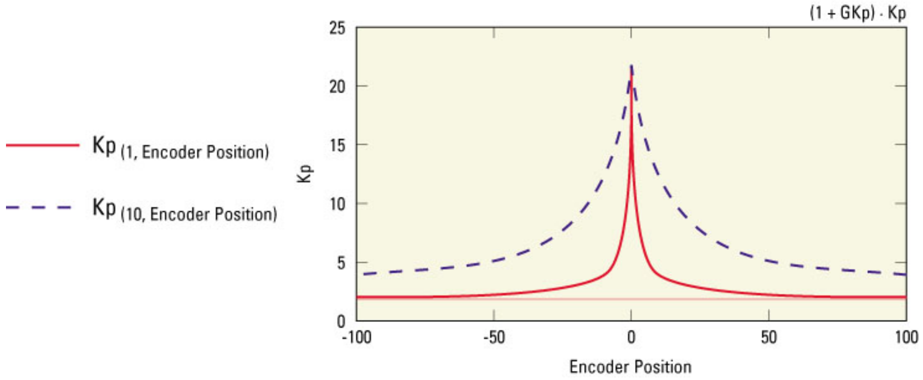


Figure 48: Variable gains.

The parameter GKx is used to adjust the amplitude of the total output and the parameter Kform is used set how soon this Gkx is applied. As seen in the figure below, if a Kform of 1 is implemented, the GKx is not applied until the positioner is very close to its target position, in this case 0. But a Kform of 10 will implement the GKx much sooner and tighten the control of the loop further from the target position. This can be very effective when positioning high inertial loads or when very short settling times are critical. The default setting for the Kform parameter is 0 for all standard Newport stages.

14.2 Filtering and Limitation

In addition to the various PID correctors and calculations, filtering and limitation parameters also have the same structure for all the correctors (PIDFFVelocity, PIDFFAcceleration and PIDFFDualVoltage, etc).

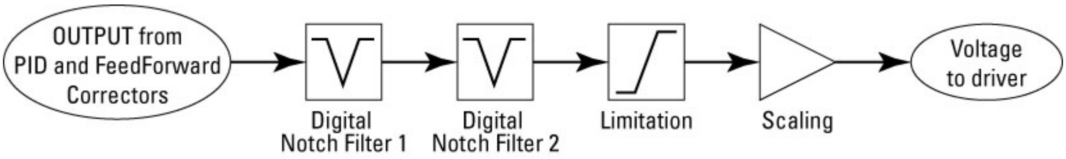


Figure 49: Filtering and limitation.

The first section of the above diagram shows the succession of two digital notch filters. Each filter is defined by its central frequency (NotchFrequency), its bandwidth (NotchBandwidth) and its gain (NotchGain).

The gain, usually in the range of 0.01 to 0.1, is the value of the amplification of a signal at a frequency equal to the central frequency and the bandwidth is the range about the central frequency for which this gain is equal to a -3 db reduction.

Notch filters are typically used to avoid the instability of the servo loop due to the mechanic's natural frequencies, by lowering the gain at these frequencies. When they are implemented, these filters add some phase shift to the signal. This phase shift increases with the filter bandwidth and must remain small in the frequency range where the servo loop is active to maintain stability. The result is that notch filters are only effective at avoiding instabilities due to excessive and constant natural frequencies.

The last section of the diagram shows the limitation and scaling features. Scaling is used to transform units of position, speed or acceleration to a corresponding voltage. The Limitation factor is a safety that is used to limit the maximum voltage that can be applied to the driver to protect against any runaway or saturation situations that may occur.

14.2.1 Current velocity and current acceleration

In XPS controller, the current velocity and current acceleration are calculated from the current position by successive derivative calculations. Because of derivative calculations these values are noisy and must be filtered by a low-pass filter to become exploitable.

Current velocity and acceleration filter parameters (stages.ini):

- Current velocity cut-off frequency: CurrentVelocityCutOffFrequency
- Current acceleration cut-off frequency: CurrentAccelerationCutOffFrequency

The CurrentVelocityCutOffFrequency (Hz) and CurrentAccelerationCutOffFrequency (Hz), set the cut-off frequencies for the low-pass filters that are applied to the CurrentVelocity and CurrentAcceleration. This filter reduces the derivative noises. They must be greater than zero (filter disabled) and less than half of the servo loop frequency (1/CorrectorISRPeriod (see system.ref)). The default value is 100 Hz which is about five times greater than the bandwidth of the position servo loop of a typical screw driven stage.

14.3 Feed Forward Loops and Servo Tuning

14.3.1 Corrector = PIDFFVelocity

The PIDFFVelocity corrector should be implemented into applications where the positioner driver requires a “speed” input (constant voltage to the driver provides constant speed output to the positioner), using MotorDriverInterface = AnalogVelocity.

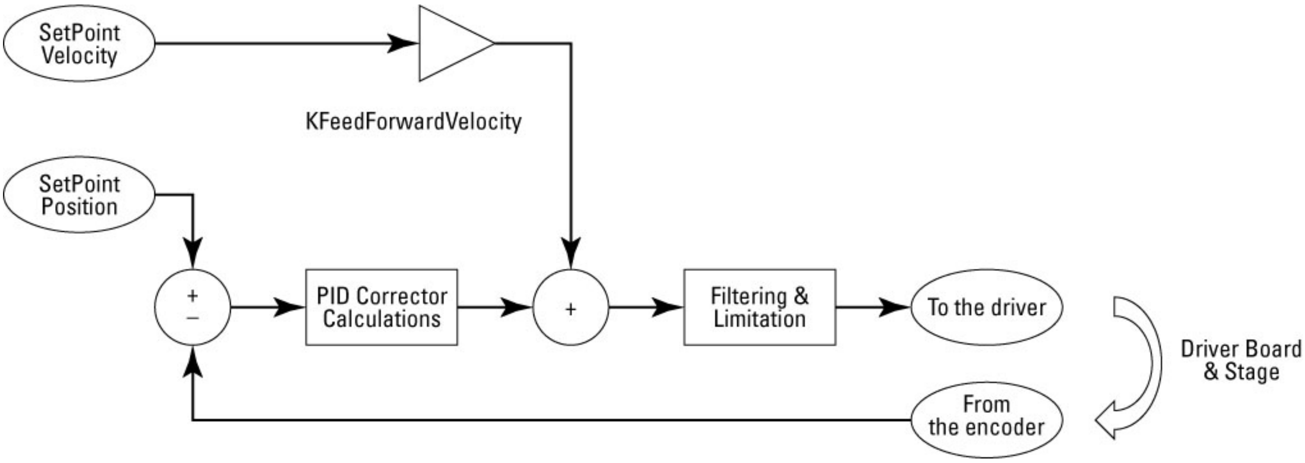


Figure 50: Corrector = PIDFFVelocity.

14.3.1.1 Parameters

FeedForward Method:

- Velocity
- KFeedForwardVelocity is a gain that can be applied to this feed forward.
- When the system is used in open loop, the PID output is not applied and the feed forward gain is set to 1 (the entire output of the controller is FF gain).

PID corrector:

- Total output of the PID is a speed (units/s), so:

  Kp is given in 1/s.

  Ki is given in 1/s^2.

  Kd has no unit.

Filtering and Limitation:

- ScalingVelocity (units/s) is the theoretical speed resulting from a 10 V input to the driver.
- VelocityLimit (units/s) is the maximum speed that can be commanded to the driver.

14.3.1.2 Basics

For a “perfect system” (no friction, all performance factors known, no following errors), a KFeedForwardVelocity value of 1 will generate the exact amount of output required to reach the TargetPosition.

The Kd parameter is generally redundant when using the speed loop of the driver and is usually set to zero, but a higher value can be used to improve the “tightness” of the speed loop.

The proportional gain Kp drives the cut-off frequency of the closed loop.

Due to the integration of the speed command in a position by the encoder, the overall gain of the proportional path at a given frequency Frq is equal to Kp/2πFrq. This gain is equal to 1 at Frq P = Kp/2π (close to the cut-off frequency).

This frequency must remain lower than the cut-off frequency of the speed loop of the driver and lower than the mechanic’s natural frequencies to maintain stability.

The integral gain Ki drives the capability of the closed loop to overcome perturbations and to limit static error.

Due to the integration of the speed command in a position by the stage encoder, the overall gain of the integral path at a given frequency Frq is:

$$\mathbf{Gain} = \frac{\mathbf{Ki}}{(2 \cdot \pi \cdot \mathbf{Frq})^2}$$

This gain is equal to one at FrqI:

$$\mathbf{FrqI} = \frac{1}{2 \cdot \pi} \cdot \sqrt{\mathbf{Ki}}$$

This frequency FrqI must typically remain lower than the frequency FrqP of the proportional path to keep the stability of the servo loop.

14.3.1.3 Methodology of Tuning PID's for PIDFFVelocity Corrector (DC motors with or without tachometer)

1. Verify the speed in open loop (adjustment done using ScalingVelocity).
2. Close the loop, set Kp, increase it to minimize following errors to the level until oscillations/vibrations start during motion, then decrease Kp slightly to cancel these oscillations.
3. Set Ki, increase it to limit static errors and improve settling time until the appearance of overshoot or oscillation conditions. Then reduce Ki slightly to eliminate these oscillations.
4. Kd is generally not needed but it can help in certain cases to improve the response when the speed loop of the driver board is not efficient enough.

NOTE

To set the corrector parameters (loop type, Ki, Kp, Kd,…), use the following functions:

- **CorrectorType = PIDFFVelocity : PositionerCorrectorPIDFFVelocitySet(…)**
- **CorrectorType = PIDFFAcceleration: PositionerCorrectorPIDFFAccelerationSet(…)**
- **CorrectorType = PIDDualFFVoltage: PositionerCorrectorPIDDualFFVoltageSet(…)**
- **CorrectorType = PIPosition: PositionerCorrectorPIPositionSet(…)"**

14.3.2 Corrector = PIDFFAcceleration

The PIDFFAcceleration must be used in association with a driver having a torque input (constant voltage gives constant acceleration), using MotorDriverInterface = AnalogAcceleration. (AnalogSin60Acceleration, AnalogSin90Acceleration, AnalogSin120Acceleration, AnalogDualSin60Acceleration, AnalogDualSin90Acceleration or AnalogDualSin120Acceleration).

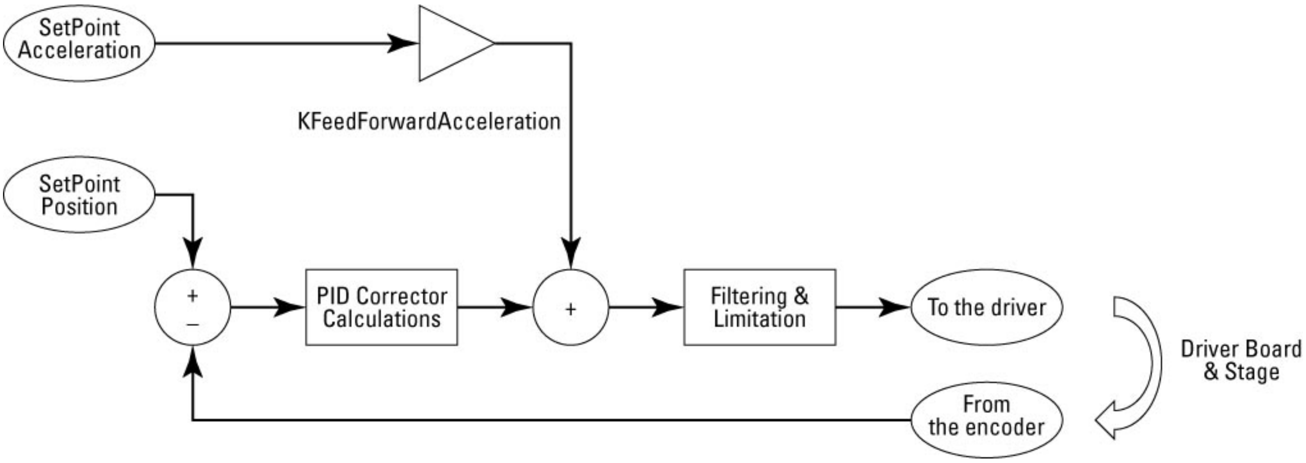


Figure 51: Corrector = PIDFFAcceleration.

14.3.2.1 Parameters

FeedForward method:

- A feed forward in acceleration is used.
- KFeedForwardAcceleration is a gain that can be applied to this feed forward.
- When the system is used in open loop, the PID output is cut and the feed forward gain is set to 1.

PID corrector:

- Output of the PID is an acceleration value in units/s^2.

  Kp is given in 1/s^2.

  Ki is given in 1/s^3.

  Kd is given in 1/s.

Filtering and Limitation:

- ScalingAcceleration (units/s^2) is the theoretical acceleration of the stage resulting from a 10 V input to the driver (depends on the stage payload).
- AccelerationLimit (units/s^2) is the maximum acceleration allowed to be commanded to the driver.

14.3.2.2 Basics

The derivative term Kd drives the cut-off frequency of the closed loop and must be adjusted first (the loop will not be stable with only Kp).

Due to the double integration of the acceleration command in a position by the stage encoder, the overall gain of the derivative path at a given frequency Frq is equal to Kd/2πFrq. This gain is equal to one at FrqD = Kd/2π (close to servo loop cut-off frequency). This frequency must remain lower than the cut-off frequency of the current loop of the driver and lower to mechanical natural frequencies to keep the stability.

The proportional gain Kp drives mainly the capability of the closed loop to overcome perturbations at medium frequencies and to limit following errors. Due to the double integration of the acceleration command in a position by the stage encoder, the overall gain of the proportional part at a given frequency Frq is:

$$\mathbf{Gain} = \frac{\mathbf{Kp}}{(2 \cdot \pi \cdot \mathbf{Frq})^2}$$

This gain is equal to one at FrqP:

$$\mathbf{FrqP} = \frac{1}{2 \cdot \pi} \cdot \sqrt{\mathbf{Kp}}$$

This frequency FrqP must remain lower than the frequency FrqD of the derivative part to keep the stability.

The integral gain Ki drives the capability of the closed loop to overcome perturbations at low frequencies and to limit static error.

Due to the double integration of the acceleration command in a position by the stage encoder, the overall gain of the integral part at a given frequency Frq is:

$$\mathbf{Gain} = \frac{\mathbf{Ki}}{(2 \cdot \pi \cdot \mathbf{Frq})^3}$$

This gain is equal to one at FrqI:

$$\mathbf{FrqI} = \frac{1}{2 \cdot \pi} \cdot \mathbf{Ki}^{\frac{1}{3}}$$

This frequency FrqI must remain lower than the frequency FrqP of the proportional part to keep the stability.

14.3.2.3 Methodology of Tuning PID's for PIDFFAcceleration Corrector (direct drive DC motors)

1. Verify the AccelerationFeedForward in open loop (adjustment done using ScalingAcceleration).

   Close the loop, set Kd, increase it to minimize following errors until vibrations appear during motion.
2. Decrease Kd to eliminate oscillations.
3. Set Kp, increase it to minimize following errors until the appearance of oscillations, decrease it to eliminate oscillations.
4. Set Ki, increase it to limit static errors and settling time until the appearance of overshoot/oscillations.

NOTE

To set the corrector parameters (loop type, Ki, Kp, Kd,…), use the following functions:

- **CorrectorType = PIDFFVelocity : PositionerCorrectorPIDFFVelocitySet(…)**
- **CorrectorType = PIDFFAcceleration: PositionerCorrectorPIDFFAccelerationSet(…)**
- **CorrectorType = PIDDualFFVoltage: PositionerCorrectorPIDDualFFVoltageSet(…)**
- **CorrectorType = PIPosition: PositionerCorrectorPIPositionSet(…)"**

14.3.3 Corrector = PIDDual FFVoltage

The PIDDualFFVoltage must be used in association with a driver having a voltage input (constant voltage gives constant motor voltage), using MotorDriverInterface = AnalogVoltage.

Can also be used in velocity or acceleration command.

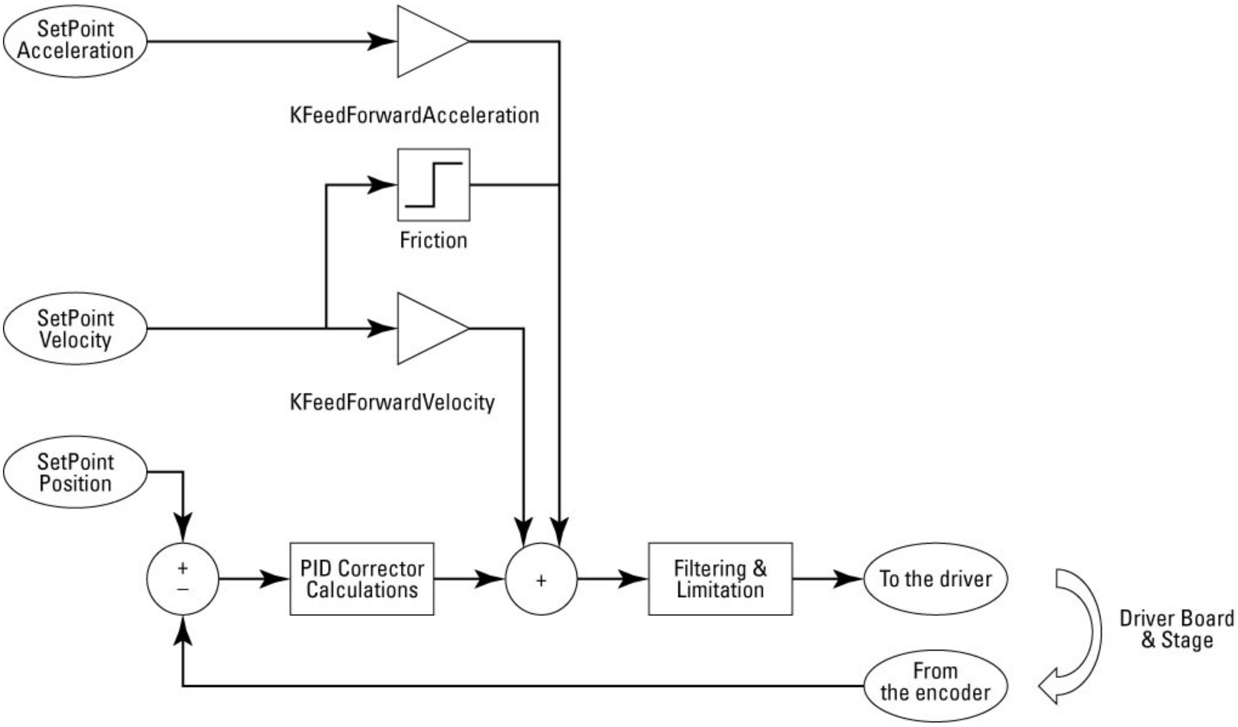


Figure 52: Corrector = PIDDual FFVoltage.

14.3.3.1 Parameters

FeedForward method:

- 3 feed forwards are used: Speed, Acceleration and Friction.
- KFeedForwardAcceleration is a gain that can be applied to the feed forward in acceleration.
- KFeedForwardVelocity is a gain that can be applied to the feed forward in velocity.
- Friction is a value which is applied with the sign of the velocity.
- When the system is used in open loop, the PID output is cut and only one feed forward in velocity is applied with the gain defined by KFeedForwardVelocityOpenLoop.

PID corrector:

- Output of the PID is a voltage.

  Kp is given in V/unit.

  Ki is given in V/unit/s.

  Kd is given in V/s/unit.

Filtering and Limitation:

- ScalingVoltage is the theoretical motor voltage resulting from a 10 V input on the driver (48 V).
- VoltageLimit (volts) is the maximum motor voltage allowed to be commanded to the driver.

Refer to the XPS-RL Configuration Wizard Document for a detailed explanation.

14.3.3.2 Basics

The PIDDualFFVoltage corrector can be seen as a mix between the PIDFFVelocity and PIDFFAcceleration correctors. It is difficult to give a precise picture of this behavior which depends a lot on the response of the stage (speed and acceleration versus motor voltage).

14.3.3.3 Methodology of Tuning PID's for PIDDualFF Corrector (DC motors with tachometers)

1. Adjust KFeedForwardVelocityOpenLoop to optimize the fidelity of the speed at high speed.
2. Close the loop using the same value for KFeedForwardVelocity, set Kp, increase it to minimize following errors until oscillations/vibrations appears during motion, decrease Kp to eliminate oscillations.
3. Set Kd, increase until oscillations/vibrations appear during motion, and decrease it to eliminate oscillations.
4. Increase Ki to cancel static error and minimize settling time until appearance of overshoot/oscillations.

14.3.4 Corrector = PIPosition

PIPosition corrector can be used with AnalogStepperPosition or AnalogPosition interface.

The AnalogPosition interface is to be used with a driver having a position input (example = piezo driver).

The AnalogStepperPosition interface is to be used with a driver having two sine and cosine current inputs (constant voltage gives constant currents in motor windings so position is constant).

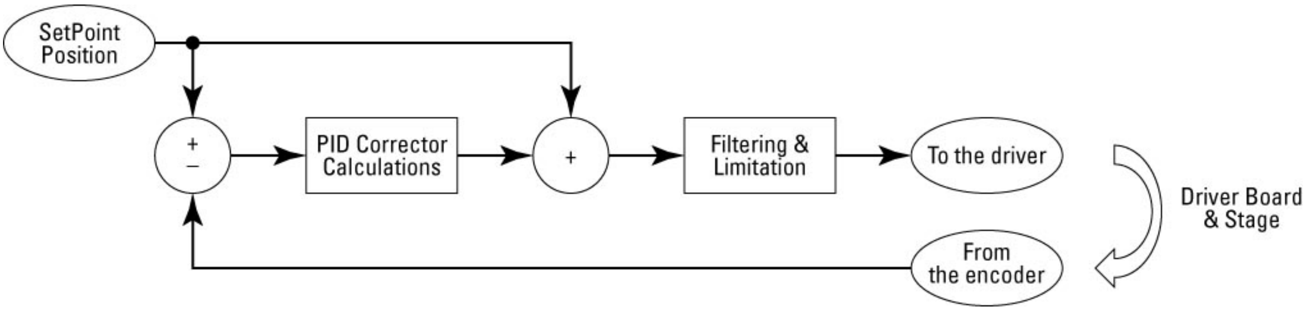


Figure 53: Corrector = PIPosition.

14.3.4.1 Parameters

FeedForward:

- One feed forward in position. No adjustable gain.
- When the system is used in open loop, the PI output is cut and the feed forward in position is applied.

PI corrector:

- Output of the PI is a position.

  Kp has no units.

  Ki is given in 1/s.

14.3.4.2 Basics & Tuning

In most cases, only Ki is needed to correct static errors.

The overall gain of the integral part of the servo loop at a given frequency Frq is:

$$\mathbf{Gain} = \frac{\mathbf{Ki}}{2 \cdot \pi \cdot \mathbf{Frq}}$$

This gain is equal to one at:

$$\mathbf{FrqI} = \frac{\mathbf{Ki}}{2 \cdot \pi}$$

15.0 Analog Encoder Calibration

This section refers only to analog sine encoder inputs. The purpose of the analog encoder interpolation feature is to improve the stage accuracy by detecting and correcting analog encoder errors such as offsets and sine to cosine amplitude differences.

Other kinds of errors can exist in the encoder such as impure sine or cosine signals. This feature will not compensate for them and will disturb the results of the calibration process.

Also, this calibration process assumes that the errors are small, i.e., less than a few percent.

Below are figures and numbers to illustrate the type of errors and their impact on accuracy.

Offset Error

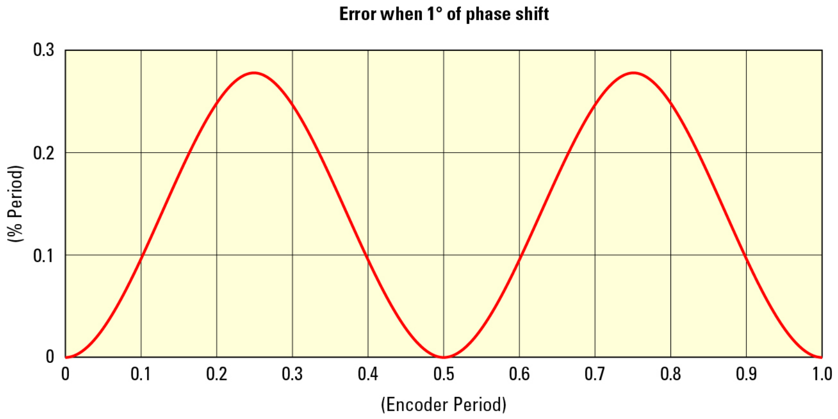


Figure 54: Offset error.

The offset error generates 0.32% interpolation error per percent offset on the sine or cosine signals. With a 20 µm scale pitch, 1% sine offset generates 63.5 nm peak to peak interpolation error.

NOTE

The real signal is not always symmetrical to 0. The offset error is defined as the difference between the signal's horizontal axis where it is symmetrical and 0.

Amplitude Mismatch

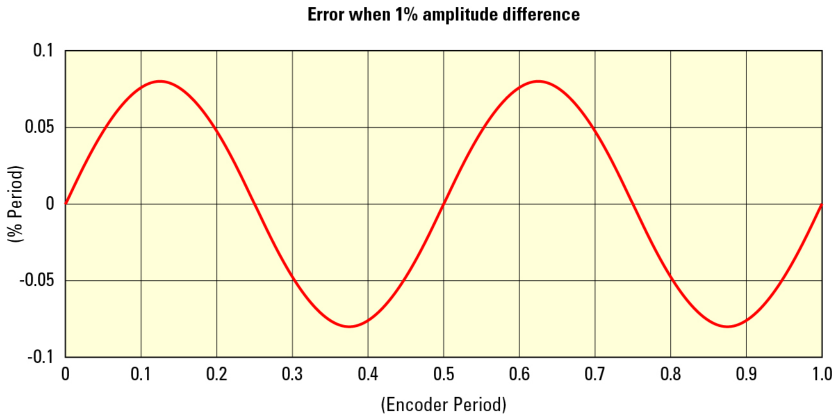


Figure 55: Amplitude mismatch.

The amplitude mismatch between sine and cosine signals generates 0.17% interpolation error per percent amplitude mismatch. With a 20 µm scale pitch, 1% amplitude mismatch generates 33 nm peak to peak interpolation error.

NOTE

Positive amplitude is the distance between the signal's maximum value and the signal axis. Negative amplitude is the distance between the signal's minimum value and the signal axis. If the positive amplitude and negative amplitude are not equal, there is amplitude mismatch.

Combined Errors

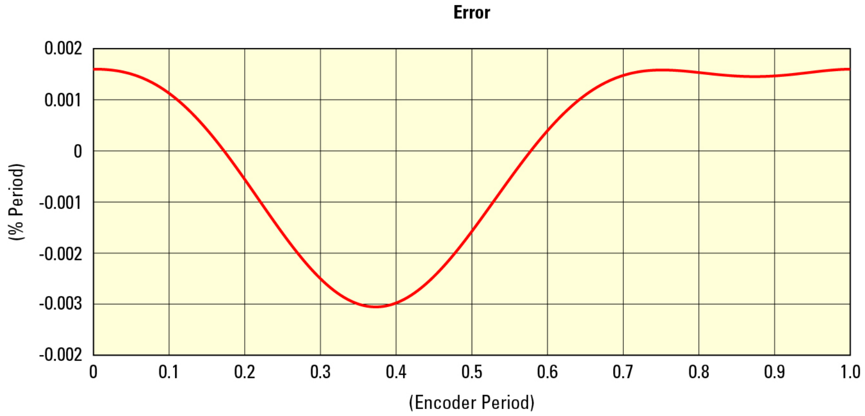


Figure 56: Combined errors.

The combination of these errors is not a simple sum but is more likely a root mean square relationship. With a 20 µm scale pitch, 1% sine offset, 1% cosine offset, 1% amplitude mismatch between sine and cosine generates 93.37 nm peak to peak error.

$$20000\sqrt{(0.32\%)^2+(0.32\%)^2+(0.164\%)^2}=96.27$$

Note that the calculated value, 96.27 nm is different than the measured 93.37 nm.

Analog encoder compensation feature

The compensation for repeatable distortions of the analog encoder input signals is always active. It uses the following parameters read from the stages.ini file. The default values are 0 for all stages:

EncoderSinusOffset = 0 Volts

EncoderCosinusOffset = 0 Volts

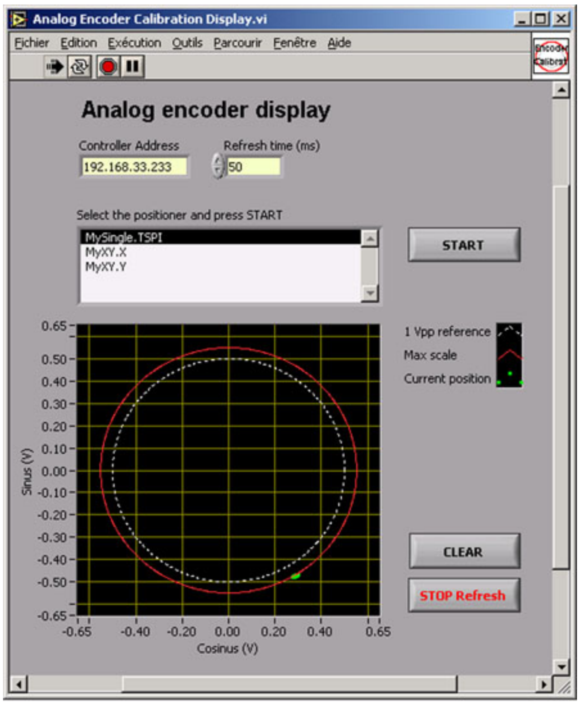
EncoderDifferentialGain = 0

EncoderPhaseCompensation = 0 deg

The function **GroupInitializeWithEncoderCalibration**() initializes the positioner and runs the encoder calibration process. During calibration, the stage moves for 25 EncoderScalePitch and the controller determines the appropriate calibration values. The controller though, will not automatically apply these values.

The function **PositionerEncoderCalibrationParametersGet**() returns the results of the last encoder calibration. To apply these values, add them manually to the appropriate section in the stages.ini file, and reboot the controller.

Stored on the XPS controller (accessible through the XPS webpage **Documentation -> Drivers & Examples -> labview -> XPS-Q8 Controller -> Examples**), embedded in Examples.llb, there is a LabVIEW application to display the current analog encoder values. The display zone matches the maximum possible amplitude of the analog signals. When they are larger than this, the AD converter will clip and the interpolation error will increase dramatically. The dotted circle represents the 1 Volt peak to peak "ideal" encoder, the red circle represents the current mean encoder settings and the green dot the current encoder value. This application uses the function PositionerEncoderAmplitudeValuesGet() for display.



Analog Encoder Calibration Display.vi

Analog encoder display

Controller Address: 192.168.33.233

Refresh time (ms): 50

Select the positioner and press START

- MySingle.TSPI
- MyXY.X
- MyXY.Y

<u>Example of the use of the functions</u>

GroupInitializeWithEncoderCalibration(MyGroup)

PositionerEncoderCalibrationParametersGet(MyGroup.MyStage)

This function returns the encoder calibration parameter values: encoder sine signal offset, encoder cosine signal offset, encoder differential gain, and encoder phase compensation. These values need to be entered in the appropriate section of the stages.ini.

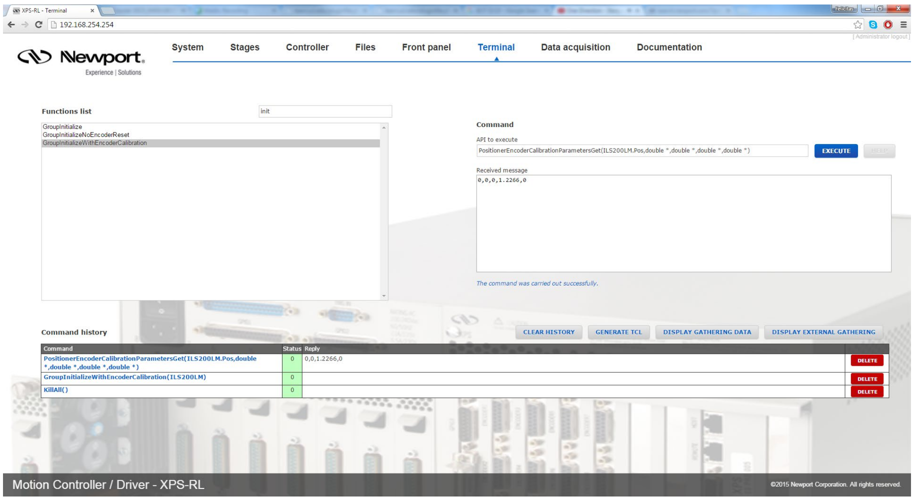
PositionerEncoderAmplitudeValuesGet(MyGroup.MyStage)

This function returns the encoder amplitude values: encoder sine signal maximum amplitude value, encoder sine signal current amplitude value, encoder cosine signal maximum amplitude value and encoder cosine signal current amplitude value.

Following is the complete process for calibrating a stage with an analog encoder interface:

Step 1

Initialize the positioner and run the calibration routine.



Functions list: init

GroupInitialize
GroupInitializeNoEncoderReset
GroupInitializeWithEncoderCalibration

Command

API to execute: PositionerEncoderCalibrationParametersGet(ILS200LM.Pos,double *,double *,double *,double *)

Received message

```
0,0,0,1.2266,0
```

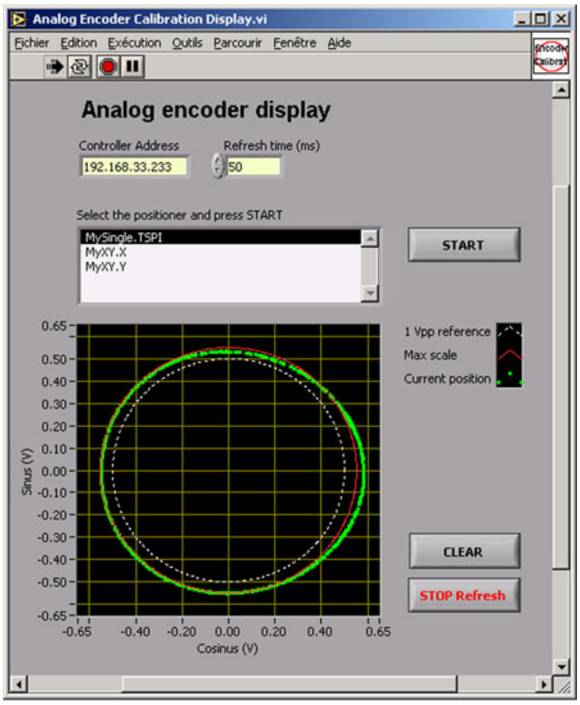
The command was carried out successfully.

Command history

Command	Status	Reply
PositionerEncoderCalibrationParametersGet(ILS200LM.Pos,double *,double *,double *,double *)	0	0,0,1.2266,0
GroupInitializeWithEncoderCalibration(ILS200LM)	0	
KillAll()	0	

Step 2

Start the AnalogEncoderCalibrationDiplay VI which is found in the ftp site. Move the positioner at very low speed.



Controller Address: 192.168.33.233

Refresh time (ms): 50

Select the positioner and press START

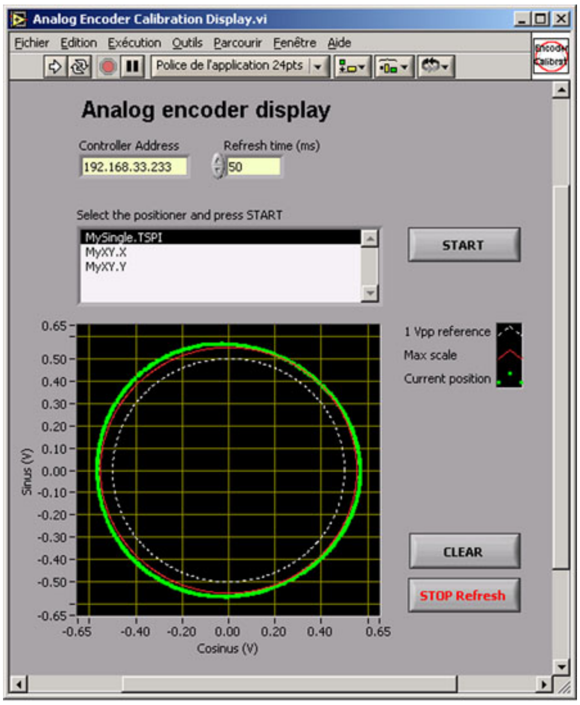
MySingle.TSPI
MyXY.X
MyXY.Y

Notice the variations between the actual (green) values and the ideal (red) values. In this case, it makes sense to apply new compensation values.

Step 3

Apply the compensation values gathered in step 1 into the stages.ini; reboot the controller.

Initialize the positioner: run the AnalogEncoderCalibrationDiplay VI; move the positioner at a very low speed.



Analog Encoder Calibration Display.vi

Analog encoder display

Controller Address: 192.168.33.233

Refresh time (ms): 50

Select the positioner and press START

MySingle.TSPI
MyXY.X
MyXY.Y

Notice the difference to the previous results. It might be necessary to run the compensation at several positions and several times to optimize the results.

16.0 Excitation Signal

16.1 Introduction

The excitation-signal function generates a typical signal (a sine, a blank noise or an echelon signal) that the controller sends to motors to excite the system. In measuring the output signal of the excited system, we can determine some system characteristics, such as the system transfer function.

16.2 How to Use the Excitation-Signal Function

The PID excitation-signal function is only available with the stages controlled in acceleration (acceleration control, ex: brushless/linear motors), velocity (velocity control) or in voltage (voltage control). It is not used with the stages controlled in position (ex: stepper motors).

The excitation-signal function **PositionerExcitationSignalSet** can be executed only when the positioner is in the "READY" state. When the excitation-signal function is in process, the positioner is in the "ExcitationSignal" state. At the end of the process, the positioner returns to the "READY" state (see group state diagram).

This function sends an excitation command to the motor over a time period. This function is allowed for "PIDFFAcceleration", "PIDFFVelocity" or "PIDDualFFVoltage" control loop. The parameters to configure are *signal type* (0:sine, 1:echelon,2:random-amplitude,3:random-pulse-width binary-amplitude, integer), *frequency* (Hz, double), *amplitude* (acceleration, velocity or voltage unit, double) and *during time* (seconds, double).

The effective functional parameters for each mode are: (Limit means AccelerationLimit, VelocityLimit or VoltageLimit) :

– Sine signal mode : *Frequency* (>=1 and <= 5000), *Amplitude* (>0 and <= Limit), *Time*(>0)

– Echelon signal mode : *Amplitude* (>0 and <= Limit, or <0 and >= -Limit), *Time* (>0).

  + During *Time* : Signal = *Amplitude*

  + End of *Time* : Signal = 0

- Random-amplitude signal mode : *Amplitude* (>0 and <= Limit), *Time*(>0), *Frequency* (>= 1 and <= 5000).

  The signal is generated with a random value at every controller base time (Tbase = 0.1 ms), then is filtered with a second order low-pass filter at the cut-off *Frequency* value.

o Random-pulse-width binary-amplitude signal mode :

  Amplitude (>0 and ≤ Limit), *Time* (>0)*, Frequency* (≥1 and ≤5000).

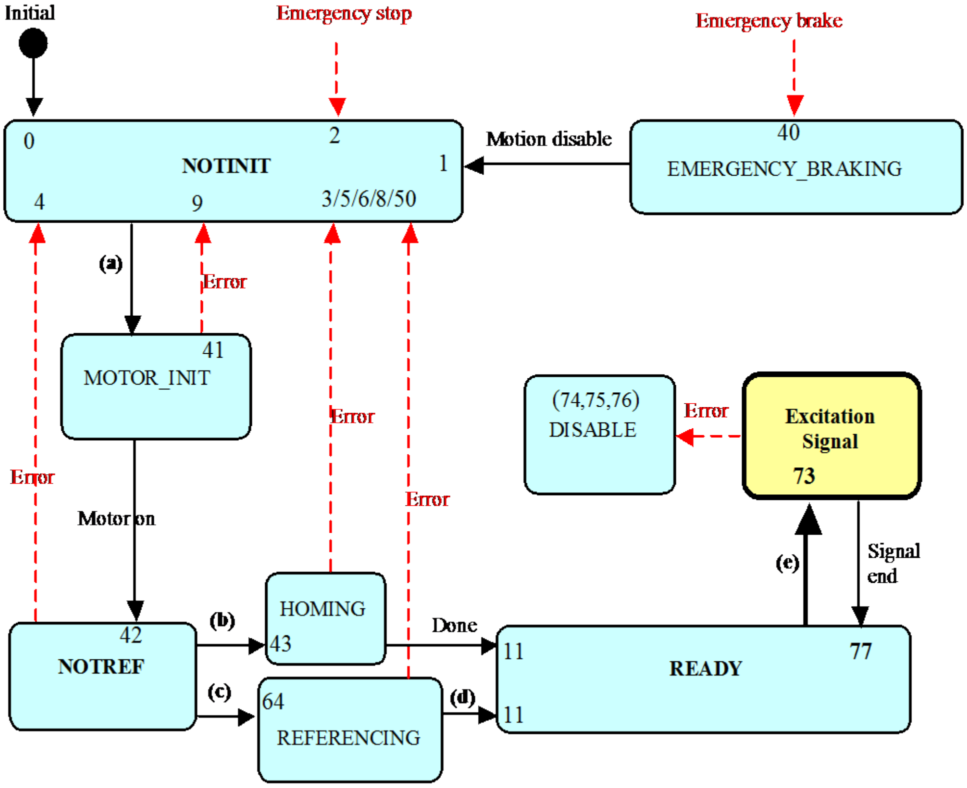
  The signal is a sequence of pulses (Signal = *Amplitude* or = 0) with the pulse randomly varied in width (multiple of Tbase).

  Frequency is the controlled system band-width (*cut-off frequency*), necessary for the PRBS (*Pseudo Random Binary Sequence*) function configuration.

The non-effective functional parameters can accept any value, the value 0 is recommended for simplicity.

The *PositionerExcitationSignalGet()* function is used to get the parameters previously used with the *PositionerExcitationSignalSet()* function.

16.3 Group State Diagram



Notes :
The numbers in the boxes represent the values of the group status.
Bold transitions are driven by function, the others are internal transitions.

(a) GroupInitialize

(b) GroupHomeSearch or **(c)** GroupReferencingStart and **(d)** GroupReferencingStop

(e) PositionerExcitationSignalSet

16.4 Function Description

- *PositionerCorrectorExcitationSignalGainGet*()
- *PositionerCorrectorExcitationSignalGainSet*()

(see ProgrammerManual.pdf)

17.0 Introduction to XPS Programming

For advanced applications and repeating tasks, it is usually better to sequence different functions in a program rather than executing them manually via the web site interface. Motion programs can be written in different ways, but essentially are distinguished between host-PC-managed and XPS-managed processes. A host-PC-managed program uses the Ethernet TCP/IP interface from a PC to control the XPS. The XPS-managed process is controlled internally by the XPS controller via a TCL script.

The chapter provides a brief introduction of the different ways of programming the XPS. This section, however, cannot address all details. For further information, refer to the software drivers manual of the XPS controller which are accessible via the XPS web site.

Host-managed processes

Host-managed processes are recommended for applications that require a lot of data management or a lot of digital communication with devices other than the XPS controller. In this case, it is more efficient to control the process from a dedicated program that runs on a PC and which sends (and receives) information to (and from) the XPS controller via the Ethernet TCP/IP communication interface. For more details, please refer to the Software .NET Manual.

XPS-managed processes (TCL)

The XPS controller is also capable of controlling processes directly using TCL scripts. TCL stands for Tool Command Language and is an open-sourced, object oriented, command language. With only a few fundamental constructs, it is very easy to learn and it is almost as powerful as C. Users of the XPS can use TCL to write complete application code with any function. The TCL script can be executed in real time but in the background, thus utilizing time that the controller does not need for servo or communication. Multiple TCL programs can run in a time sharing mode. To learn more about implementing TCL, refer to the TCL website www.tcl.tk.

The advantages of XPS-managed processes compared to host-managed processes are faster execution and better synchronization in many cases without any processing time taken by the communication link. XPS-managed processes or sub-processes are particularly valuable for repeating tasks, tasks that run in a continuous loop, and tasks that require a lot of data from the XPS controller. Examples include: anti-collision processes (processes that utilize security switches to stop motion when stages are in danger of collision); tracking, auto-focusing or alignment processes (processes that use external data inputs to control the motion); or custom initialization routines (processes that must constantly be executed during a system's use).

The XPS controller has real-time multi-tasking functionality, and with most applications there is not only a choice between a host-managed or an XPS-managed process, but also a recognition of splitting the application into the right number of sub-tasks, and defining the most efficient process for each sub-task. An efficient process design is one of the main challenges with complex and critical applications in terms of time and precision. It is recommended to spend time thinking about the proper process definition and the best approach to control the XPS using a program.

However, not all details can be addressed in this chapter.

NOTE

The controller's time and date should be used only as reference. It is not intended to be used as an absolute reference.

17.1 TCL Generator

The TCL generator provides a convenient way of generating simple executable TCL scripts. These scripts are also a good place to start for the development of more complex scripts. Note that applications that are memory intensive or require links other XPS may require a script that is external to the XPS.

The TCL generator is accessible from the terminal page of the XPS web site. Clicking the GENERATE TCL button generates a TCL script that includes the commands previously executed and listed in the Command history list. Note that the command order in the generated TCL script is chronological hence the order is the inverse of the Command history list order. The name of the generated TCL script can be specified after clicking GENERATE TCL otherwise a default name is given, New history yyyy-mm-dd.tcl. The generated TCL script is stored in the controller and can be downloaded, edited or deleted under the webpage **Files → TCL script**.

Example

This is an example using three stages, two in an XY group (named XY) and one in a SingleAxis group (named S).

The following functions were executed in the Terminal web page.

KillAll()

GroupInitialize(S)

GroupInitialize(XY)

GroupHomeSearch(S)

GroupHomeSearch(XY)

GroupMoveAbsolute(S, 25)

GroupMoveAbsolute(S, 0)

GPIODigitalSet(GPIO1.DO, 63, 0)

EventExtendedConfigurationTriggerSet(XY.XYLineArcTrajectory.Start, 0, 0, 0, 0)

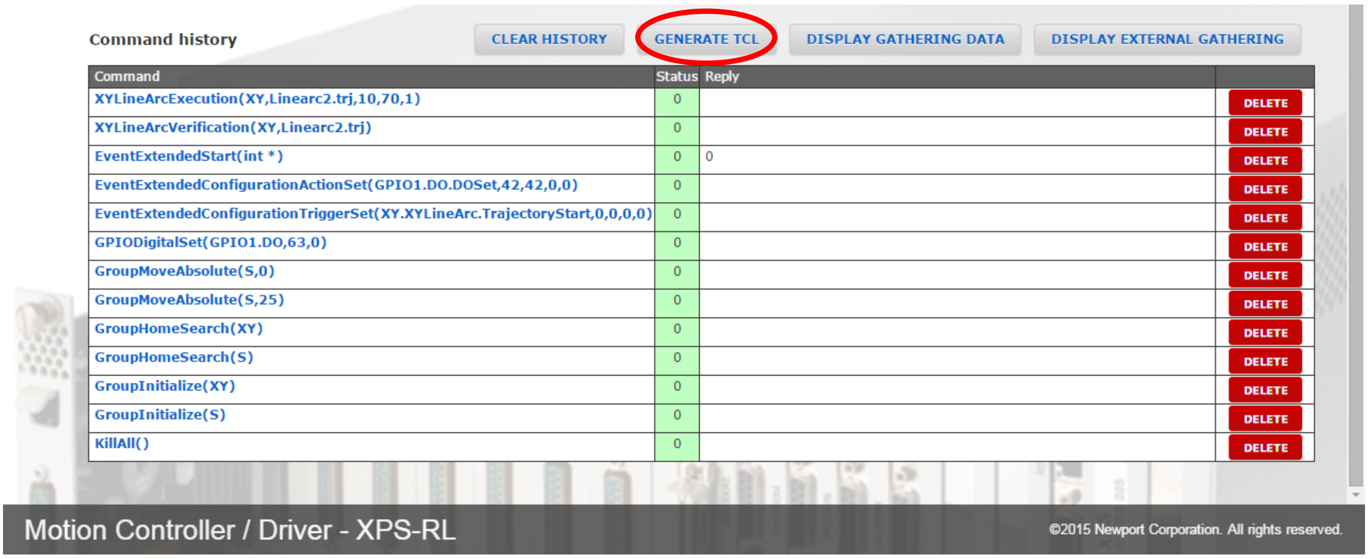
EventExtendedConfigurationActionSet(GPIO1.DO.DOSet, 42, 42, 0, 0)

EventExtendedStart()

XYLineArcVerification(XY, Linearc2.trj)

XYLineArcExecution(XY, Linearc2.trj, 10, 70, 1)

Then, the “GENERATE TCL” button is pressed to create a TCL script file. The default file name is "New history yyyy-mm-dd.tcl". When executed, that TCL file will execute all of the functions used individually in the terminal.



Command	Status	Reply	
XYLineArcExecution(XY,Linearc2.trj,10,70,1)	0		DELETE
XYLineArcVerification(XY,Linearc2.trj)	0		DELETE
EventExtendedStart(int *)	0	0	DELETE
EventExtendedConfigurationActionSet(GPIO1.DO.DOSet,42,42,0,0)	0		DELETE
EventExtendedConfigurationTriggerSet(XY.XYLineArc.TrajectoryStart,0,0,0,0)	0		DELETE
GPIODigitalSet(GPIO1.DO,63,0)	0		DELETE
GroupMoveAbsolute(S,0)	0		DELETE
GroupMoveAbsolute(S,25)	0		DELETE
GroupHomeSearch(XY)	0		DELETE
GroupHomeSearch(S)	0		DELETE
GroupInitialize(XY)	0		DELETE
GroupInitialize(S)	0		DELETE
KillAll()	0		DELETE

Motion Controller / Driver - XPS-RL
©2015 Newport Corporation. All rights reserved.

To execute the script, use the API function, TCLScriptExecute() and designate task name and parameter. Example with the default TCL script name: TCLScriptExecute(New history yyyy-mm-dd.tcl, task1, 0). Alternatively the user has the option to execute the TCL script and either block the socket or designate a priority level to the task.

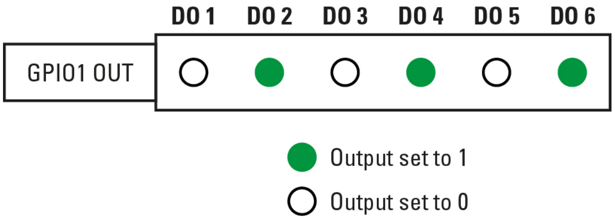
1. Execute the TCL script with the given task name (in this example task1) and block the socket until the script terminates use the API TCLScriptExecureAndWait(New history yyyy-mm-dd.tcl, task1, 0, char*).
2. Execute the TCL script with the given task name (in this example task1) at a user defined priority level: HIGH, MEDIUM or LOW use the API TCLScriptExecuteWithPriority: (New history yyyy-mm-dd.tcl, task1,HIGH, 0).

In this example, after initializing and homing both groups, the TCL script moves the single axis stage to the position of 70 units, then to the position of –70 units. It then sets all pins 1 - 6 on the digital output GPIO1 to 0.

Once checked, the line arc trajectory defined in the Linearc2.trj file gets executed with a velocity of 10 units/s and an acceleration of 70 units/s^2. When this trajectory starts, more precisely when the positioner of the X axis starts moving, the bits #2, #4 and #6 of the output GPIO1 are set to 1 (42 = 101010).

NOTE

GPIO1.DO is only available for the Basic GPIO option.



NOTE

Selecting the function TCLScriptExecute() from the terminal menu opens a drop-down list for the available TCLFileNames. However, this list is limited to 100 entries.

To kill a running script, use the API function TCLScriptKill() and input the user defined task name; example TCLScriptKill(task1). Alternatively the user can kill all running scripts by using the API function TCLScriptKillAll(). To get a list of task names, of all running TCL scripts, use the API function TCLScriptRunningListGet().

To learn more about TCL programming, refer to the TCL website.

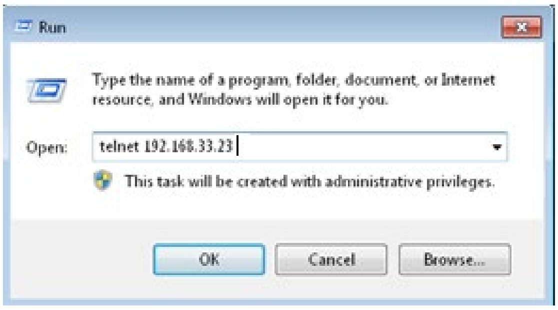
17.2 Principle of a TCL Script Redirection to a Telnet Session

17.2.1 Principle of a Tcl Script Output Redirection to a Telnet Session

Opening a Telnet session is a convenient and easy way to observe Tcl Script responses, as well as pass information to and from the XPS controller.

A telnet connection is opened with any valid login, which can be administrator, anonymous, or whatever other logins are configured.

- For windows users, click *Start* -> *Run* -> then type *telnet* + IP address (IP address is the controller address you are connecting to) as below:

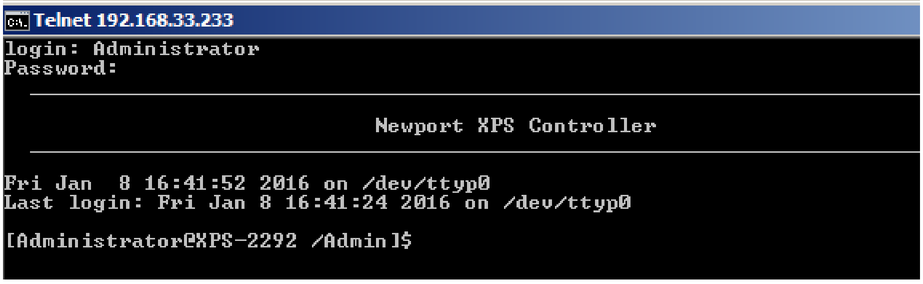


Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: telnet 192.168.33.23

This task will be created with administrative privileges.

- The telnet window is opened, type login (here login and password are “Administrator”):



```
login: Administrator
Password:

                           Newport XPS Controller

Fri Jan  8 16:41:52 2016 on /dev/ttyp0
Last login: Fri Jan 8 16:41:24 2016 on /dev/ttyp0

[Administrator@XPS-2292 /Admin]$
```

- An arrow appears which indicates that the telnet connection is ready for communication.
- Several telnet session windows can be opened concurrently.

17.2.2 Example of a Tcl Script Redirection to a telnet Session

Tcl script example

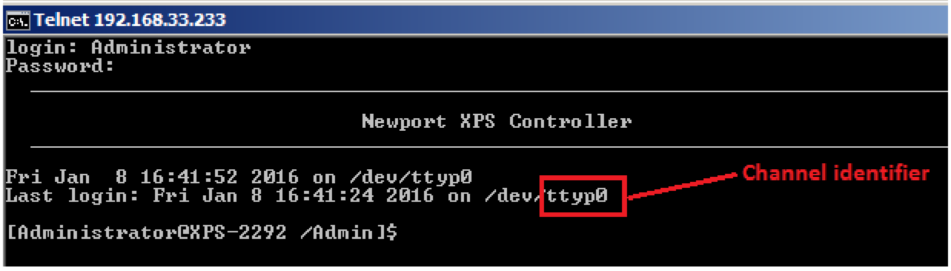
The following example shows the redirection of a Tcl script to a telnet session the telnet window displays the results of the Tcl execution (gets the firmware version).

```
# Open socket
set TimeOut 60
set code [catch "OpenConnection $TimeOut socketID"]
if {$code != 0} {
      puts stdout "OpenConnection failed => $code"
      # Force transfer to channel’s output buffer
      flush stdout
} else {
      # Get firmware version
      set code [catch "FirmwareVersionGet $socketID strVersion"]
      if {$code != 0} {
             ErrorStringGet $socketID $code strError
             puts stdout "FirmwareVersionGet Not OK => error = $code :
$strError"
             # Force transfer to channel’s output buffer
```

```
            flush stdout
        } else {
            puts stdout "Firmware Version = $strVersion"
            # Force transfer to channel's output buffer
            flush stdout
        }
        # Close TCP socket
        set code [catch "TCP_CloseSocket $socketID"]
}
```

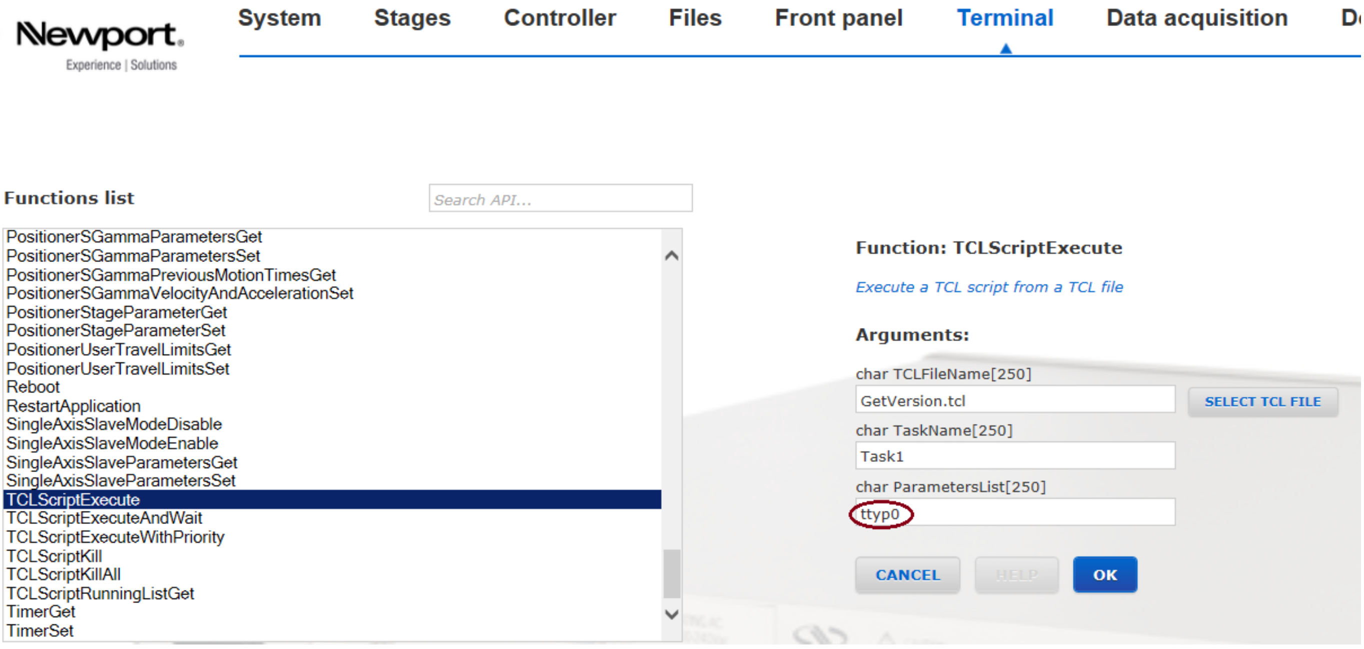
Tcl script execution

When you open a telnet session you can see the channel’s identifier as shown below:



```
Telnet 192.168.33.233
login: Administrator
Password:

                    Newport XPS Controller

Fri Jan  8 16:41:52 2016 on /dev/ttyp0
Last login: Fri Jan 8 16:41:24 2016 on /dev/ttyp0

[Administrator@XPS-2292 /Admin]$
```

The channel identifier will be used as argument for the function called from “Terminal” web page to execute the Tcl script:



Functions list

PositionerSGammaParametersGet
PositionerSGammaParametersSet
PositionerSGammaPreviousMotionTimesGet
PositionerSGammaVelocityAndAccelerationSet
PositionerStageParameterGet
PositionerStageParameterSet
PositionerUserTravelLimitsGet
PositionerUserTravelLimitsSet
Reboot
RestartApplication
SingleAxisSlaveModeDisable
SingleAxisSlaveModeEnable
SingleAxisSlaveParametersGet
SingleAxisSlaveParametersSet
TCLScriptExecute
TCLScriptExecuteAndWait
TCLScriptExecuteWithPriority
TCLScriptKill
TCLScriptKillAll
TCLScriptRunningListGet
TimerGet
TimerSet

Function: TCLScriptExecute

Execute a TCL script from a TCL file

Arguments:

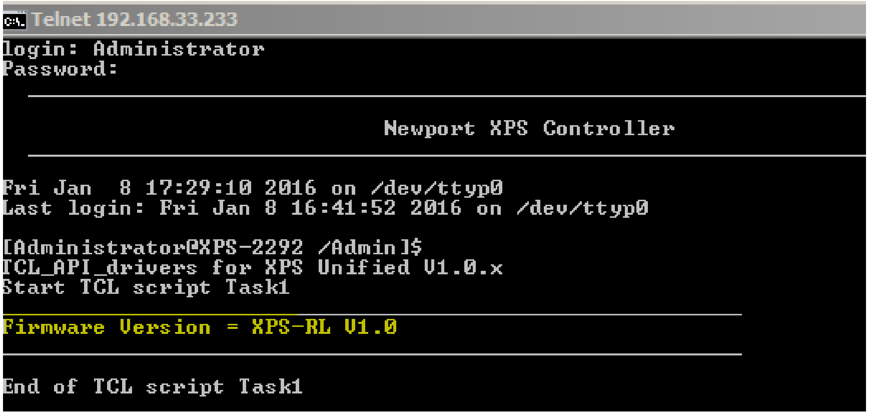
char TCLFileName[250]: GetVersion.tcl

char TaskName[250]: Task1

char ParametersList[250]: ttyp0

Tcl script execution result

The Tcl script execution result is shown on the opened telnet session window:



```
login: Administrator
Password:

                    Newport XPS Controller

Fri Jan  8 17:29:10 2016 on /dev/ttyp0
Last login: Fri Jan 8 16:41:52 2016 on /dev/ttyp0

[Administrator@XPS-2292 /Admin]$
TCL_API_drivers for XPS Unified V1.0.x
Start TCL script Task1

Firmware Version = XPS-RL V1.0

End of TCL script Task1
```

17.3 Running Processes in Parallel

TCP provides a reliable, point-to-point communication channel that client-server applications on the Internet use to communicate with each other. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server both read from and write to the socket that binds the connection.

Sockets are interfaces that can “plug into” each other over a network. Once “plugged in”, the connected programs can communicate.

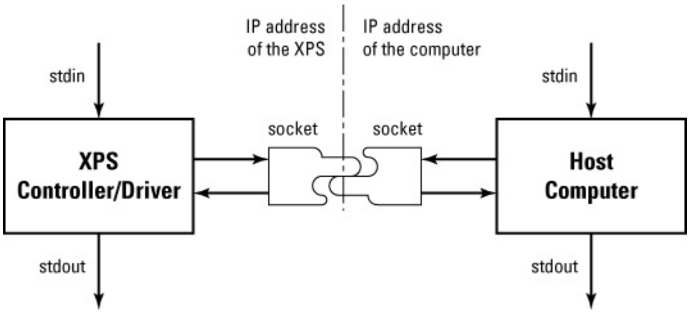


Figure 57: Running processes in parallel.

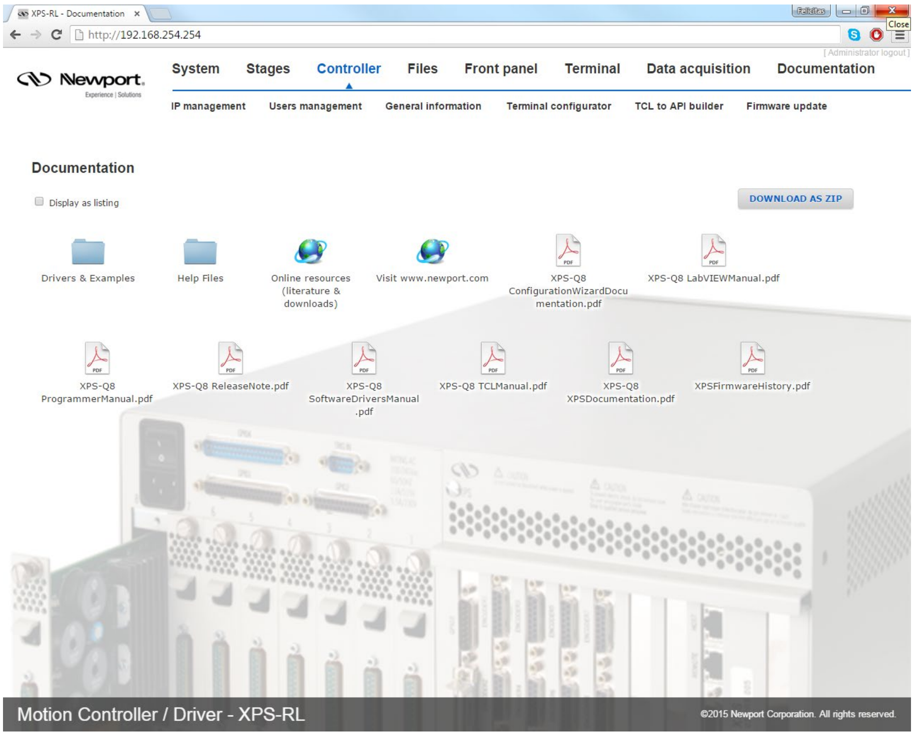
XPS uses blocking sockets. In other words, the programs/commands are “blocked” until the request for data has been satisfied. When the remote system writes data on the socket, the read operation will complete it and write the data in the received message window of the Terminal menu (‘0’ if command has been executed without error, or the error number in case of an error). That way, commands are executed sequentially since each command always waits for a response before finishing and then allowing execution of the next function. The main benefit of using this type of socket is that an execution acknowledgement is sent to the host computer with each function. In case of any error, it allows an exact diagnostic, which function has caused the error. It also allows a precise sequential process execution. On the other hand, more functions could be sent in parallel using non-blocking sockets. However, the drawback is that it is almost impossible to diagnose which function caused an error.

To execute several processes in parallel, for instance to request the current position during a motion and other data simultaneously, it is possible to communicate to the XPS controller via different sockets. The XPS controller supports a maximum number of 84 parallel opened sockets. The total number of open communication channels to the XPS controller, be it via the website, TCL scripts, a LabVIEW program, or any other program can not be larger than 84.

Users who prefer not to use blocking sockets, or whose programming languages don't support multiple sockets, such as Visual Basic versions prior to version .Net, can disable the blocking feature by setting a low TCPTimeOut value, 20 ms for instance. In this case, the XPS will unblock the last socket after the TCPTimeOut time. However, this method loses the ability to pinpoint which commands were not properly executed.

17.4 Documentation

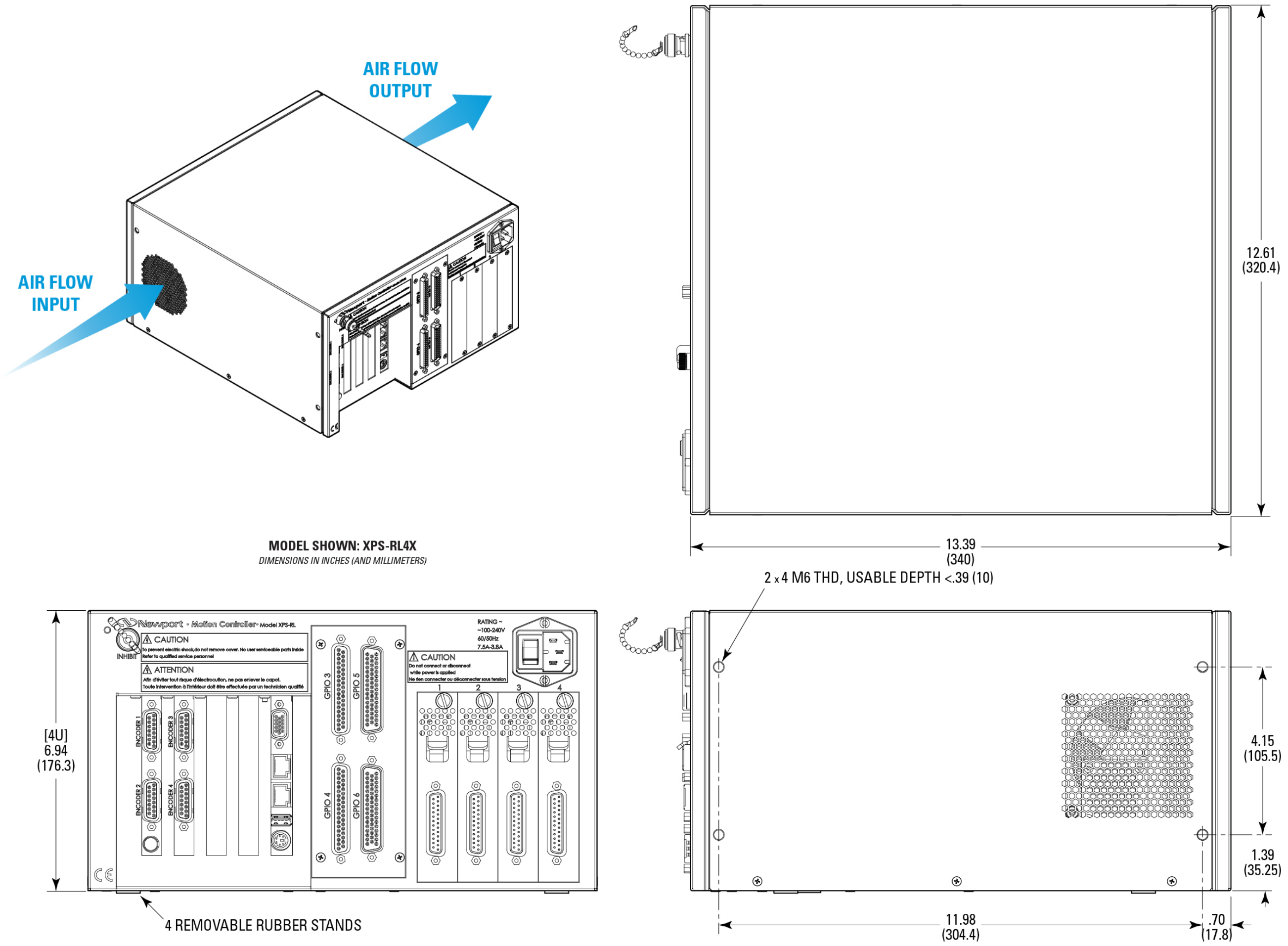
Under the webpage Documentation user's can open and download XPS-RL manuals, help files, drivers and example code.



Appendix

18.0 Appendix A: Hardware

18.1 Controller



MODEL SHOWN: XPS-RL4X

DIMENSIONS IN INCHES (AND MILLIMETERS)

Weight:	8 kg (16 lb)
Input voltage:	110–230 VAC
Input current:	7.5 A/115 V 3.8 A/230 V
Frequency:	60/50 Hz
Air flow:	227.9 ~ 59.8 CFM

18.2 Rear Panel Connectors

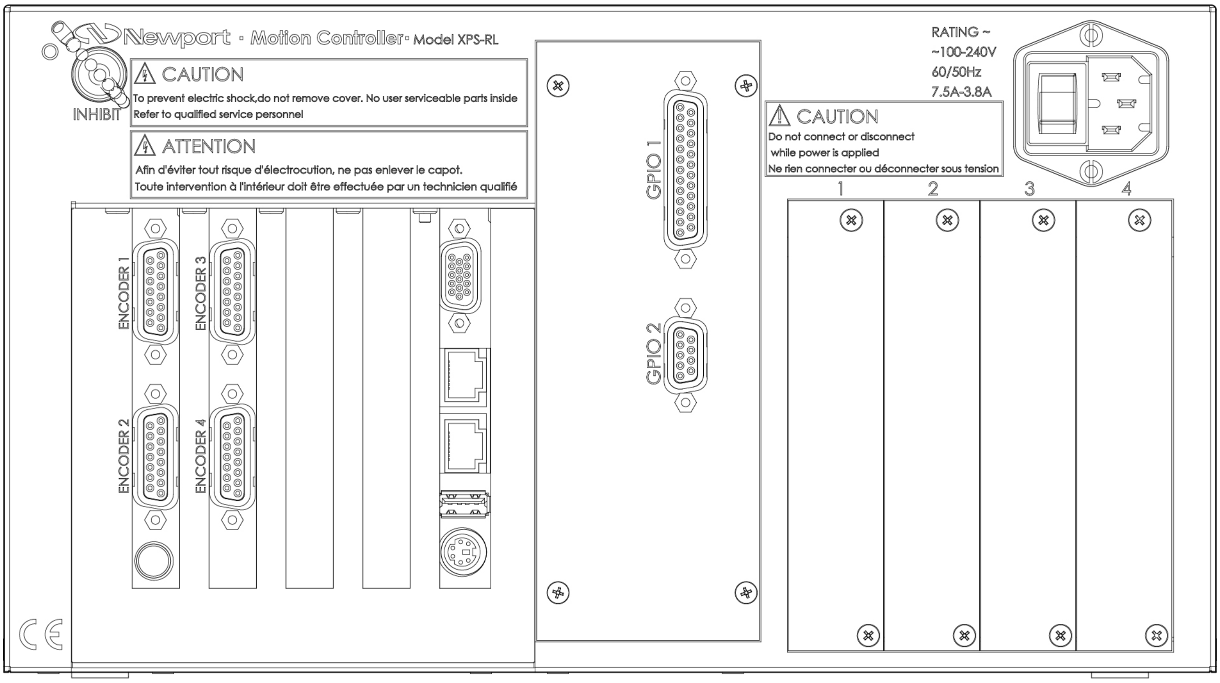

Newport · Motion Controller · Model XPS-RL
INHIBIT
CAUTION
To prevent electric shock,do not remove cover. No user serviceable parts inside
Refer to qualified service personnel
ATTENTION
Afin d'éviter tout risque d'électrocution, ne pas enlever le capot.
Toute intervention à l'intérieur doit être effectuée par un technicien qualifié
ENCODER 1
ENCODER 2
ENCODER 3
ENCODER 4
GPIO 1
GPIO 2
RATING ~
~100-240V
60/50Hz
7.5A-3.8A
CAUTION
Do not connect or disconnect
while power is applied
Ne rien connecter ou déconnecter sous tension
1
2
3
4

Figure 58: Rear Panel with Basic GPIO.

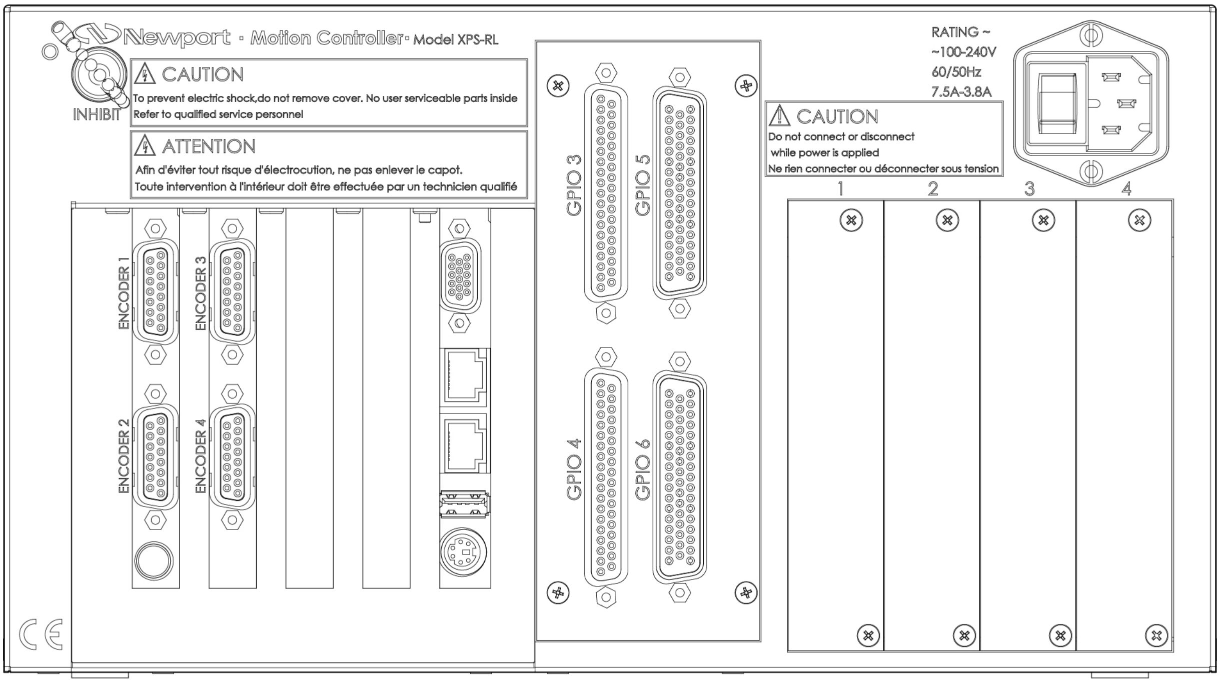

Newport · Motion Controller · Model XPS-RL
INHIBIT
CAUTION
To prevent electric shock,do not remove cover. No user serviceable parts inside
Refer to qualified service personnel
ATTENTION
Afin d'éviter tout risque d'électrocution, ne pas enlever le capot.
Toute intervention à l'intérieur doit être effectuée par un technicien qualifié
ENCODER 1
ENCODER 2
ENCODER 3
ENCODER 4
GPIO 3
GPIO 5
GPIO 4
GPIO 6
RATING ~
~100-240V
60/50Hz
7.5A-3.8A
CAUTION
Do not connect or disconnect
while power is applied
Ne rien connecter ou déconnecter sous tension
1
2
3
4

Figure 59: Rear Panel with Extended GPIO.

18.3 Environmental Requirements

Temperature range:

Storage:	-20 to +80°C
Operating:	+5 to +40°C

Relative Humidity (Non-condensing):

Storage:	10 to 95% RH
Operating:	10 to 85% RH

Altitude:

Above 1000 m, derate at 1% per 100 m. Max. 2000 m

Pollution:

Pollution Degree 2, exempt of conducting dust

19.0 Appendix B: General I/O Description

This chapter briefly describes all XPS signal types and details each of the XPS connector interfaces.

19.1 Basic GPIO – Digital I/O (GPIO1)

All digital I/Os are TTL compatible:

- All digital I/Os are not isolated, but are referenced to electrical ground (GND).
- Input levels must be between 0 V and +5 V.
- Output levels should be at least +5 V (up to 30 V absolute maximum rating with open collector outputs).
- Outputs must be pulled up to the user's external power supply (+5 V to +24 V). This external power supply must be referenced to the XPS ground (GND).

All digital I/Os are refreshed asynchronously on user requests. Therefore, digital inputs or outputs have no refresh rate.

Typical delay is 100µs due to the clock cycle and priorities made to other functions.

All digital inputs are identical: negative logic and have internal +5 V pull up resistors.

19.1.1 Digital Inputs

Parameter	Symbol	Min.	Max.	Units
Low Level Input Voltage	V_{IL}	0	0.8	V
High Level Input Voltage	V_{IH}	1.6	5	V
Input Current LOW	I_{IL}	–	-2.5	mA
Input Current HIGH	I_{IH}	–	0.4	mA

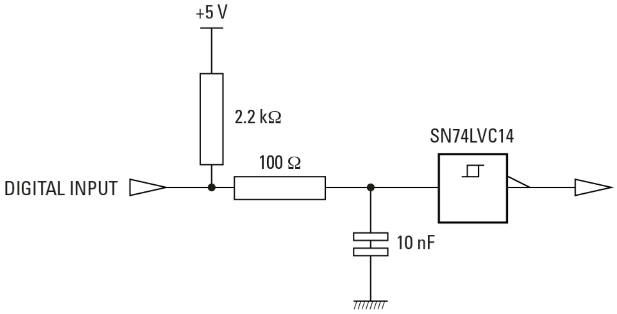


Figure 60: Digital TTL input.

GPIO1 inputs can be accessed via the GPIODigitalGet(GPIO1.DI, …) function.

19.1.2 Digital Outputs

Parameter	Symbol	Min.	Max.	Units
Low Level Output Voltage	V_{OL}	0	1	V
High Level Output Voltage	V_{OH}	2.4	30	V
Input Current LOW	I_{OL}	–	-40	mA
Input Current HIGH	I_{OH}	–	0.2	mA

All digital outputs are in negative logic (NPN open collector, 74LS06 TTL type circuit) and have no internal pull up to permit levels above +5 V.

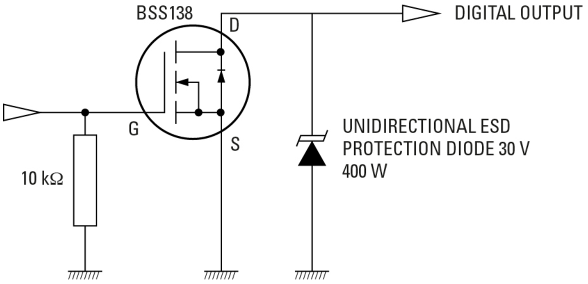
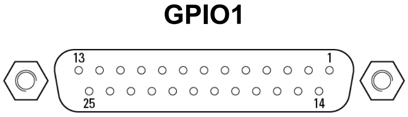


Figure 61: Open collector digital output.

GPIO1 outputs can be accessed via the GPIODigitalSet(GPIO1.DO, …) function.

19.1.3 GPIO1 Connector



NOTE

Mating connector: Male SUB-D25 with UNC4/40 lockers.

GPIO1			
Pin #	**Function**	**Pin #**	**Function**
1	GND	14	TTL Input 1
2	TTL Input 2	15	GND
3	TTL Input 3	16	TTL Input 4
4	GND	17	TTL Input 5
5	TTL Input 6	18	GND
6	Synch. Input 1 (TTL Input 7)	19	Synch. Input 2 (TTL Input 8)
7	GND	20	O.D. Output 1
8	O.D. Output 2	21	GND
9	O.D. Output 3	22	O.D. Output 4
10	GND	23	O.D. Output 5
11	O.D. Output 6	24	GND
12	Synch. Output 1 (O.D. Output 7)	25	Synch. Output 2 (O.D. Output 8)
13	+5 V		

Figure 62: GPIO1 digital I/O connector.

General Purpose Inputs Outputs GPIO1 is the main XPS digital I/O connector.

Synchronization Input 1: GPIO or external trigger input.

Synchronization Input 2: GPIO or slave controller clock input.

Synchronization Output 1: GPIO or trajectory trigger output pulses.

Synchronization Output 2: GPIO or aster controller lock output.

19.2 Basic GPIO – Analog I/O (GPIO2)

19.2.1 Analog Inputs

The 2 analog inputs have a range of ±10 V, 12 Bit resolution, and a 200 kHz 1st order low pass filter.

In all cases, the analog input values must be within the ±10 V range. The analog input impedance is typically 30 kΩ. The maximum input current is ±350 µA.

1 LSB = 20 V/4096 ≈ 4.88 mV

ADC offset and gain error are compensated.

19.2.2 Analog Outputs

The 2 analog outputs have a range of ±10 V and 12 Bit resolution. DAC offset and gain error are compensated. The DAC settling time is 5 µs and the outputs have a 200 kHz 1st order low pass filter.

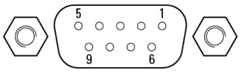
Analog outputs are voltage outputs (output current less than 1 mA), so to use them properly, they must be connected to an impedance higher than 10 kΩ.

1 LSB = 20 V/4096 ≈ 4.88 mV.

Analog outputs can be accessed via the GPIOAnalogSet(GPIO2.DACn,…) function.

19.2.3 GPIO2 Connector

GPIO2



NOTE

Mating connector: Male SUB-D9 with UNC4/40 lockers.

GPIO2			
Pin #	**Function**	**Pin #**	**Function**
1	GND	6	Analog Input 1
2	GND	7	Analog Input 2
3	GND	8	Analog Output 1
4	GND	9	Analog Output 2
5	GND		

Figure 63: GPIO2 is the basic analog I/O connector of the XPS.

General Purpose Inputs Outputs GPIO2 is the basic analog I/O connector with 2 analog inputs and 2 analog outputs.

19.3 Extended GPIO – Digital I/O (GPIO3, GPIO5, GPIO6)

All extended digital I/Os are TTL compatible:

- All digital I/Os are not isolated, but are referenced to electrical ground (GND).
- Input levels must be between 0 V and +5 V or 0 V and +12 V.
- All digital inputs are identical, except for GPIO5.DI_15 and GPIO5.DI_16 (synchronization inputs) that have no internal pull-up
- Output levels should be at least +5 V (up to 30 V absolute maximum rating with open collector outputs).
- Outputs must be pulled up to the user's external power supply (+5 V to +24 V). This external power supply must be referenced to the XPS ground (GND).

All digital I/Os are refreshed asynchronously on user requests. Therefore, digital inputs or outputs have no refresh rate.

Typical delay is 100µs due to the clock cycle and priorities made to other functions.

All digital inputs are in negative logic and have internal +5 V or +12 V pull up resistors, except for the synchronization inputs, GPIO5.DI_15 and GPIO5.DI_16, that have no internal pull-up resistors (available for the Extended GPIO).

19.3.1 Digital Inputs

Parameter	Symbol	Min.	Max.	Units
Low Level Input Voltage	V_{IL}	0	0.8	V
High Level Input Voltage	V_{IH}	1.6	5	V
Input Current LOW	I_{IL}	–	-2.5	mA
Input Current HIGH	I_{IH}	–	0.4	mA

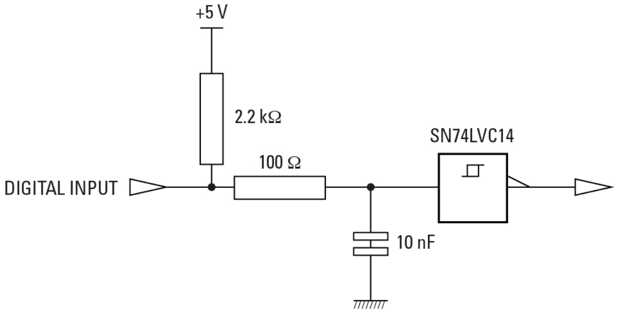
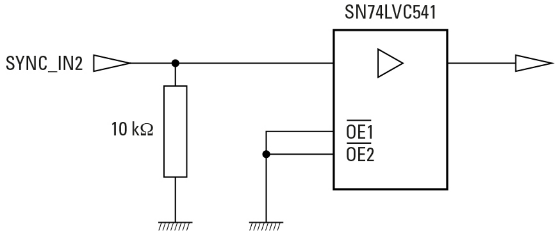


Figure 64: Digital TTL input.

GPIOn (n = 3, 5 or 6) inputs can be accessed via the GPIODigitalGet(GPIOn.DI, …) function.

GPIO5 Synchronization Inputs



Synchronization Input 1: GPIO or External Trigger Input.

Synchronization Input 2: GPIO or Slave Controller Clock Input.

19.3.2 Digital Outputs

Parameter	Symbol	Min.	Max.	Units
Low Level Output Voltage	V_{OL}	0	1	V
High Level Output Voltage	V_{OH}	2.4	30	V
Input Current LOW	I_{OL}	–	-40	mA
Input Current HIGH	I_{OH}	–	0.2	mA

All digital outputs are in negative logic (NPN open collector, 74LS06 TTL type circuit) and have no internal pull up to permit levels above +5 V.

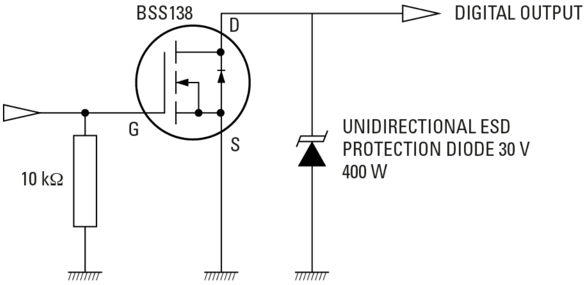
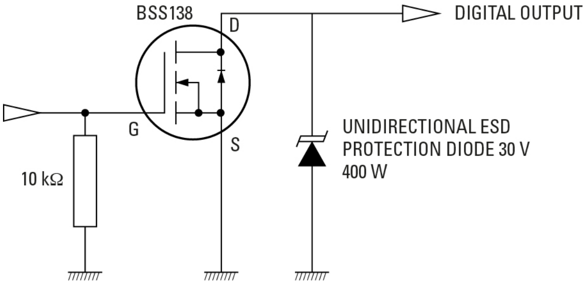


Figure 65: Open collector digital output.

GPIOn (n = 3, 5 or 6) outputs can be accessed via the GPIODigitalSet(GPIOn.DO, …) function.

GPIO5 Synchronization Outputs

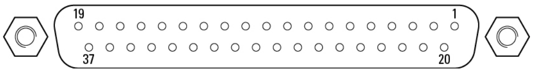


Synchronization Output 1: GPIO or Trajectory trigger output pulses.

Synchronization Output 2: GPIO or Master Controller Clock output.

19.3.3 GPIO3 Connector

GPIO3



NOTE

Mating connector: Male SUB-D37 with UNC4/40 lockers.

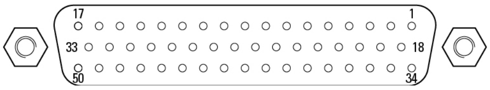
GPIO3			
Pin #	**Function**	**Pin #**	**Function**
1	N.C.	20	GND
2	+12 V (fused: max 63 mA)	21	GND
3	+5 V (fused: max 63 mA)	22	GND
4	TTL Input 1	23	GND
5	TTL Input 2	24	GND
6	TTL Input 3	25	GND
7	TTL Input 4	26	GND
8	TTL Input 5	27	GND
9	TTL Input 6	28	GND
10	TTL Input 7	29	GND
11	TTL Input 8	30	GND
12	O.D. Output 1	31	GND
13	O.D. Output 2	32	GND
14	O.D. Output 3	33	GND
15	O.D. Output 4	34	GND
16	O.D. Output 5	35	GND
17	O.D. Output 6	36	GND
18	O.D. Output 7	37	GND
19	O.D. Output 8		

Figure 66: GPIO3 digital I/O connector.

General Purpose Inputs Outputs GPIO3 is an extended digital I/O connector.

19.3.4 GPIO5 Connector

GPIO5



NOTE

Mating connector: Male SUB-D50 with UNC4/40 lockers.

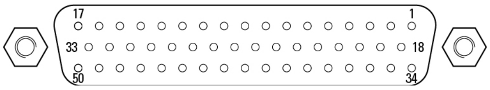
GPIO5					
Pin #	**Function**	**Pin #**	**Function**	**Pin #**	**Function**
1	+12 V (fused: max 63 mA)	18	GND	34	+5 V (fused: max 63 mA)
2	TTL Input 1	19	GND	35	O.D. Output 1
3	TTL Input 2	20	GND	36	O.D. Output 2
4	TTL Input 3	21	GND	37	O.D. Output 3
5	TTL Input 4	22	GND	38	O.D. Output 4
6	TTL Input 5	23	GND	39	O.D. Output 5
7	TTL Input 6	24	GND	40	O.D. Output 6
8	TTL Input 7	25	GND	41	O.D. Output 7
9	TTL Input 8	26	GND	42	O.D. Output 8
10	TTL Input 9	27	GND	43	O.D. Output 9
11	TTL Input 10	28	GND	44	O.D. Output 10
12	TTL Input 11	29	GND	45	O.D. Output 11
13	TTL Input 12	30	GND	46	O.D. Output 12
14	TTL Input 13	31	GND	47	O.D. Output 13
15	TTL input 14	32	GND	48	O.D. Output 14
16	Sync Input 1 (TTL Input 15)	33	GND	49	Sync Output 1 (O.D. Output 15)
17	Sync Input 2 (TTL Input 16)			50	Sync Output 2 (O.D. Output 16)

Figure 67: GPIO3 Digital I/O Connector.

General Purpose Inputs Outputs GPIO5 is an extended digital I/O connector.

19.3.5 GPIO6 Connector

GPIO6



NOTE

Mating connector: Male SUB-D50 with UNC4/40 lockers.

GPIO6					
Pin #	**Function**	**Pin #**	**Function**	**Pin #**	**Function**
1	+12 V (fused: max 63 mA)	18	GND	34	+5 V (fused: max 63 mA)
2	TTL Input 1	19	GND	35	O.D. Output 1
3	TTL Input 2	20	GND	36	O.D. Output 2
4	TTL Input 3	21	GND	37	O.D. Output 3
5	TTL Input 4	22	GND	38	O.D. Output 4
6	TTL Input 5	23	GND	39	O.D. Output 5
7	TTL Input 6	24	GND	40	O.D. Output 6
8	TTL Input 7	25	GND	41	O.D. Output 7
9	TTL Input 8	26	GND	42	O.D. Output 8
10	TTL Input 9	27	GND	43	O.D. Output 9
11	TTL Input 10	28	GND	44	O.D. Output 10
12	TTL Input 11	29	GND	45	O.D. Output 11
13	TTL Input 12	30	GND	46	O.D. Output 12
14	TTL Input 13	31	GND	47	O.D. Output 13
15	TTL Input 14	32	GND	48	O.D. Output 14
16	TTL Input 15	33	GND	49	O.D. Output 15
17	TTL Input 16			50	O.D. Output 16

Figure 68: GPIO6 digital I/O connector.

General Purpose Inputs Outputs GPIO6 is an extended digital I/O connector.

19.4 Extended GPIO – Analog I/O (GPIO4)

19.4.1 Analog Inputs

The 8 analog inputs have a range of ±10 V, 16 Bit resolution, and a 1.6 MHz 1st order low pass filter.

In all cases, the analog input values must be within the ±10 V range. The analog input impedance is typically 47 kΩ. The maximum input current is ±200 µA.

1 LSB = 20 V/65536 ≈ 0.3 mV

The ADC offset and gain errors are compensated.

19.4.2 Analog Outputs

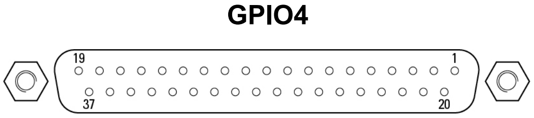
The 8 analog outputs have a configurable full scale ±5 V, ±10 V, or ±12.288 V and 16 Bit resolution. The DAC offset and gain errors are compensated. The output settling time is typically 15 µs.

Analog outputs are voltage outputs (output current less than 1 mA), so to use them properly, they must be connected to an impedance higher than 10 kΩ.

1 LSB = 20 V/65536 ≈ 0.3 mV.

Analog outputs can be accessed via the GPIOAnalogSet(GPIO4.DACn,…) function.

19.4.3 GPIO4 Connector



NOTE

Mating connector: Male SUB-D37 with UNC4/40 lockers.

GPIO4			
Pin #	**Function**	**Pin #**	**Function**
1	+12 V (fused: max 63 mA)	20	GND
2	+5 V (fused: max 63 mA)	21	GND
3	Analog Output 1	22	GND
4	Analog Output 2	23	GND
5	Analog Output 3	24	GND
6	Analog Output 4	25	GND
7	Analog Output 5	26	GND
8	Analog Output 6	27	GND
9	Analog Output 7	28	GND
10	Analog Output 8	29	GND
11	N.C.	30	GND
12	Analog Input 1	31	GND
13	Analog Input 2	32	GND
14	Analog Input 3	33	GND
15	Analog Input 4	34	GND
16	Analog Input 5	35	GND
17	Analog Input 6	36	GND
18	Analog Input 7	37	GND
19	Analog Input 8		

Figure 69: GPIO4 the extended analog I/O pin out.

General Purpose Inputs Outputs GPIO4 is the extended analog I/O connector of the XPS.

19.5 Digital Encoder Inputs (Driver Boards & DRV00P)

All digital encoder inputs are RS-422 standard compliant:

- All digital encoder signals are not isolated, but are referenced to the electrical ground (GND).
- Encoder signals must be differential pairs (using 26LS31 or MC3487 line driver type circuits). Encoder inputs have a terminating impedance of 120 Ω.
- Inputs are always routed on differential pairs. For a high level of signal integrity, we recommend using shielded twisted pairs of wires for each differential signal.
- Encoder power supply is +5 V @ 250 mA maximum (referenced to the electrical ground) and is sourced directly by the driver boards. The +5 V power supply is low noise (approximately 20 mVpp), fuse protected up to 500 mA/plug, and supplies 5.13 V without load.

19.6 Digital Servitudes (Driver Boards, DRV00P & Analog Encoders Connectors)

All servitude inputs are TTL compatible:

- All servitude inputs are not isolated, but are referenced to the electrical ground (GND).
- Input levels must be between 0 V and +5 V.

All servitude inputs are refreshed synchronously with the XPS servo rate.

All servitude inputs are identical.

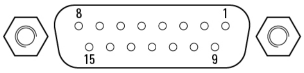
All servitude inputs expect normally closed sensors referenced to ground (input is activated if the sensor is open) and have internal 2.2 kΩ pull up resistors to the +5 V.

19.7 Analog Encoder Inputs (Analog Encoder Connectors)

The analog encoder interface complies with the Heidenhain LIF481 glass scales wiring standard.

19.7.1 Analog Encoder Connector

ENCODER 1 TO 4



NOTE

Mating connector: Male SUB-D15 with UNC4/40 lockers.

ENCODER			
Pin #	**Function**	**Pin #**	**Function**
1	Analog Cosine (0.5 Vpp)	9	Analog /Cosine (0.5 Vpp)
2	GND	10	GND
3	Analog Sine (0.5 Vpp)	11	Analog /Sine (0.5 Vpp)
4	+5 V	12	+5 V
5	N.C.	13	N.C.
6	Limit or Limit-	14	Analog Index
7	Analog /Index	15	N.C.
8	Home or Limit+		

Figure 70: Analog encoders connector.

This connector is used to receive sine/cosine encoder signals.

The sinusoidal position signals, sine and cosine, must be phase-shifted by 90° and have signal levels of approximately 1 Vpp. Each of these two signals is composed of an analog sinusoidal signal and its complement entering in a differential amplifier.

Sine, Cosine and Index signals

Analog Sine, Analog /Sine, Analog Cosine and Analog /Cosine inputs are the sine and cosine information from the encoder glass scale. Levels for these individual signals are typically 1 Vpp differential signals (maximum 1.2 Vpp) ADC Full Scale, 2.4MHz maximum input frequency and interpolation at x65536.

Analog Index and Analog /Index inputs are used to receive Index information from the encoder glass scale.

Encoder Power Supply

The +5 V provided on Encoder plugs is a dedicated power supply with the following characteristics:

- Low noise (approximately 20 mVpp)
- Fuse protected (500 mA/plug)
- 5.13 V without load

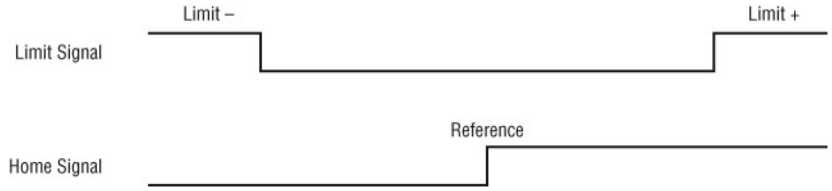


Figure 71: Limit and Home TTL input signals.

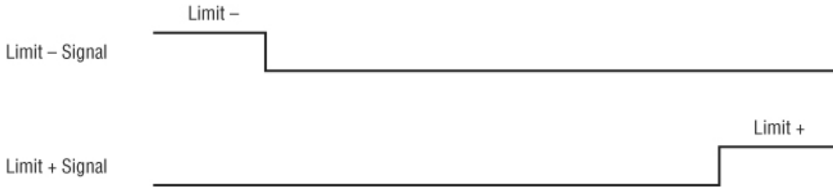


Figure 72: Limit - and Limit + TTL input signals.

20.0 Appendix C: Power Inhibit Connector

Inhibit in XPS-RL is a female BNC Connector intended for wiring a remote STOP ALL switch.

The BNC pin is connected to XPS-RL inhibit and BNC outer shell is connected to GND.

For normal operation of the XPS, the Inhibit must be connected to GND in order to work. When inhibit is open the drivers power supply is switched OFF

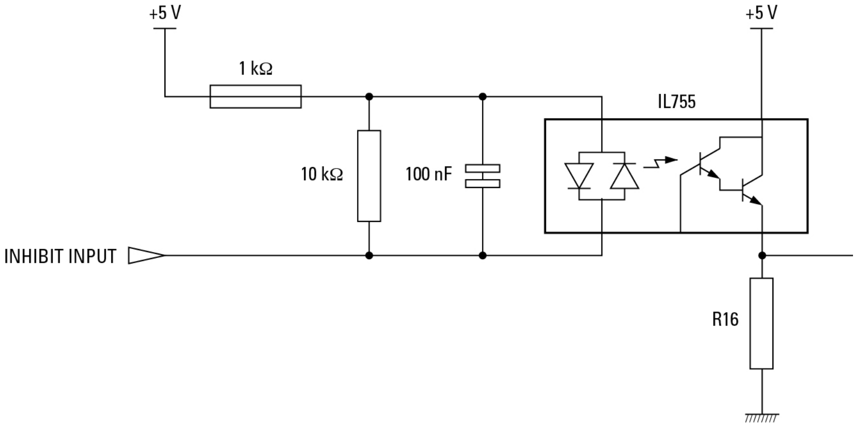
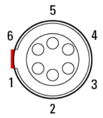


Figure 73: Inhibition connector.

21.0 Appendix D: PCO Connector

This chapter briefly describes the PCO (Position Compare Output) feature of the XPS. There are two versions of PCO for the XPS, namely Basic PCO and Extended PCO. The Extended PCO differs from the Basic PCO in that error mapping can be used to compensate position. Refer to each respective subsection below for a more detail description of each PCO type.

PCO



NOTE

Mating connector: LEMO FGG0B306CLAD52 (or 42).

PCO	
Pin #	**Function**
1	+5 V (fused: max 63 mA)
2	Axis 1 PCO Pulse/QuadA
3	Axis 1 PCO Enable/QuadB
4	Axis 2 PCO Pulse/QuadA
5	Axis 2 PCO Enable/QuadB
6	GND

Figure 74: PCO (Position Compare Output) connector.

	Time Spaced Pulses	Distance Spaced Pulses	AquadB
Minimum pulse width	35 ns	35 ns	–
Minimum pulse frequency	0.05 Hz		
Maximum pulse frequency	20 MHz	1.6 MHz (5 MHz for less than 4096 pulses)	
Interpolation	–	x256	AquadB encoder: x4 AnalogInterpolated encoder: x4, x256, x4096
Accuracy	5 ns	5 ns	5 ns
Position Window capability	YES		YES

Figure 75: Basic position compare output.

	Time Spaced Pulses	**Distance Spaced Pulses**	**AquadB**
Minimum pulse width	35 ns	35 ns	–
Minimum pulse frequency	0.05 Hz		
Maximum pulse frequency	20 MHz	1.6 MHz (5 MHz for less than 4096 pulses)	
Interpolation	–	x65536	AquadB encoder: x4 AnalogInterpolated encoder: x4, x256, x4096
Accuracy	5 ns	5 ns	5 ns
Position Window capability	YES		YES

Figure 76: Extended position compare output.

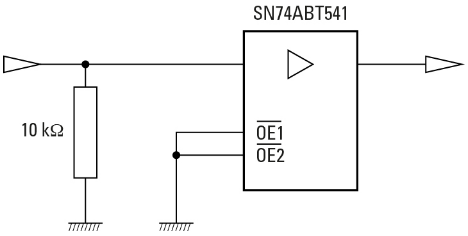


Figure 77: Output high level = 3 V minimum.

There is one PCO connector for Axis 1 and Axis 2 (PCO is not available for Axis 3 and 4). Axis #1 refers to the upper (odd) encoder plug and axis #2 refers to the lower (even) encoder plug. The signals provided on this plug depend on the configuration of the output triggers, see chapter 13.0: “Output Triggers” for more details.

The state of the enable signal is active when the stage is inside the programmed position compare window. The pulse configuration can enable the pulse polarity.

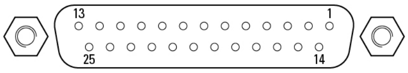
The duration of the pulse is 2 µs by default and can be modified using the function **PositionerPositionComparePulseParametersSet**(). Possible values for PCOPulseWidth are: 35 ns to 327.68 µs (5 ns resolution). Successive trigger pulses should have a minimum time lag of 625 ns (200 ns for less than 4096 pulses).

The PCO signal is SN74ABT541 line driver.

22.0 Appendix E: Motor Driver Cards

22.1 DC and Stepper Motor Driver XPS-DRV01

MOTOR DRIVER 1 TO 4



NOTE

Mating connector: Male SUB-D25 with UNC4/40 lockers.

MOTOR DRIVER					
Pin #	**DC Motor**	**Stepper Motor**	**Pin #**	**DC Motor**	**Stepper Motor**
1	Tachometer+	Phase 1	14	Shield GND	Shield GND
2	Tachometer+	Phase 1	15	Index	Index
3	Tachometer-	Phase 2	16	Limit GND	Limit GND
4	Tachometer-	Phase 2	17	+Travel Limit	+Travel Limit
5	Motor+	Phase 3	18	- Travel Limit	- Travel Limit
6	Motor+	Phase 3	19	Encoder A	Encoder A
7	Motor-	Phase 4	20	Encoder B	Encoder B
8	Motor-	Phase 4	21	+5 V	+5 V
9	Brake+	Common 3 & 4	22	Encoder GND	Encoder GND
10	N.C.	N.C.	23	Encoder /A	Encoder /A
11	Brake-	Common 1 & 2	24	Encoder /B	Encoder /B
12	N.C.	N.C.	25	/Index	/Index
13	Origin	Origin			

Figure 78: XPS-DRV01 motor driver connectors.

Motor + This output must be connected to the positive lead of the DC motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Motor - This output must be connected to the negative lead of the DC motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Ph1 This output must be connected to Winding A+ lead of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Ph2 This output must be connected to Winding A- lead of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Ph3 This output must be connected to Winding B+ lead of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Ph4 This output must be connected to Winding B- lead of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Common 3&4 This output must be connected to the center tab of Winding B of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Common 1&2 This output must be connected to the center tab of Winding A of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

+ Travel limit This input is pulled-up to +5 V with a 2.2 kΩ resistor by the controller and represents the stage positive direction hardware travel limit.

- Travel limit This input is pulled-up to +5 V with a 2.2 kΩ resistor by the controller and represents the stage negative direction hardware travel limit.

Encoder A & /A These A and /A inputs are differential inputs. Signals are compliant with RS422 electrical standard and are received with a 26LS32 differential line receiver. A resistor of 120 Ω adapts the input impedance. The A and /A encoder signals originate from the stage position feedback circuitry and are used for position tracking.

Encoder B and /B These B and /B inputs are differential inputs. Signals are compliant with RS-422 electrical standard and are received with a 26LS32 differential line receiver. A resistor of 120 Ω adapts the input impedance. The B and /B encoder signals originate from the stage position feedback circuitry and are used for position tracking.

Index & /Index These Index and /Index inputs are differential inputs. Signals are compliant with RS422 electrical standard and are received with a 26LS32 differential line receiver. A resistor of 120 Ω adapts the input impedance. The Index and /Index signals originate from the stage and are used for homing the stage to a repeatable location.

Encoder ground Ground reference for encoder feedback.

Origin This input is pulled-up to +5 V with a 2.2 kΩ resistor by the controller. The Origin signal originates from the stage and is used for homing the stage to a repeatable location.

+5 V (DRV01: 250 mA Maximum) A +5 V DC supply is available from the driver. This supply is provided for stage home, index, travel limit, and encoder feedback circuitry.

Limit ground Ground for stage travel limit signals. Limit ground is combined with digital ground at the controller side.

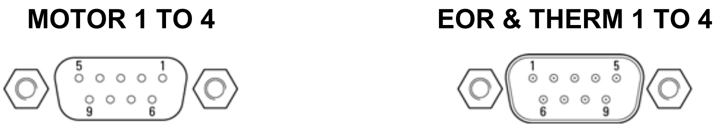
Shield GND Motor cable shield ground.

Brake + (available only on DRVM board) Voltage command (24 V or 48 V: strap on the driver board) to drive the brake.

Brake – (available only on DRVM board) Reference of the above voltage command.

Tachometer + & Tachometer – These inputs are used to receive tachometer voltage information. This voltage depends on the output voltage rating of the employed tachometer.

22.2 Three phase AC Brushless Driver XPS-DRV02



NOTE

Mating connectors:

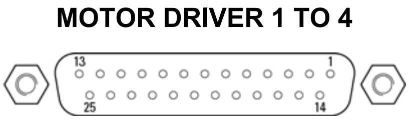
Male SUB-D9 **Female SUB-D9**

with UNC4/40 lockers.

MOTOR DRIVER			SERVITUDES	
Pin #	**Function**		**Pin #**	**Function**
1	Phase U		1	+Travel Limit
2	Phase U		2	-Travel Limit
3	Phase V		3	Origin
4	Phase V		4	N.C.
5	Thermistor+		5	+5 V
6	Phase W		6	Thermistor+
7	Phase W		7	GND
8	GND		8	Thermistor-
9	Thermistor-		9	GND

Figure 79: XPS-DRV02 motor driver connectors.
The stage thermistor can be connected to either connector.

22.3 DC Motor Driver XPS-DRV03



NOTE

Mating connector: Male SUB-D25 with UNC4/40 lockers.

MOTOR DRIVER			
Pin #	**Function**	**Pin #**	**Function**
1	Tachometer+	14	Shield GND
2	Tachometer+	15	Index
3	Tachometer-	16	Limit GND
4	Tachometer-	17	+Travel Limit
5	Motor+	18	- Travel Limit
6	Motor+	19	Encoder A
7	Motor-	20	Encoder B
8	Motor-	21	+5 V
9	N.C.	22	Encoder GND
10	N.C.	23	Encoder /A
11	N.C.	24	Encoder /B
12	N.C.	25	/Index
13	Origin		

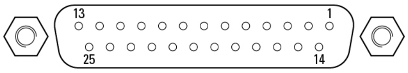
Figure 80: XPS-DRV03 motor driver connectors.

22.4 Pass-Through Board XPS-DRV00P

WARNING

The Pass-through board connector replaces the motor interface connector only if the axis is connected to an external motor driver.

PASS-THROUGH BOARD



NOTE

Mating connector: Male SUB-D25 with UNC4/40 lockers.

PASS-THROUGH			
Pin #	**Function**	**Pin #**	**Function**
1	Reserved	14	Reserved
2	+5 V	15	Inhibition Output
3	Origin Input	16	Reserved
4	-Travel Limit Input	17	Reserved
5	+Travel Limit Input	18	Reserved
6	Main Fault Input	19	Encoder /A Input
7	Encoder A Input	20	Encoder /B Input
8	Encoder B Input	21	/Index Input
9	Index Input	22	Reserved
10	Pulse/Pulse+ Output	23	GND
11	Direction/Pulse- Output	24	N.C.
12	Analog A Output	25	GND
13	Analog B Output		

Figure 81: DRV00P pass-through connector.

Analog A output and Analog B output have 16 bit resolution at ±10 V output. These signals are used to command an external driver.

22.5 Piezoelectric Stages Driver XPS-DRVP1

The XPS-DRVP1 driver board is specially designed to drive Newport stages motorized by piezoelectric actuators of the following series:

NPA, NPM, NPO, NPX, NPXY, NPXY and PSM.

Refer to the XPS-DRVP1 User's Manual for operation instructions.

23.0 Appendix F: Configuration Wizard

This chapter refers to the configuration of the XPS motion controller and drivers to motors and stages that are not included in the XPS general stage database, e.g. non-Newport stages. The chapter will present all possible configurations of the XPS controller with regards to drive and control capabilities.

The XPS controller uses two configuration files, named “system.ini” and “stages.ini”. The files are stored on the XPS controller and are accessible through the XPS webpage **Files → Configuration files**. These configuration files are read during the boot sequence of the controller. The “system.ini” file specifies the system configuration and the configured motion groups. The “stages.ini” file defines the parameters for all positioners.

The aim of this chapter is to provide a better understanding of the drive and control capabilities of the XPS controller and possible settings in the “stages.ini” file. It is important that users have a working understanding of the structure of this file, because it may be necessary to make modifications to the default parameters to access all features of the XPS controller. In order to configure the XPS controller to drive non-Newport stages, it is important that users have an in-depth understanding of this file and the meaning of the included entries.

In each subsection we provide a detailed definition of each parameter, the physical meaning and one example to set it. Since there are many options available for configuration, not all strategies and methods are detailed. Especially for the definition of the tuning parameters (PID, filters, etc.). Please refer to supporting literature for an in-depth treatment of tuning.

IMPORTANT NOTE ABOUT THE UNITS

The XPS controller accepts any dimension for the position unit such as: mm, inch, µm, deg, rad, etc. In this documentation, the generic term “unit” is used for the position unit. This generic unit is carried forward into units that reference the position unit, for example speed and acceleration would carry units such as: units/s or units/s². The physical dimension assignment of the position unit for closed-loop systems is done by *stage displacement per encoder count* as part of the parameters of the *position encoder interface.* For open-loop systems the physical dimension assignment is done as part of the parameter settings for the driver command interface; examples include: *stage displacement per motor full step* or *command voltage at minimum target position.* It is important to note that the position unit in the configuration files will determine the values of all derived parameters. Therefore the choice of the position unit will impact most parameters.

23.1 Webpage Description

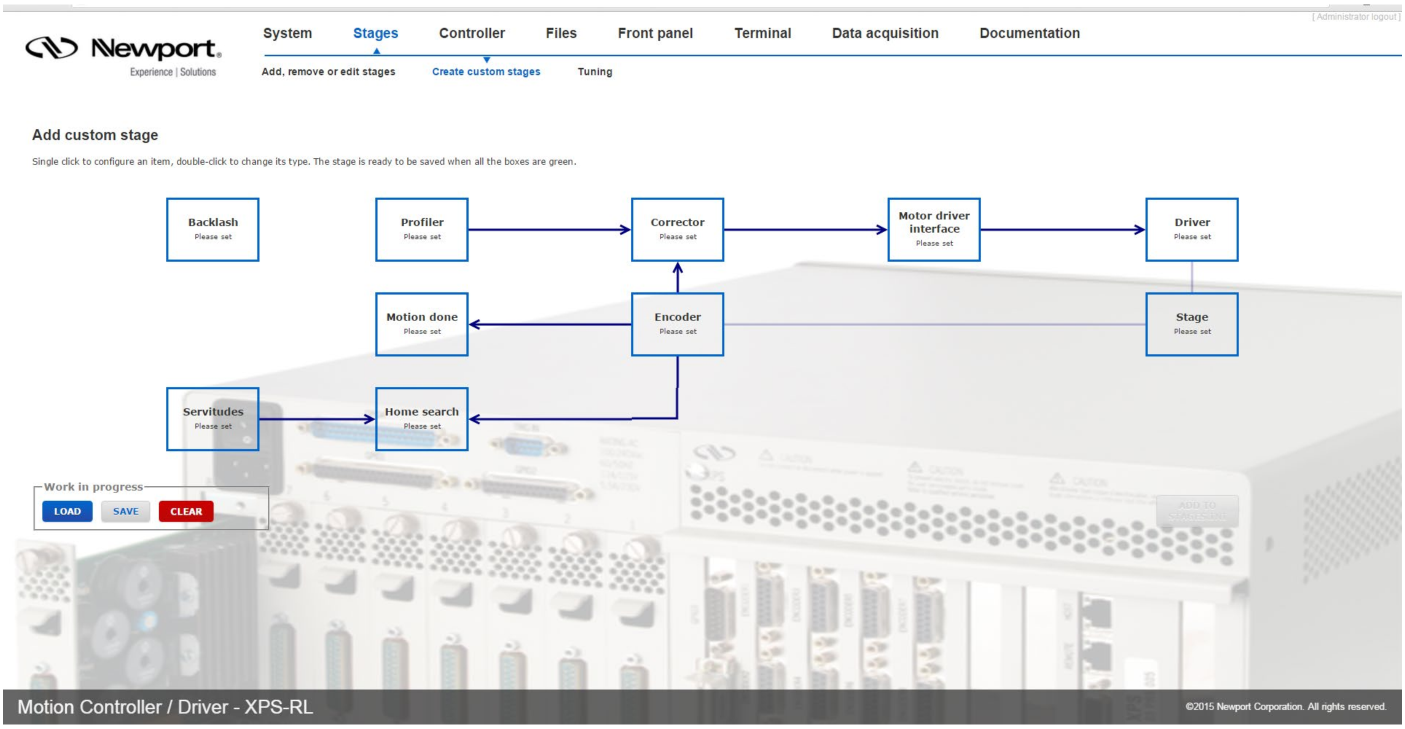
The integrated web tool, **Stages → Create custom stages,** is accessible when logged in as administrator. This web tool is designed to help users configure the XPS controller for motors and stages that are not included in the XPS general stage data base such as stages not manufactured by Newport. The tool generates a new entry in the customer’s stage database, *stages.ini*, which is stored on the controller and is accessible through the webpage **Files → Configuration files**.

To generate a new stage configuration using the web tool, users must complete an entry for each configuration category, following the arrow sequence where applicable. The software dynamically updates the next category list based on compatibility to prior selections so that only correct settings for the determined configuration are available. This avoids configurations that are not supported by the XPS controller.

NOTE

This web tool cannot compensate for configurations that are not compatible with the connected hardware such as stages with external amplifiers (example using XPS-DRV00P drive).

23.1.1 First Level Parameters



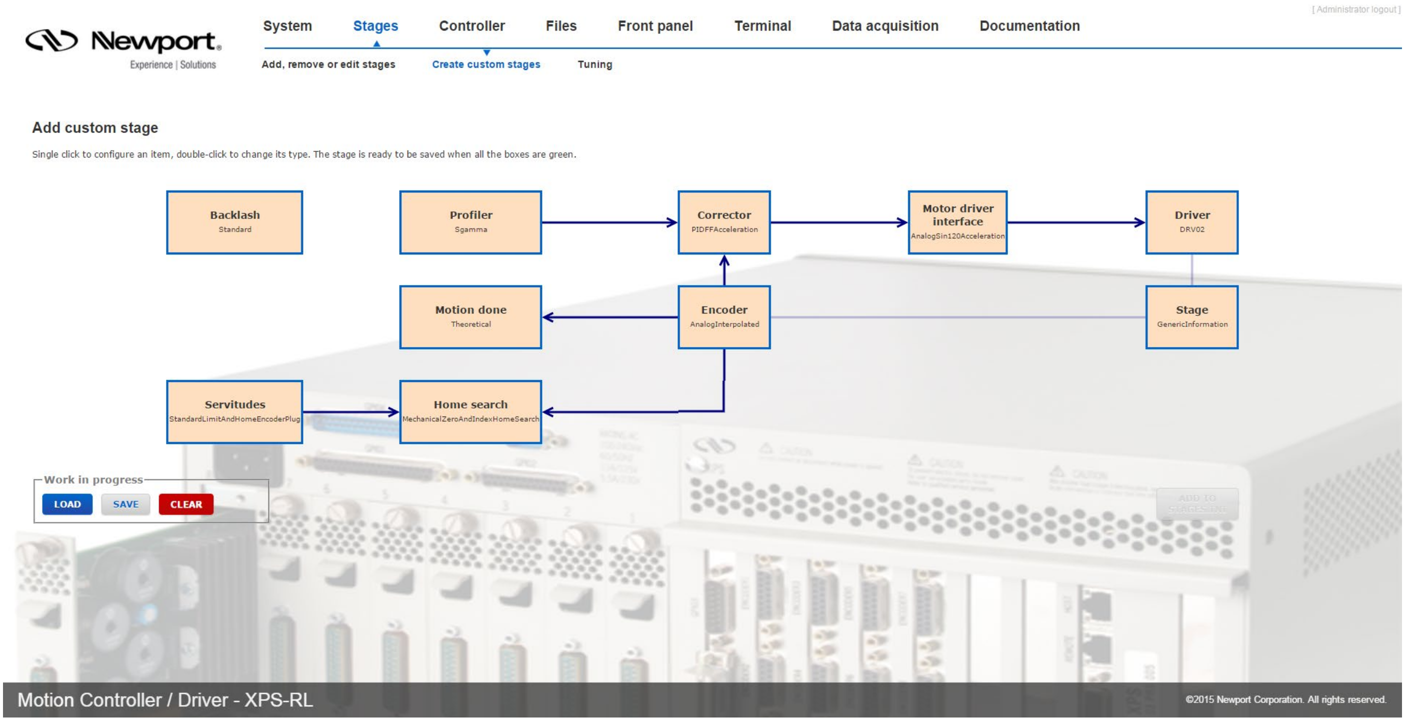
Add custom stage

Single click to configure an item, double-click to change its type. The stage is ready to be saved when all the boxes are green.

Backlash: Please set
Profiler: Please set
Corrector: Please set
Motor driver interface: Please set
Driver: Please set
Motion done: Please set
Encoder: Please set
Stage: Please set
Servitudes: Please set
Home search: Please set

The first level settings displayed above outline the stage configuration. These settings must be set following the arrows such as defining **Encoder** before **Corrector**, **Motion done** or **Home search**. To enter the first level settings for a configuration category, click on the category to get a list of the available options. These settings are presented in the preceding sections.

Once the first level of settings is entered, the configuration category changes to a light orange color.



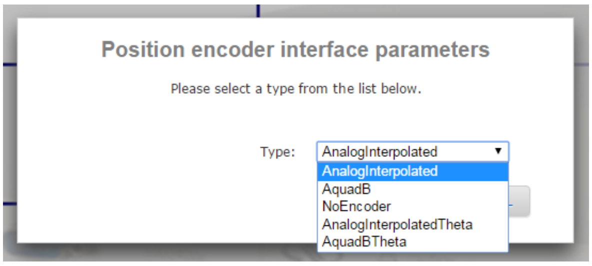
Add custom stage

Single click to configure an item, double-click to change its type. The stage is ready to be saved when all the boxes are green.

Backlash: Standard
Profiler: Sgamma
Corrector: PIDFFAcceleration
Motor driver interface: AnalogSin120Acceleration
Driver: DRV02
Motion done: Theoretical
Encoder: AnalogInterpolated
Stage: GenericInformation
Servitudes: StandardLimitAndHomeEncoderPlug
Home search: MechanicalZeroAndIndexHomeSearch

23.1.2 Second Level Parameters

Once the first level setting for a configuration category is selected, click on the category again and then the second level settings become available. The second level settings define all the details for the configuration category. These settings can be done in any order. A new page opens prompting the parameter values for the setting. The type and number of parameters in each of these screens will depend on the first level setting. Define all settings and press the OK button to apply them.



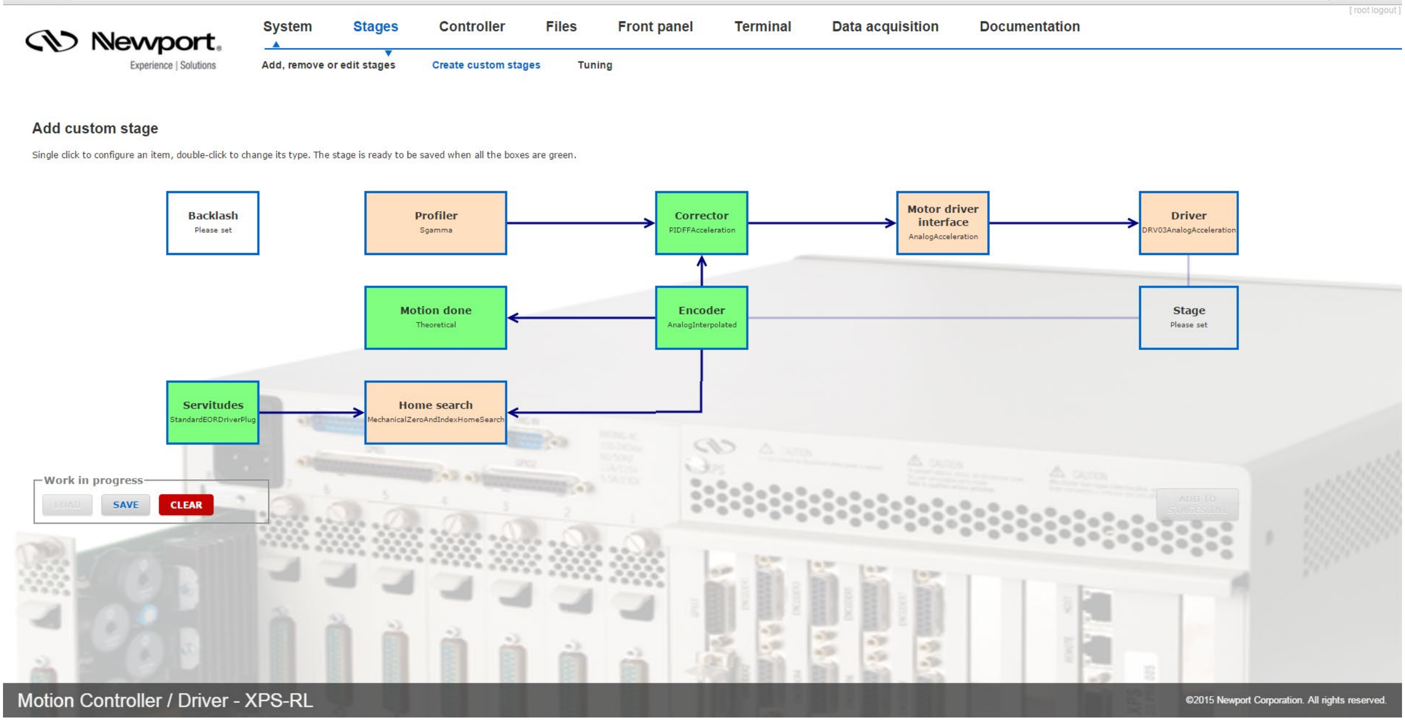
Position encoder interface parameters

Please select a type from the list below.

Type: AnalogInterpolated

(Left) First level settings for Encoder (Right) Second level settings for Encoder after selecting first level setting AnalogIntergolated.

Once the second level settings for a category is set, the category is complete and changes to green color.



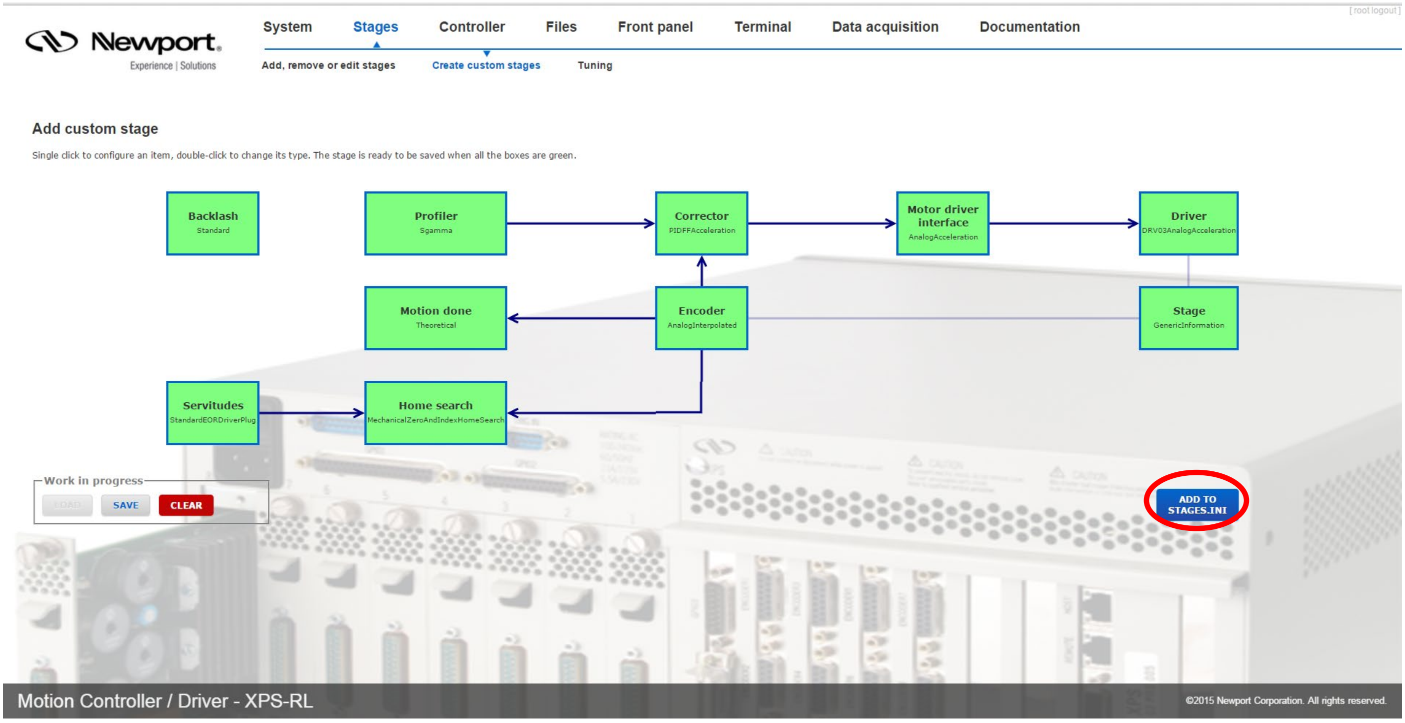
Add custom stage

Single click to configure an item, double-click to change its type. The stage is ready to be saved when all the boxes are green.

Backlash: Please set
Profiler: Sgamma
Corrector: PIDFFAcceleration
Motor driver interface: AnalogAcceleration
Driver: DRV03AnalogAcceleration
Motion done: Theoretical
Encoder: AnalogInterpolated
Stage: Please set
Servitudes: StandardEORDriverPlug
Home search: MechanicalZeroAndIndexHomeSearch

Work in progress
LOAD
SAVE
CLEAR
ADD TO STAGES.INI
Motion Controller / Driver - XPS-RL
©2015 Newport Corporation. All rights reserved.

When all second level settings are complete and green, the stage configuration parameters are set. An "ADD TO STAGE.INI" button appears in the lower right hand corner of the screen.



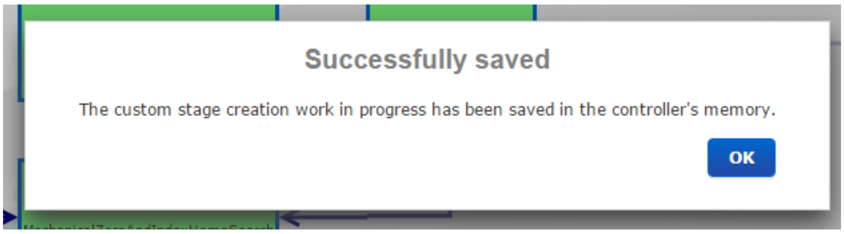
Add custom stage

Single click to configure an item, double-click to change its type. The stage is ready to be saved when all the boxes are green.

23.1.3 Save and Run the Stage

Save Option 1

Users can save their progress at any time by click on the save button in the lower left hand corner of the screen. A screen appears confirming the progress is stored to the controller's memory. From here users can move to other webpages and return to **Stages → Create custom stages** to continue the configuration wizard process.

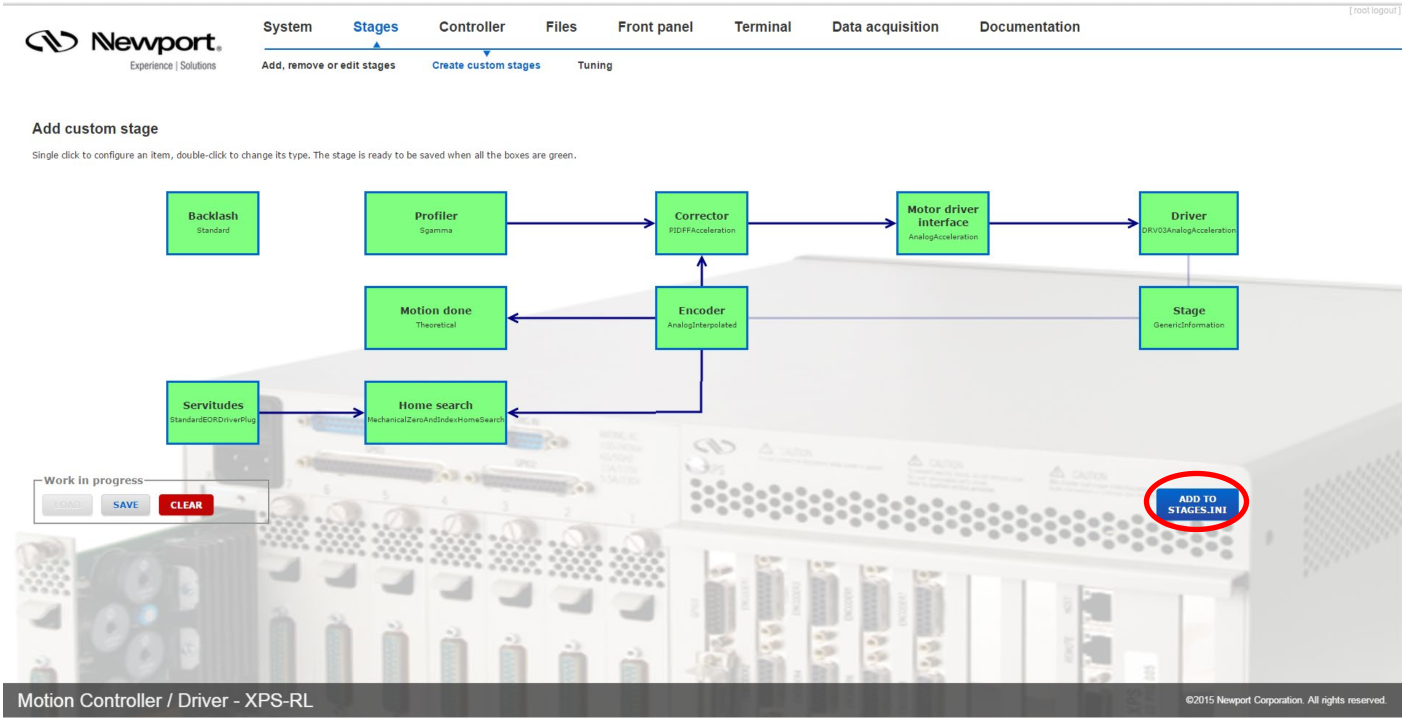


Successfully saved

The custom stage creation work in progress has been saved in the controller's memory.

Save Option 2

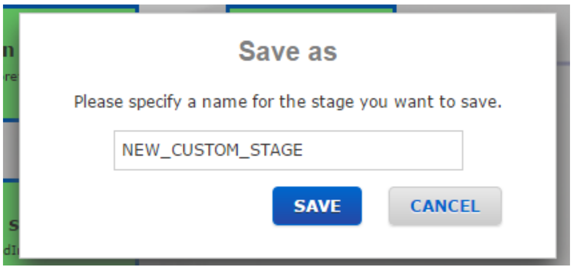
When all second level settings are complete and green, the stage configuration parameters are set. An "ADD TO STAGE.INI" button appears in the lower right hand corner of the screen.



Add custom stage

Single click to configure an item, double-click to change its type. The stage is ready to be saved when all the boxes are green.

Pressing the “ADD TO STAGE.INI” button saves the new stage configuration into the “stages.ini” file under the name “NEW_CUSTOM_STAGE”.

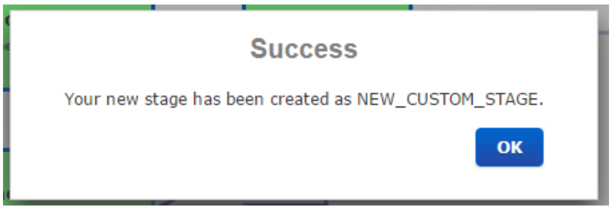


Save as

Please specify a name for the stage you want to save.

NEW_CUSTOM_STAGE

After the stage is added to the stages.ini file a confirmation window appears.



Success

Your new stage has been created as NEW_CUSTOM_STAGE.

Run the stage

After the confirmation that the new stage has been created as “NEW_CUSTOM_STAGE”, users can find the stage under **Stages → Add, remove, or edit stages**. The “NEW_CUSTOM_STAGE”, once physically connected can now be configured as describe in section 4.12: “System – Quick Configuration” or 4.13: “System – Manual Configuration”.

23.2 Encoder

In this configuration category, users are building the Position encoder interface parameters section of the stages.ini file:

Example:

```
; --- Position encoder interface parameters
; --- <Encoder.AquadB>
EncoderType = AquadB
EncoderResolution = 0.001 ; Unit
LinearEncoderCorrection = 0 ; Ppm
PositionerMappingFileName =
PositionerMappingLineNumber = 0
PositionerMappingMaxPositionError = 0 ; Unit
EncoderIndexOffset = 0 ; Unit
```

The XPS controller supports 4 types of position encoder interfaces:

- AquadB differential
- AquadB differential THETA
- Analog Interpolated with sine/cosine 1 Vpp
- Analog Interpolated THETA with sine/cosine 1 Vpp THETA

23.2.1 No Encoder (NoEncoder)

Encoder Type: NoEncoder

This encoder type is used when there is no position feedback from an encoder.

There are no additional parameters to set for this encoder type.

23.2.2 RS422 Differential (AquadB)

Encoder Type: AquadB

This encoder type is used when the position sensor delivers two square waves RS422 A differential signals.

Encoder Resolution

The stage displacement per encoder count, *EncoderResolution* (units), sets the resolution of the position encoder. It must be greater than zero.

NOTE

The encoder resolution is equal to 4-times the signal period (quadrature effect).

The *Stage displacement per encoder count* essentially defines the measurement units of the stage. Many parameters are derived from this value, in particular all parameters with units of length, such as velocities or accelerations. Therefore, it is critical this parameter be set correctly for proper operation.

To calculate the *Stage displacement per encoder count* value all of the following must be taken into account: the number of steps per revolution, screw pitch and any gear reduction in the stage.

Linear Encoder Correction

$$LinearEncoderCorrection = \left(\frac{Real\ increment}{Rounded\ increment} - 1\right) \cdot 1{,}000{,}000$$

The linear correction, *LinearEncoderCorrection* (parts per million), sets the correction applied to the *EncoderResolution* to compensate for linear error effects (see section 10.3: "Linear Error Correction"). This value must be between ±0.5*106. A zero value disables this feature.

The default value is zero.

Encoder Index Offset

- In units.

The EncoderIndexOffset is used with the fast PCO functions. It allows defining a preset value to extend the PCO position register when the travel of the stage is larger than the register range. This parameter should be used with care.

(Optional- three entries) Positioner mapping parameters.

Positioner Mapping File Name

The *PositionerMappingFileName* defines the name of the mapping file used for the positioner mapping compensation. No entry for the file name disables the feature.

Positioner Mapping Line Number

The *PositionerMappingLineNumber* defines the number of data lines in the positioner mapping file. This value must be greater than or equal to 3 and less than 200. This parameter is primarily used as a check to confirm the correctness of the mapping file.

Positioner Mapping Max Position Error

The *PositionerMappingMaxPositionError* (units) defines the maximum absolute value of the error corrections in the mapping file. This parameter is primarily used as a check to confirm the correctness of the mapping file.

Please refer to the Motion Tutorial chapter 10.0: "Compensation" and section 10.4: "Positioner Mapping" for further information about the positioner mapping functionality.

23.2.3 RS422 Differential with 3 Encoders (AquadBTheta) PP version only

Encoder Type: AquadBTheta

This encoder type is composed of three encoders and is used when the position sensor delivers two square waves RS422 A differential signals.

Encoder Radius

The theta encoder radius, *EncoderRadius (XY *10^{-6}),* is the radius of the theta (three encoders)

Maximum Encoder Correction X

The *MaximumEncoderCorrectionX* sets the maximum allowed correction for the X positioner.

Maximum Encoder Correction Y

The *MaximumEncoderCorrectionY* sets the maximum allowed correction for the Y positioner.

For all other configuration file parameters refer to section 23.2.2: "*RS422 Differential (AquadB)*".

23.2.4 Sine/Cosine 1 Vpp (AnalogInterpolated)

Encoder Type: AnalogInterpolated

This encoder type is used when the position sensor delivers two 1 Vpp sine/cosine signals that are interpolated by the XPS controller.

For more information on this refer to chapter 15.0: "Analog Encoder Calibration".

Encoder Scale Pitch

The stage displacement per encoder period, *EncoderScalePitch* (units), sets the displacement of the stage per encoder period. This parameter essentially defines the measurement units of the stage. Many parameters are derived from this entry, in particular all parameters with units of length, such as velocities or accelerations. Therefore it is critical this parameter value is set correctly for proper operation of the stage.

Encoder Interpolation Factor

The Encoder signal subdivisor, *EncoderInterpolationFactor*, sets the interpolation factor for the encoder signal interpolation. This value defines the resolution of the CurrentPosition and SetpointPosition of the stage as follows:

$$Resolution = EncoderScalePitch/EncoderInterpolationFactor$$

For a better understanding of the meaning of the CurrentPosition and SetpointPosition see section 10.1: "Definitions".

The *EncoderInterpolationFactor* must be an integer value greater than or equal to 1 and less than or equal to 65536.

The *EncoderInterpolationFactor* should be set relative to the position noise of the stage. It is not recommended to set a position resolution that is far less than the amplitude of the position noise. It is also important to note that the value of the *EncoderInterpolationFactor* has no impact on the resolution of the position used for the position servo loop. The position servo loop always uses calculations at the maximum position resolution possible. For example, (EncoderScalePitch/65536).

Linear Encoder Correction

Stages.ini file entry: *LinearEncoderCorrection*

LinearEncoderCorrection = ((Real increment/Rounded increment) - 1) * 1e^6

The linear correction, *LinearEncoderCorrection* (parts per million), sets the correction applied to the *EncoderResolution* to compensate for linear error effects (see section 10.3: "Linear Error Correction"). This value must be between ±0.5*10^6. A zero value disables this feature.

$$Corrected\mathrm{Resolution} = (1 + LinearCorrection \times 10^{-6}) \times Encoder\,\mathrm{Re}solution$$

The default value is zero.

Encoder ZM Plug

Stages.ini file entry: *EncoderZMPlug*

The mechanical zero input plug, *EncoderZMPlug*, defines where the mechanical zero signal is input. There are two possible settings:

- *Driver* — for mechanical zero through the driver board plug.
- *Encoder* — for mechanical zero through the encoder driver board plug.

Encoder Sinus Offset & Encoder Cosine Offset

Needed entries in the configuration file:

- sine channel offset correction: *EncoderSinusOffset.*
- sine channel offset correction: *EncoderCosinusOffset.*

The sine/cosine channel offset corrections, *EncoderSinusOffset* (V) and *EncoderCosinusOffset* (V), set the offset values of the two encoder signals for correction. They must be between ±0.1 V. A zero value disables this feature.

The default value is zero.

Encoder Phase Compensation

Stages.ini file entry: *EncoderPhaseCompensation*

The signal phase correction, *EncoderPhaseCompensation* (°), sets the phase correction of the cosine signal phase. This correction is applied after the differential amplitude correction. This value must be between ±10°. A zero value disables this feature.

$$PhaseCorrectedCosine = \frac{Sine \times \sin\left(PhaseCompensation \times \frac{\pi}{180}\right) + CorrectedCosine}{\cos\left(PhaseCompensation \times \frac{\pi}{180}\right)}$$

The default value is zero.

Encoder Differential Gain

Stages.ini file entry: *EncoderDifferentialGain*

The amplitude correction, *EncoderDifferentialGain*, sets the amplitude correction of the cosine signal amplitude. It must be between ±0.1. A zero value disables this feature.

$$CorrectedCosine = (1 + EncoderDifferentialGain) \times Cosine$$

The default value is zero.

Encoder Index Offset

- In units.

The EncoderIndexOffset is used with the fast PCO functions. It allows defining a preset value to extend the PCO position register when the travel of the stage is larger than the register range. This parameter should be used with care**.**

Encoder Hard Interpolator Error Check

Not Applicable — *Only for XPS-Q8.*

Encoder Sin Cos Radius Check

- Enabled or Disabled.

The purpose is to verify in real time that the Lissajou radius is not too small or too high for the interpolation denoting a problem with the stage 1 Volt peak to peak sin cos encoder. The limits are (0.6 to 1.2 V) for a good signal; out of that range, a positioner error is generated**.**

Current Position Filter Select

Frequency of the filter applied on the servo loop position. This filter does not limit the encoder bandwidth because it is applied after sin/cos interpolation however it generates a latency on the position. The ideal value for Curent Position Filter is 3.11. It is the lowest frequency with a latency compatible with the corrector frequency. See table below for more information.

Possible values for Current Position Filter Frequency or PCO Position Filter Frequency with associated latency:

Parameter Value	Filter Frequency	Position Latency
5000	No filter	
600	600 kHz	210 ns
230	230 kHz	540 ns
110	110 kHz	1.14 µs
49.74	49.74 kHz	2.5 µs
24.87	24.87 kHz	5 µs
12.43	12.43 kHz	10 µs
6.22	6.22 kHz	20 µs
3.11	3.11 kHz	40 µs
1.55	1.55 kHz	80 µs
0.78	0.78 kHz	160 µs

PCO Position Filter Select

The ideal value for Curent Position Filter is 3.11. It is the lower frequency with a latency compatible with the corrector frequency. Replace:

Frequency of the filter applied on the position used for PCO, External Gathering and AquadB output

This filter does not limit the encoder bandwidth because it is applied after sin/cos interpolation however it generates a latency on the position. The ideal value for PCO Position Filter depends of the application. See table above for more information.

(Optional- three entries) Positioner mapping parameters.

Positioner Mapping File Name

The *PositionerMappingFileName* defines the name of the mapping file used for the positioner mapping compensation. No entry for the file name disables the feature.

Positioner Mapping Line Number

The *PositionerMappingLineNumber* defines the number of data lines in the positioner mapping file. This value must be greater than or equal to 3 and less than 200. This parameter is primarily used as a check to confirm the correctness of the mapping file.

Positioner Mapping Max Position Error

The *PositionerMappingMaxPositionError* (units) defines the maximum absolute value of the error corrections in the mapping file. This parameter is primarily used as a check to confirm the correctness of the mapping file.

Please refer to the Motion Tutorial chapter 10.0: “Compensation” and section 10.4: “Positioner Mapping” for further information about the positioner mapping functionality.

Example:

```
; --- Position encoder interface parameters
; --- <Encoder.AnalogInterpolated>
EncoderType = AnalogInterpolated
LinearEncoderCorrection = 0 ; Ppm
EncoderZMPlug = Encoder
EncoderInterpolationFactor = 4000
EncoderScalePitch = 0.004 ; Unit
EncoderSinusOffset = 0 ; Volt
EncoderCosinusOffset = 0 ; Volt
EncoderPhaseCompensation = 0 ;--- deg
EncoderDifferentialGain = 0
PositionerMappingFileName =
PositionerMappingLineNumber = 0
PositionerMappingMaxPositionError = 0 ; Unit
EncoderIndexOffset = 0 ; Unit
EncoderHardInterpolatorErrorCheck = Enabled ; # Only used by ISA CIE
EncoderSinCosRadiusCheck = Enabled ; # Only used by PCI CIE
CurrentPositionFilterSelect = 3.1 ; kHz # Only used by PCI CIE
PCOPositionFilterSelect = 5000 ; kHz # Only used by PCI CIE
```

23.2.5 Sine/Cosine 1 Vpp (AnalogInterpolated Theta) PP version only

Encoder Type: AnalogInterploatedTheta

This encoder type is composed of three encoders and is used when the position sensor delivers two 1 Vpp sine/cosine signals that get interpolated by the XPS controller.

Encoder Radius

The theta encoder radius, *EncoderRadius (XY *10^{-6}),* is the radius of the theta (three encoders).

Maximum Encoder Correction X

The *MaximumEncoderCorrectionX* sets the maximum allowed (limit) correction for the X positioner.

Maximum Encoder Correction Y

The *MaximumEncoderCorrectionY* sets the maximum allowed (limit) correction for the Y positioner.

***For all other configuration file parameters refer to section 23.2.4:* "Sine/Cosine 1 Vpp (AnalogInterpolated)".**

23.3 Profiler

In this configuration category, users are building the Profile generator parameters section of the stages.ini file.

Example:

```
; --- Profile generator parameters
; --- <Profiler.Sgamma>
MaximumVelocity = 300 ; Unit/s
JogMaximumVelocity = 300 ; Unit/s
MaximumAcceleration = 2500 ; Unit/s²
JogMaximumAcceleration = 2500 ; Unit/s²
EmergencyDecelerationMultiplier = 4
MinimumJerkTime = 0.02 ; s
MaximumJerkTime = 0.02 ; s
TrackingCutOffFrequency = 25 ; Hz
```

23.3.1 Type: Sgamma

Maximum Velocity

The maximum velocity, *MaximumVelocity* (units/s), sets the maximum velocity the stages can be commanded to move. This value must be greater than zero and less than or equal to the stage velocity at maximum command (see section 23.9.18).

When used with stepper motors and sine/cosine position control the value for the *maximum velocity* must be less than or equal to the *DisplacementPerFullStep* (see section 23.9.17) multiplied by the servo loop frequency (8 kHz). For smooth operation, however, we recommend to not exceed one quarter of this maximum possible value.

Jog Maximum Velocity

The maximum velocity while in jog mode. *JogMaximumVelocity* (units/s) sets the maximum velocity the stages can be commanded to move in jog mode. This value must be greater than zero and less than or equal to the stage velocity at maximum command (see section 23.9.18).

When used with stepper motors and sine/cosine position control the value for the *jog maximum velocity* must be less than or equal to the *DisplacementPerFullStep* (see section 23.9.17) multiplied by the servo loop frequency (8 kHz). For smooth operation, however, we recommend to not exceed one quarter of this maximum possible value.

Maximum Acceleration

The maximum acceleration, *MaximumAcceleration* (units/s^2), sets the maximum acceleration the stage can be commanded to move. This value must be greater than zero and less than or equal to the stage acceleration at maximum command (see section 23.9.2).

The recommended value for smooth displacement is four times the maximum velocity.

Jog Maximum Acceleration

The maximum acceleration while in jog mode. *MaximumAcceleration* (units/s^2) sets the maximum acceleration the stage can be commanded to move in jog mode. This value must be greater than zero and less than or equal to the stage acceleration at maximum command (see section 23.9.2).

The recommended value for smooth displacement is four times the jog maximum velocity.

Emergency Deceleration Multiplier

The emergency deceleration multiplier, *EmergencyDecelerationMultiplier*, defines the ratio between the emergency deceleration during an emergency brake and the normal deceleration during normal brake. This value must be greater than or equal to one.

The default setting is 4.

Minimum/Maximum Jerk Time:

The SGamma profile generator jerk times, *MinimumJerkTime* (s) and *MaximumJerkTime* (s), set the jerk times of the SGamma profile generator. The minimum jerk time must be less than the maximum jerk time and greater than or equal to the profile generator cycle period (0.4 ms).

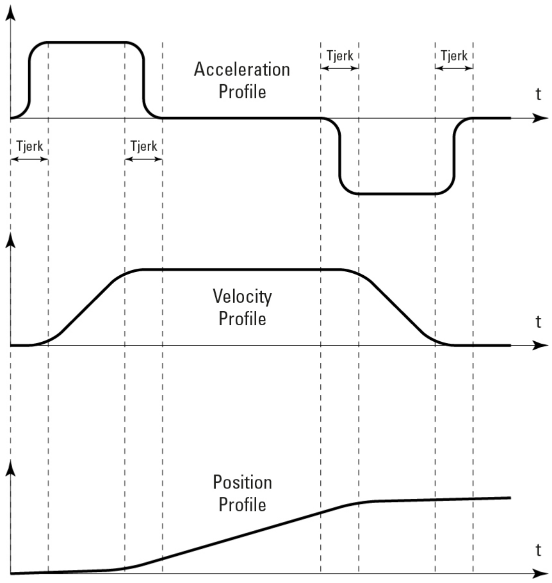


Figure 82: SGamma motion profile.

The recommended values for a smooth displacement are:

- MaximumJerkTime 0.05 s
- MinimumJerkTime 0.005 s

For more information about the SGamma motion profiler, see also the Motion Tutorial, chapter 7.0: "Motion".

Tracking Cut Off Frequency

The tracking mode filter cut off frequency, *TrackingCutOffFrequency* (Hz), sets the cut off frequency of the filter applied to the tracking input. This frequency must be at least ten times less than the servo loop bandwidth to avoid poor tracking. It must be also greater than or equal to zero (disable) and less than or equal to half of the profile generator cycle frequency (2500 Hz).

The default setting is 5 Hz.

23.4 Servitudes

In this configuration category, users are building the Position encoder interface parameters section of the stages.ini file:

Example:

```
; --- Travels and servitudes type parameters
; --- <Servitudes.StandardEORDriverPlug>
ServitudesType = StandardEORDriverPlug
MinimumTargetPosition = -45 ; Unit
MaximumTargetPosition = 45 ; Unit
HomePreset = 0 ; Unit
```

The XPS controller supports 5 settings for the limit sensors input plug:

- No Servitudes
- Driver board
- Encoder board
- None for spindle group
- None for piezo driver

The choice of the setting for the limit sensor input plug depends on where these signals are electrically connected to the XPS controller: either through the plug on the driver board or through the plug on the encoder board. The setting *none for spindle group* is only used for spindle groups.

The limit sensor input plug setting is based on other stage settings, such as: maximum stage travel, maximum drivable speeds and accelerations, etc.

23.4.1 No Servitudes (NoServitudes)

Servitudes Type: NO_SERVITUDES

This setting disregards the end of run detection, even if the signals are present. There are no parameters to set for this servitude type.

Minimum Target Position

The minimum position, *MinimumTargetPosition* (units), sets the minimum allowed position for any move command. This value must be less than the maximum position.

Maximum Target Position

The maximum position, MaximumTargetPosition (units), sets the maximum allowed position for any move command. This value must be greater than the Home Preset.

Home Preset

The home position, *HomePreset* (units), sets the position value of the home reference. This value must be between the minimum position and the maximum position defined above.

23.4.2 Piezo

Servitudes Type: Piezo

This setting can only be used with piezo stages. It disregards the end of run detection, even if the signals are present.

23.4.3 Spindle

Servitudes Type: Spindle

This setting can only be used with stages configured as spindle groups. It disregards the end of run detection, even if the signals are present.

Home Preset

The home position, *HomePreset* (units), sets the position value of the home reference. This value must be greater than zero and less than the *SpindlePeriod.*

Spindle Period

The *SpindlePeriod* (units) sets the period of the spindle rotation. This value must be greater than zero.

23.4.4 Driver Board (StandardEORDriverPlug)

Servitudes Type: StandardEORDriverPlug

This setting is used when the travel limit signals are wired to the XPS controller through the driver board plug.

Minimum Target Position

The minimum position, *MinimumTargetPosition* (units), sets the minimum allowed position for any move command. This value must be less than the maximum position.

Maximum Target Position

The maximum position, *MaximumTargetPosition* (units), sets the maximum allowed position for any move command. This value must be greater than the Home Preset.

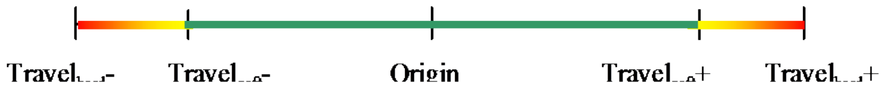


Figure 83: Minimum and maximum travel position.

Home Preset

The home position, *HomePreset* (units), sets the position value of the home reference. This value must be between the minimum position and the maximum position defined above.

23.4.5 Encoder Board (StandardLimitAndLimitEncoderPlug)

Servitudes Type: StandardLimitAndLimitEncoderPlug

This setting is used when **only** the travel limit signals are wired to the XPS controller through the encoder board plug.

For all other configuration file parameters refer to section 23.4.4: "Driver Board (StandardEORDriverPlug)".

23.4.6 Encoder Board (StandardLimitAndHomeEncoderPlug)

Servitudes Type: StandardLimitAndHomeEncoderPlug

This setting is used when **both** the Home signal and the travel limit signals are wired to the XPS controller through the encoder board plug.

For all other configuration file parameters refer to section 23.4.4: "Driver Board (StandardEORDriverPlug)".

23.5 Backlash

In this configuration category, users are building the Backlash compensation parameters section of the stages.ini file.

Example:

```
; --- Backlash parameters
; --- <Backlash.Standard>
Backlash = 0 ; Unit
CurrentVelocityCutOffFrequency = 100 ; Hz
CurrentAccelerationCutOffFrequency = 100 ; Hz
```

23.5.1 Type: Standard

Backlash

The stage backlash, *Backlash* (units), sets the backlash compensation value applied to the target position for a move (see Motion Tutorial, section 10.2: "Backlash Compensation"). This value must be greater than or equal to zero (backlash disabled).

The default value is zero.

Current Velocity and Acceleration Cut Off Frequency entries for the configuration file:

- ***Current Velocity Cut Off Frequency --*** Gathering velocity cut off frequency
- ***Current Acceleration Cut Off Frequency --*** Gathering acceleration cut off frequency

The gathering cut off frequencies, *CurrentVelocityCutOffFrequency* (Hz) and *CurrentAcceleration-CutOffFrequency* (Hz), set the cut off frequencies for low-pass filters that are applied to the CurrentVelocity and CurrentAcceleration saved by the gathering feature. These filters reduce derivative noise. They must be greater than zero (filter disabled) and less than half of the servo loop frequency (8 kHz).

The default value is 100 Hz which is about five times greater than the bandwidth of the position servo loop of a typical screw driven stage.

23.6 Home Search

In this configuration category, users are building the Home search process parameters section of the stages.ini file.

Example:

```
; --- Home search process parameters
; --- <HomeSearch.MechanicalZeroAndIndexHomeSearch>
HomeSearchSequenceType = MechanicalZeroAndIndexHomeSearch
HomeSearchMaximumVelocity = 100 ; Unit/s
HomeSearchMaximumAcceleration = 500 ; Unit/s²
HomeSearchTimeOut = 5 ; s
HomingSensorOffset = 0 ; Unit
```

The XPS controller supports 8 different home search processes:

- No home search
- Mechanical zero only
- Mechanical zero and index
- Minus end of run only
- Minus end of run and index
- Current position as home
- Plus end of run only
- Index only

The choice depends on the availability of reference signals such as: a mechanical zero signal, a minus end of run signal or an index signal.

For more information about the possible home search processes, please refer to the Motion Tutorial, section 7.2: “Home Search”.

23.6.1 No Home Search (NoHomeSearch)

Home Search Sequence Type: NO_HOME_SEARCH

This home search process is used only used for Dummy configuration.

There are no parameters to set for this home search type.

23.6.2 Mechanical Zero and Index (MechanicalZeroAndIndexHomeSearch)

Home Search Sequence Type: MechanicalZeroAndIndexHomeSearch

This home search process is used when the mechanical zero signal and the index signal are both used for the home search.

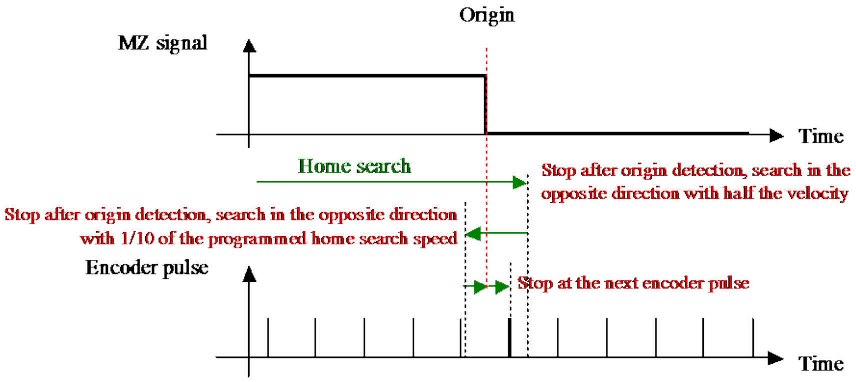

Origin
MZ signal
Time
Home search
Stop after origin detection, search in the opposite direction with half the velocity
Stop after origin detection, search in the opposite direction with 1/10 of the programmed home search speed
Encoder pulse
Stop at the next encoder pulse
Time

Figure 84: Mechanical zero and index home search process.

Home Search Maximum Velocity

The home search maximum velocity, *HomeSearchMaximumVelocity* (units/s), sets the maximum velocity of the profile generator during the home search process. It must be greater than zero and less than or equal to the *MaximumVelocity* (see section 23.4.5).

The default value is one half of the *MaximumVelocity*.

Home Search Maximum Acceleration

The home search maximum acceleration, *HomeSearchMaximumAcceleration* (units/s²), sets the maximum acceleration of the profile generator during the home search process. This value must be greater than zero and less than or equal to the *MaximumAcceleration* (see section 23.4.5).

The default value is one half of the maximum acceleration.

Home Search Time Out

The home search time out, *HomeSearchTimeOut* (s), sets the time out value for the home search process. When the *HomeSearchTimeOut* time is elapsed and the home search process is not yet finished, the XPS controller will generate an emergency stop and will abort the home search process. The *HomeSearchTimeOut* value must be greater than or equal to the profile generator period (400 µs).

Homing Sensor Offset

- In units.
- [0 : 1000]

This offset value is used to move the stage *HomingSensorOffset* from the home detected position. Example: With a stage having a HomePreset of zero and a *HomingSensorOffset* of zero will have the stage carriage being at a position. The same stage having a HomePreset of zero and a HomingSensorOffset of ten units will have the stage carriage being ten units away from the previous configuration.

23.6.3 Mechanical Zero Only (MechanicalZeroHomeSearch)

Home Search Sequence Type: MechanicalZeroHomeSearch

This home search process is used only when the mechanical zero signal is used for the home search.

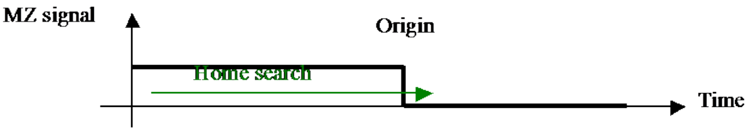


Figure 85: Mechanical zero home search process.

For all other configuration file parameters refer to section 23.6.2: "Mechanical Zero and Index (MechanicalZeroAndIndexHomeSearch)".

23.6.4 MinusEndOfRunAndIndexHomeSearch

Home Search Sequence Type: MinusEndOfRunAndIndexHomeSearch

This home search process is used when both the minus end of run signal and the index signal are used for the home search.

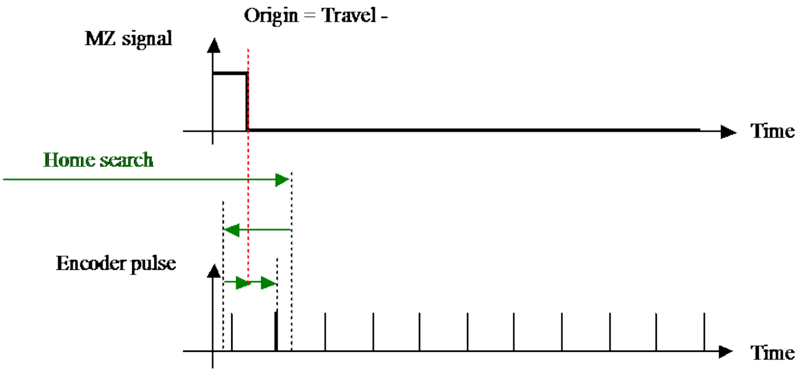


Figure 86: Minus end of run and index home search process.

***For all other configuration file parameters refer to section 23.6.2:* “Mechanical Zero and Index (MechanicalZeroAndIndexHomeSearch)”.**

23.6.5 MinusEndOfRunHomeSearch

Home Search Sequence Type: MinusEndOfRunHomeSearch

This home search process is used only when the minus end of run signal is used for the home search.

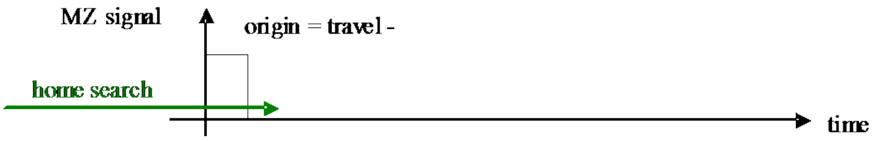


Figure 87: Minus end of run home search process.

***For all other configuration file parameters refer to section 23.6.2:* “Mechanical Zero and Index (MechanicalZeroAndIndexHomeSearch)”.**

23.6.6 PlusEndOfRunHomeSearch

Home Search Sequence Type: PlusEndOfRunHomeSearch

This home search process is used only when the plus end of run signal is used for the home search.

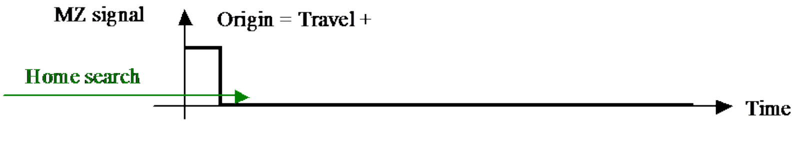


Figure 88: Plus end of run home search process.

***For all other configuration file parameters refer to section 23.6.2:* “Mechanical Zero and Index (MechanicalZeroAndIndexHomeSearch)”.**

23.6.7 IndexHomeSearch

Home Search Sequence Type: IndexHomeSearch

This home search process is used only when the plus end of run signal is used for the home search.

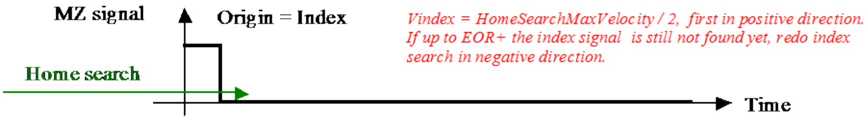

MZ signal
Origin = Index
Vindex = HomeSearchMaxVelocity / 2, first in positive direction.
If up to EOR+ the index signal is still not found yet, redo index search in negative direction.
Home search
Time

Figure 89: Index home search process.

For all other configuration file parameters refer to section 23.6.2:
"Mechanical Zero and Index (MechanicalZeroAndIndexHomeSearch)*".*

23.6.8 CurrentPositionAsHome

Home Search Sequence Type: CurrentPositionAsHome

This home search process is used when the current position is defined as the home position.

For all other configuration file parameters refer to section 23.6.2:
"Mechanical Zero and Index (MechanicalZeroAndIndexHomeSearch)*".*

23.7 Motion done

In this configuration category, users are building the Home search process parameters section of the stages.ini file.

<u>Example:</u>

```
; --- Motion done parameters
; --- <MotionDone.Theoretical>
MotionDoneMode = Theoretical
```

The motion done condition mode defines when a motion is completed. When set to *theoretical motion end,* or a setting *position and velocity checking* can be defined by the user to allow a more precise definition of the motion done by conditioning the motion completion to a number of parameters that take the settling of the positioner into account. The default value for this parameter is *theoretical motion end*.

23.7.1 Theoretical

Motion Done Mode: Theoretical

A theoretical definition of when motion is complete calculated by the motion profiler. Note that this does not take into account the settling of the positioner at the end of the move. See section 7.5: "Motion Done" for more information.

There are no additional parameters to set for this Motion done type.

23.7.2 VelocityAndPositionWindow

Motion Done Mode: VelocityAndPositionWindow

The VelocityAndPositionWindow MotionDone allows a more precise definition of when a motion is complete by specifying the end of the move with a number of parameters that take the settling of the positioner into account. See section 7.5: "Motion Done" for more information.

Other configuration parameters:

- ***Motion Done Mode***
- ***Motion Done Position Threshold***
- ***Motion Done Velocity Threshold***
- ***Motion Done Checking Time***
- ***Motion Done Mean Period***
- ***Motion Done Timeout***

23.8 Corrector

In this configuration category, users are building the Position servo loop parameters section of the stages.ini file.

Example:

```
; --- Position servo loop parameters
; --- <Corrector.PIDFFAcceleration>
CorrectorType = PIDFFAcceleration
ClosedLoopStatus = Closed
FatalFollowingError = 1 ; Unit
KP = 300000
KI = 10000000
KD = 800
KS = 0.8
GKP = 0
GKD = 0
GKI = 0
KForm = 0 ; Unit
IntegrationTime = 1E+99 ; s
DerivativeFilterCutOffFrequency = 4000 ; Hz
DeadBandThreshold = 0 ; Unit
KFeedForwardAcceleration = 1
NotchFrequency1 = 0 ; Hz
NotchBandwidth1 = 0 ; Hz
NotchGain1 = 0
NotchFrequency2 = 0 ; Hz
NotchBandwidth2 = 0 ; Hz
NotchGain2 = 0
KFeedForwardJerk = 0
```

23.8.1 NoCorrector

Corrector Type: NoCorrector

This servo loop type is only used for Dummy configuration.

There are no additional parameters to set for this servo loop type.

23.8.2 NoEncoderPosition

Corrector Type: NoEncoderPosition

This servo loop type is used for stages without encoder, i.e. stages used always in open loop.

There are no additional parameters to set for this corrector type.

23.8.3 PIDDualFFVoltage

Corrector Type: PIDDualFFVoltage

This servo loop type is used when the position servo loop directly drives the voltage applied to the motor, for instance a DC motor without tachometer connected to a voltage amplifier.

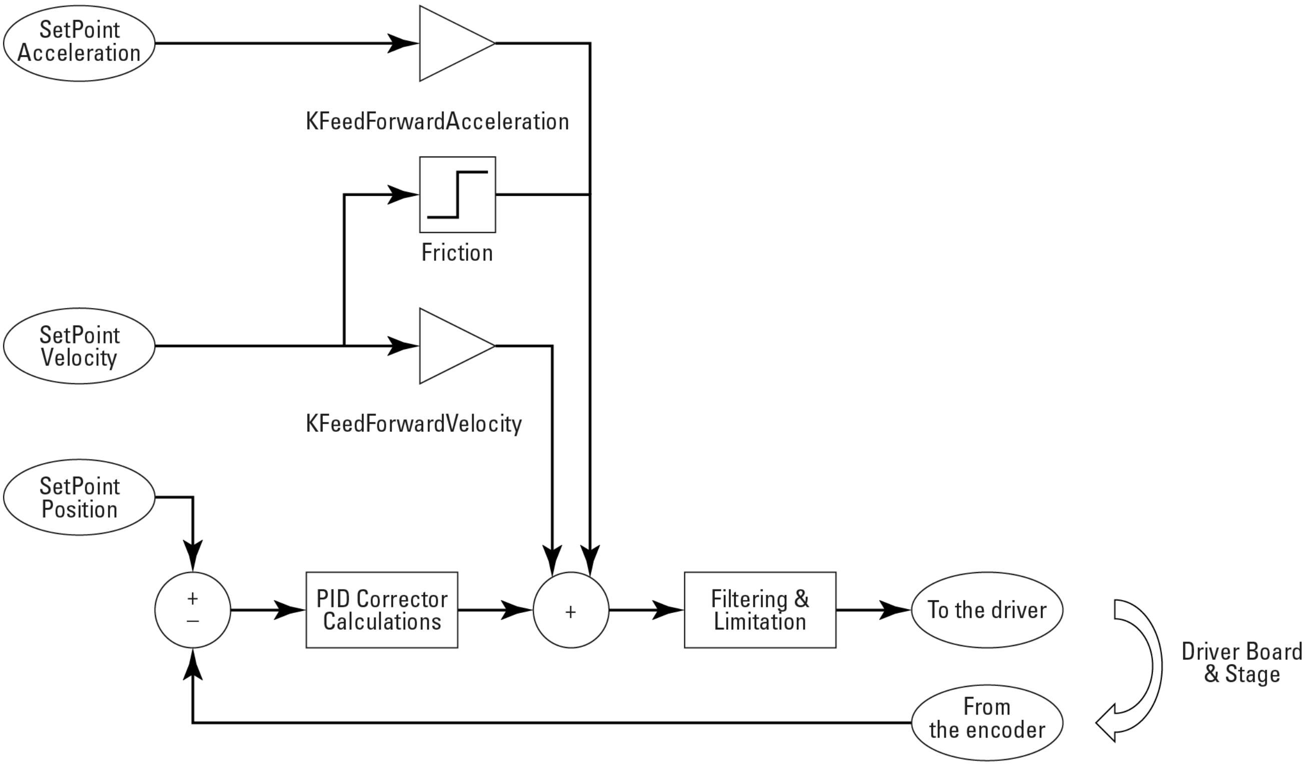


Figure 90: PIDDualFFVoltage corrector.

This servo loop type features a parallel PID servo loop with feed forwards for velocity and acceleration, and two notch filters.

Closed Loop Status

The position servo loop status parameter sets the position servo loop either to open loop i.e. without feedback of a position encoder, or to closed loop, i.e. with feedback from a position encoder. The default value for this parameter is *closed*.

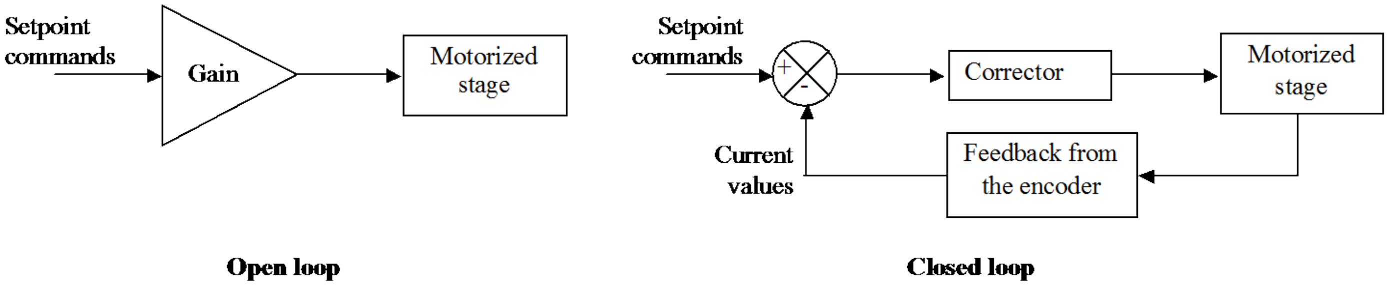


Figure 91: Open and closed loop.

Fatal Following Error

The value for the fatal following error sets the maximum allowed following error of the positioner before generating an error response from the controller. This error is defined as the absolute value of the difference between the setpoint position and the current position. This value is calculated each servo cycle. A following error that exceeds this value will generate the corresponding error code and action. It must be greater than zero.

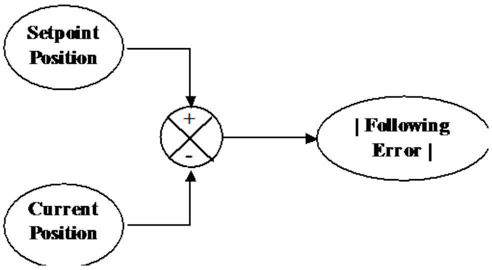


Figure 92: Following error.

Dual Feed Forward (*PIDDualFFVoltage)* PID Parameters entries for the configuration file:

- ***K P*** — PID servo loop proportional gain
- ***K I*** — PID servo loop integral gain
- ***K D*** — PID servo loop derivative gain
- ***K S*** — PID integral saturation value
- ***Integration Time* --** PID integration time
- ***Derivative Cut Off Frequency* --** PID derivative filter cut off frequency
- ***K Feed Forward Velocity*** -- Velocity feedforward gain
- ***K Feed Forward Acceleration* --** Acceleration feedforward gain
- ***K Feed Forward Velocity Open Loop* --** Velocity feedforward gain in open loop

The PID servo loop parameters *KP*, *KI*, *KD*, *KFeedForwardVelocity* and *FeedForwardAcceleration* define the bandwidth of the servo loop. They must be greater than or equal to zero.

The PID integral saturation value *KS* sets the limit of the integral part of the PID servo loop that is applied to the total servo loop output. The value for *KS* must be between 0 and 1.

The PID *IntegrationTime* (seconds) defines the time span for integration of the residual errors. A small value limits the effect of the integral gain *KI*. It must be greater than or equal to the servo loop period (125 µs).

The PID *DerivativeFilterCutOffFrequency* (Hz) sets the cut-off frequency of the derivative filter. It can be used to reduce the noise introduced by the numerical derivation of the following error. It must be greater than or equal to zero (zero means filter is disabled) and less than or equal to half of the servo loop frequency (8 kHz).

As the output of the position servo loop is neither velocity nor acceleration, individual feed forwards for the velocity and for the acceleration can be set. The *KFeedForwardVelocityOpenLoop* is only used when the position servo loop is in open loop.

All parameters will have an impact on system performance and should be set together. It should be noted that parameters are best set for desired performance according the requirements of the motion application (small following error during trajectory motion, short settling time after a displacement, …). This document will present context specific information on setting PID parameters, but is not intended to be a tutorial on servo loops. Please refer to the common literature for a general treatment of servo loops.

Choice: PID servo loop

$C(p) = KP + \frac{KI}{p} + \frac{KD \times p}{(\tau_d \times p + 1)} \approx KP + \frac{KI}{p} + KD \times p$, where p is the Laplace variable.

Mechanical transfer function: $H(p) = \frac{1}{p \times 60 \times G \times Kv \times (\tau_m \times p + 1)}$

Where: G: ratio between motor rotation and stage displacement (revolution/units)

Kv: motor voltage constant (V/rpm)

$\tau_m = \frac{R_{mot} \times J}{Kt^2}$: stage mechanical time constant (s)

Where: R_{mot}: motor winding resistance per phase (Ω)

J: total inertia on the motor axis (kg.m²). See section 23.10.6: "DRV01AnalogVelocity (with tachometer feedback)" for detail.

Kt: motor torque constant (N.m/A)

Assumptions

- Position closed loop time constants (real and resonance): $\tau_1 = \tau_2$ (s) is between 8 ms and 16 ms (10 Hz and 20 Hz) depending on the first mechanical resonance.
- The damping factor of the closed loop to avoid overshoots (see closed transfer function numerator) is: $\zeta = 1.25$

Notice: $\tau = \frac{1}{2 \times \pi \times f}$

PID parameters

The closed loop transfer function is:

$$T(p) \approx \frac{\frac{KD}{KI} \times p^2 + \frac{KP}{KI} \times p + 1}{\frac{60 \times G \times Kv \times \tau_m}{KI} \times p^3 + \left(\frac{KD}{KI} + \frac{60 \times G \times Kv}{KI}\right) \times p^2 + \frac{KP}{KI} \times p + 1}$$

$$\approx \frac{\left(2 \times \zeta \times \tau_1 \times \tau_2 + \tau_2^2 - \frac{\tau_1 \times \tau_2^2}{\tau_m}\right) \times p^2 + (\tau_1 + 2 \times \zeta \times \tau_2) \times p + 1}{(\tau_1 \times p + 1) \times (\tau_2^2 \times p^2 + 2 \times \zeta \times \tau_2 + 1)}$$

Identification with $(\tau_1 \times p + 1) \times (\tau_2^2 \times p^2 + 2 \times \zeta \times \tau_2 + 1)$ results in:

- PID servo loop proportional gain: $KP = \frac{(\tau_1 + 2 \times \zeta \times \tau_2) \times 60 \times G \times Kv \times \tau_m}{\tau_1 \times \tau_2^2}$
- PID servo loop integral gain: $KI = \frac{60 \times G \times Kv \times \tau_m}{\tau_1 \times \tau_2^2}$
- PID servo loop derivative gain:

$$KD = \frac{\left((2 \times \zeta \times \tau_1 \times \tau_2 + \tau_2^2) \times \tau_m - \tau_1 \times \tau_2^2\right) \times 60 \times G \times Kv}{\tau_1 \times \tau_2^2}$$

- PID integral saturation value: default $KS = 0.8$
- PID integration time: $\mathrm{IntegrationTime} = 1e^{99} s$ (full integration)
- PID derivative filter cut off frequency: $\mathrm{DerivativeCutOffFrequency} = 5000\ \mathrm{Hz}$ (maximum)

- velocity feedforward gain: $\mathrm{KFeedForwardVelocity} = 60 \times G \times Kv$
- acceleration feedforward gain: $\mathrm{KFeedForwardAcceleration} = \dfrac{\mathrm{R_{mot}} \times J}{Kt}$
- velocity feedforward gain in open loop: $\mathrm{KFeedForwardVelocityOpenLoop} = 60 \times G \times Kv$

Dual Feed Forward (*PIDDualFFVoltage)* Variable PID parameters entries for the configuration file:

- ***G K P*** — variable PID proportional gain multiplier
- ***G K I*** — variable PID integral gain multiplier
- ***G K D*** — variable PID derivative gain multiplier
- ***K Form*** — variable PID form coefficient

In addition to the classical gains of the PID servo loop, the XPS controller PID position servo loop features variable gain factors *GKP*, *GKD*, and *GKI*. These gains can be used to reduce settling times of systems exhibiting non-uniform behavior or to tighten the servo loop during the final segment of a move. For example, a stage with a high level of friction will have a response which is dependent on the size of the move: friction is negligible for a large move, but becomes a predominant factor for small moves. For this reason, the required response of the system to reach the commanded position is not the same for small and large moves. The optimum values of PID parameters for small moves are often higher than the optimum values for large moves. Users that do not want to set individual PID gains for different size motions can benefit from variable PID gains. Variable gains are driven by the distance between the target position and the current position. They must be greater than -1.

The parameter PID form coefficient value *KForm* (units) defines the relationship between the distance to the target and the change of the PID gains:

$$K_x = \left(1 + GK_x \bullet \frac{KForm}{|TagetPosition - CurrentPosition| + KForm)}\right) \bullet K_x$$

It must be greater than or equal to zero.

The smaller the variable PID form coefficient value is, the sharper the change of the PID gains.

GKp = 10 Kp = 2

Target Position = 0

Encoder Position = -100 to 100

$$\mathrm{Kp}_{\,(\mathrm{Kform,\ Encoder\ Position})} = \left[1 + \mathrm{GK} \cdot \left(\frac{\mathrm{Kform}}{|\,\mathrm{Target\ Position} - \mathrm{Encoder\ Position}\,| + \mathrm{Kform}}\right)\right] \cdot \mathrm{Kp}$$

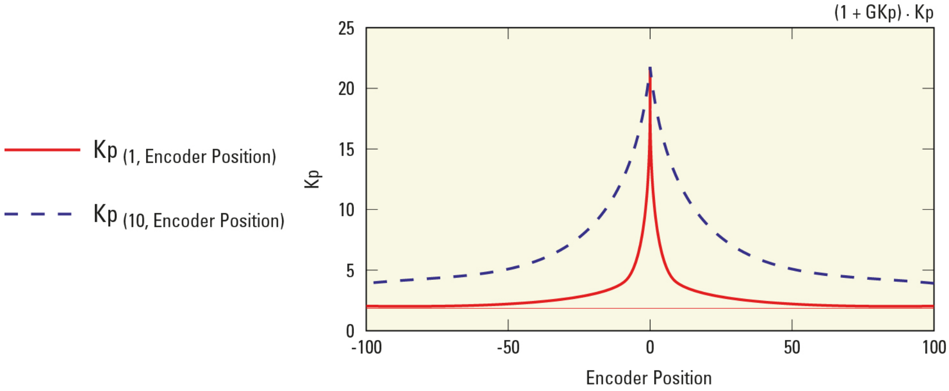


Figure 93: Influence of variable gains.

The default value for these parameters is 0 which disables the variable gains.

Dead Band Threshold

The servo loop dead band threshold sets the dead band value of the position loop. When set to a value other than zero, the position loop is disabled when the following error is less than the value for the dead band threshold AND the theoretical motion is done. In some cases, this can avoid oscillations of stages with backlash or friction. It can also reduce stage settling times, but may result in residual error from the target position. It must be greater than or equal to zero. The default value for this parameter is 0, which disables this feature.

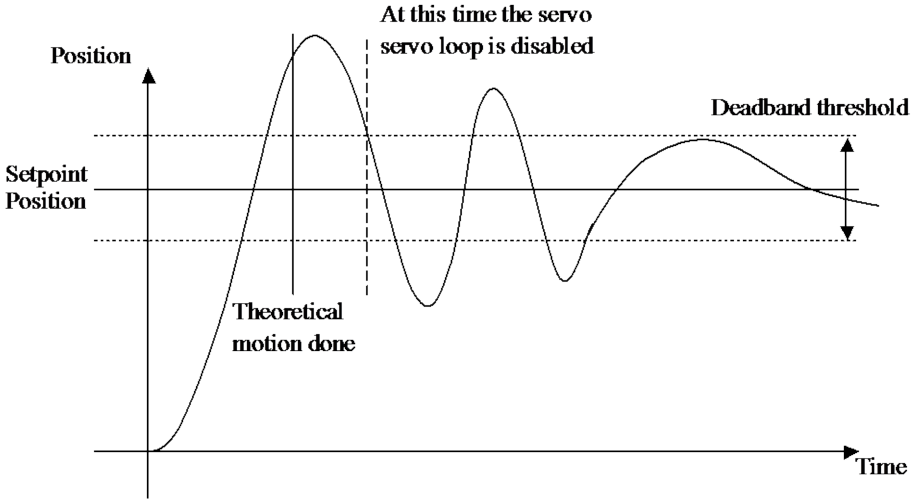


Figure 94: Deadband threshold.

Friction

The friction compensation can be used to compensate for friction of stages during motion. When set to a value other than zero, the friction parameter defines a voltage applied directly to the motor consistent with the direction of motion. The default value for this parameter is 0, which disables this feature.

Dual Feed Forward (*PIDDualFFVoltage)* Notch Filters Parameters entries for the configuration file:

- ***Notch Frequency 1*** and ***Notch Frequency 2*** — first and second notch filter center frequency
- ***Notch Bandwidth 1*** and ***Notch Bandwidth 2*** — first and second notch filter bandwidth:
- ***Notch Gain 1*** and ***Notch Gain 2*** — first and second notch filter gain

The output of the position servo loop is filtered by two notch filters. These filters can be used to avoid the excitation of specific frequencies. They are defined by their center frequencies *NotchFrequency<n°>* (Hz), bandwidths *NotchBandwidth<n°>* (Hz) and their gains *NotchGain<n°>*. The frequencies and bandwidths must be greater than or equal to zero (filter disabled) and less than or equal to half of the servo loop frequency (8 kHz). The gain must be greater than or equal to zero. The default value for these parameters is 0, which disables this feature.

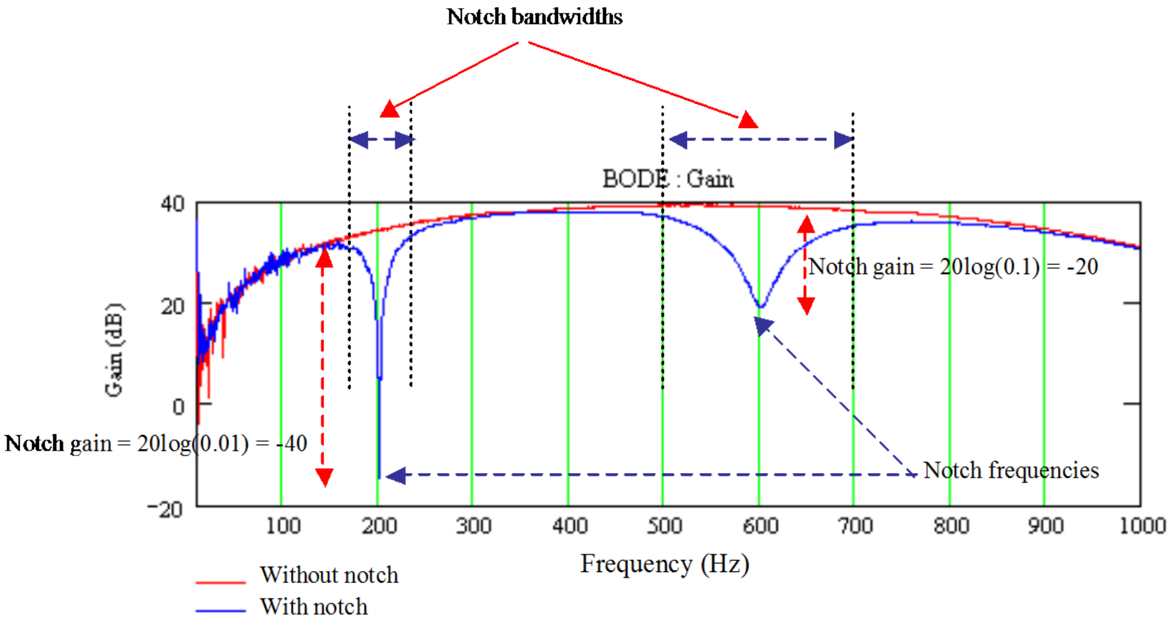


Figure 95: Notch filters.

23.8.4 PIDFFAcceleration

Corrector Type: PIDFFAcceleration

This servo loop type is used when a constant value applied to the driver results in a constant acceleration of the stage.

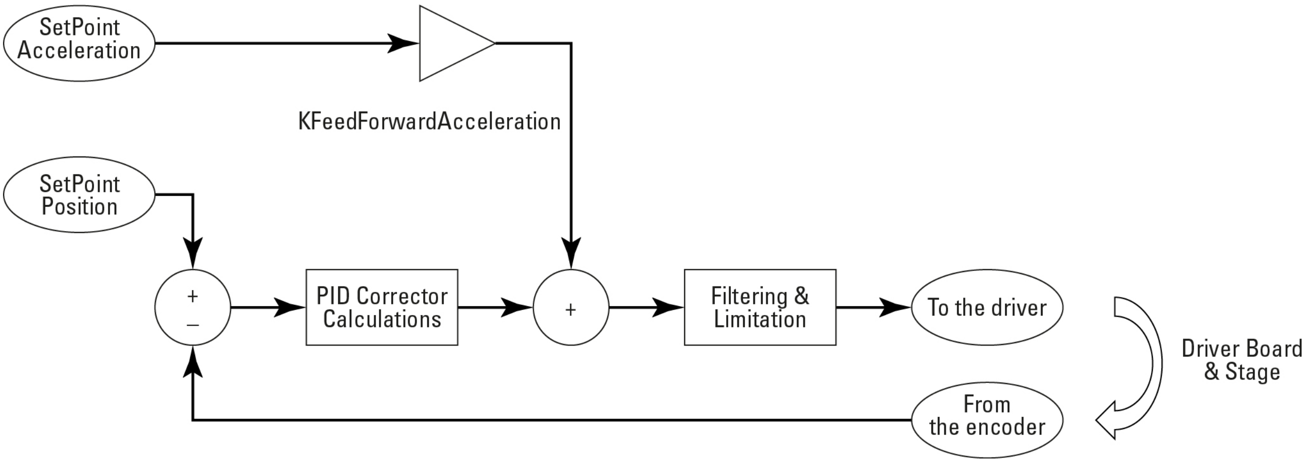


Figure 96: PIDFFAcceleration corrector.

This servo loop type features a parallel PID servo loop with feedforward acceleration and two notch filters.

Closed Loop Status

The position servo loop status parameter sets the position servo loop either to open loop i.e. without feedback of a position encoder, or to closed loop, i.e. with feedback from a position encoder. The default value for this parameter is *closed*.

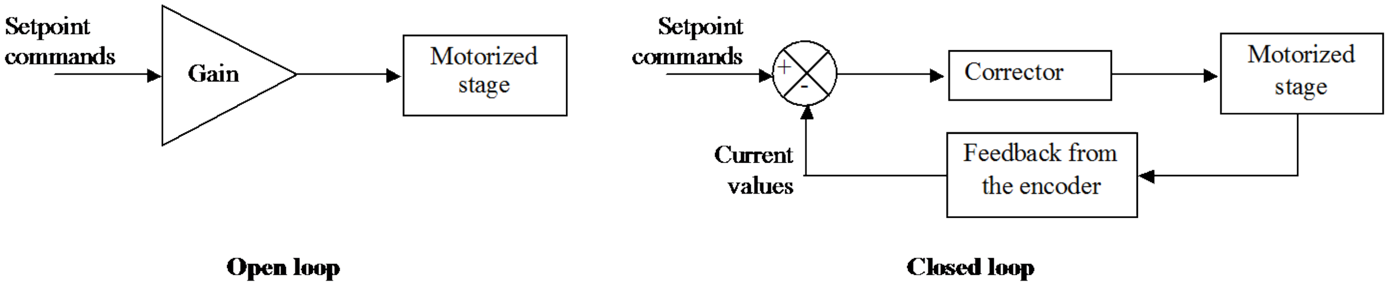


Figure 97: Open and closed loop.

Fatal Following Error

The value for the fatal following error sets the maximum allowed following error of the positioner before generating an error response from the controller. This error is defined as the absolute value of the difference between the setpoint position and the current position. This value is calculated each servo cycle. A following error that exceeds this value will generate the corresponding error code and action. It must be greater than zero.

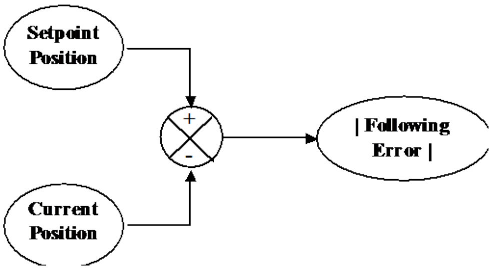


Figure 98: Following error.

Feed Forward (*PIDFFAcceleration*) PID Parameters entries for the configuration file:

- ***K P*** — PID servo loop proportional gain
- ***K I*** — PID servo loop integral gain
- ***K D*** — PID servo loop derivative gain
- ***K S*** — PID integral saturation value
- ***Integration Time*** — PID integration time:
- ***Derivative Cut Off Frequency*** — PID derivative filter cut off frequency
- ***K Feed Forward Acceleration*** — Acceleration feedforward gain
- ***K Feed Forward Jerk*** — Jerk feedforward gain

The PID servo loop parameters *KP*, *KI*, *KD* and *KFeedForwardAcceleration, KFeedForwardJerk* define the bandwidth of the servo loop. They must be greater than or equal to zero.

The PID integral saturation value *KS* sets the limit of the integral part of the PID servo loop that is applied to the total servo loop output. The *KS* parameter must be between 0 and 1.

The PID *IntegrationTime* (seconds) defines the time span for integration of the residual errors. A small value limits the effect of the integral gain *KI*. It must be greater than or equal to the servo loop period (125 µs).

The PID *DerivativeFilterCutOffFrequency* (Hz) sets the cut-off frequency of the derivative filter. It can be used to reduce the noise introduced by the numerical derivation of the following error. It must be greater than or equal to zero (zero means filter is disabled) and less than or equal to half of the servo loop frequency (8 kHz).

All parameters will have an impact on system performance and should be set together. It should be noted that parameters are best set for desired performance according the requirements of the motion application (small following error during trajectory motion, short settling time after a displacement, …). This document will present context specific information on setting PID parameters, but is not intended to be a tutorial on servo loops. Please refer to the common literature for a general treatment of servo loops.

Choice: PID servo loop

$C(p) = KP + \frac{KI}{p} + \frac{KD \times p}{(\tau_d \times p + 1)} \approx KP + \frac{KI}{p} + KD \times p$, where p is the Laplace variable.

Mechanical transfer function: $H(p) = \frac{1}{p^2}$ (driver transfer function disregarded)

Assumptions

- Position closed loop time constants (real and resonance), $\tau_1 = \tau_2$ (s) is between 4 ms and 8 ms (20 Hz and 40 Hz) depending on the first mechanical resonance.
- The damping factor of the closed loop to avoid overshoots (see closed transfer function numerator) is: $\zeta = 1.25$

Notice: $\tau = \frac{1}{2 \times \pi \times f}$

PID parameters

The closed loop transfer function is:

$$T(p) \approx \frac{\frac{KD}{KI} \times p^2 + \frac{KP}{KI} \times p + 1}{\frac{1}{KI} \times p^3 + \frac{KD}{KI} \times p^2 + \frac{KP}{KI} \times p + 1}$$

$$\approx \frac{\left(2 \times \zeta \times \tau_1 \times \tau_2 + \tau_2^2\right) \times p^2 + \left(\tau_1 + 2 \times \zeta \times \tau_2\right) \times p + 1}{\left(\tau_1 \times p + 1\right) \times \left(\tau_2^2 \times p^2 + 2 \times \zeta \times \tau_2 + 1\right)}$$

Identification with $\left(\tau_1 \times p + 1\right) \times \left(\tau_2^2 \times p^2 + 2 \times \zeta \times \tau_2 + 1\right)$ results in:

- PID servo loop proportional gain: $KP = \frac{\tau_1 + 2 \times \zeta \times \tau_2}{\tau_1 \times \tau_2^2} = 1.4e^5$ with $\tau_1 = \tau_2 = 5$ ms
- PID servo loop integral gain: $KI = \frac{1}{\tau_1 \times \tau_2^2} = 8e^6$ with $\tau_1 = \tau_2 = 5ms$
- PID servo loop derivative gain: $KD = \frac{2 \times \zeta \times \tau_1 \times \tau_2 + \tau_2^2}{\tau_1 \times \tau_2^2} = 7e^2$ with $\tau_1 = \tau_2 = 5$ ms
- PID integral saturation value: default $KS = 0.8$
- PID integration time: $\text{IntegrationTime} = 1e^{99} s$ (full integration)
- PID derivative filter cut off frequency: $\text{DerivativeCutOffFrequency} = 5000$ Hz (maximum)
- acceleration feedforward gain: $\text{KFeedForwardAcceleration} = 1$

Feed Forward (*PIDFFAcceleration*) Variable PID parameters entries for the configuration file:

- ***G K P*** — variable PID proportional gain multiplier
- ***G K I*** — variable PID integral gain multiplier
- ***G K D*** — variable PID derivative gain multiplier
- ***K Form*** — variable PID form coefficient

In addition to the classical gains of the PID servo loop, the XPS controller PID position servo loop features variable gain factors *GKP*, *GKD*, and *GKI*. These gains can be used to reduce settling times of systems exhibiting non-uniform behavior or to tighten the servo loop during the final segment of a move. For example, a stage with a high level of friction will have a response which is dependent on the size of the move: friction is negligible for a large move, but becomes a predominant factor for small moves. For this reason, the required response of the system to reach the commanded position is not the same for small and large moves. The optimum values of PID parameters for small moves are often higher than the optimum values for large moves. Users that do not want to set individual PID gains for different size motions can benefit from variable PID gains. Variable gains are driven by the distance between the target position and the current position. They must be greater than -1.

The parameter PID form coefficient value *KForm* (units) defines the relationship between the distance to the target and the change of the PID gains:

$$K_x = \left(1 + GK_x \bullet \frac{KForm}{|TagetPosition - CurrentPosition| + KForm)}\right) \bullet K_x$$

It must be greater than or equal to zero.

The smaller the variable PID form coefficient value is, the sharper the change of the PID gains.

GKp = 10 Kp = 2
Target Position = 0
Encoder Position = -100 to 100

$$Kp_{(\text{Kform, Encoder Position})} = \left[1 + GK \cdot \left(\frac{\text{Kform}}{|\text{Target Position - Encoder Position}| + \text{Kform}}\right)\right] \cdot Kp$$

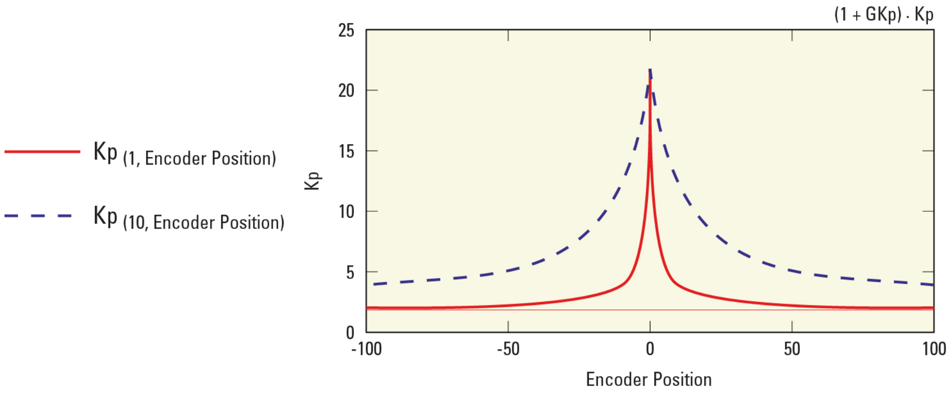


Figure 99: Influence of variable gains.

The default setting for these parameters is 0 which disables the variable gains.

Dead Band Threshold

The servo loop dead band threshold sets the dead band value of the position loop. When set to a value other than zero, the position loop is disabled when the following error is less than the value for the dead band threshold AND the theoretical motion is done. In some cases, this can avoid oscillations of stages with backlash or friction. It can also reduce stage settling times, but may result in residual error from the target position. It must be greater than or equal to zero. The default value for this parameter is 0, which disables this feature.

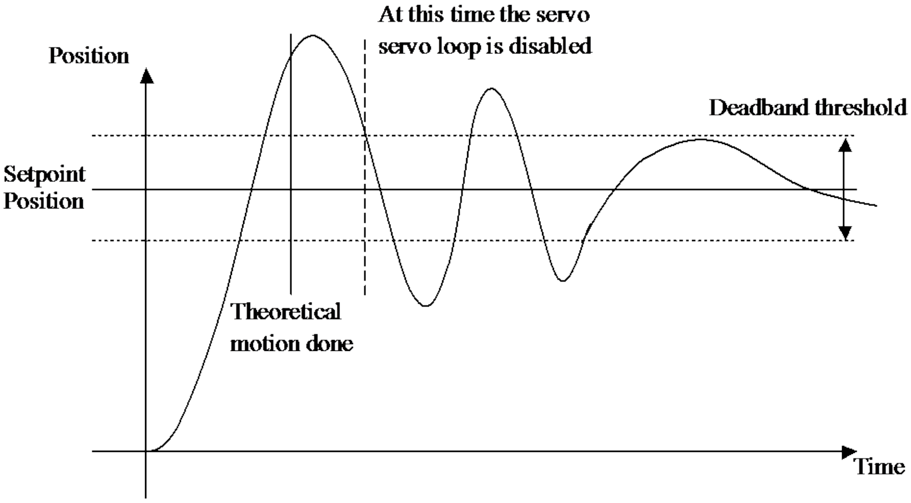


Figure 100: Deadband threshold.

Feed Forward (*PIDFFAcceleration*) Notch filters parameters entries for the configuration file:

- ***Notch Frequency 1*** and ***Notch Frequency 2*** — first and second notch filter center frequency
- ***Notch Bandwidth 1*** and ***Notch Bandwidth 2*** — first and second notch filter bandwidth:
- ***Notch Gain 1*** and ***Notch Gain 2*** — first and second notch filter gain

The output of the position servo loop is filtered by two notch filters. These filters can be used to avoid the excitation of specific frequencies. They are defined by their center frequencies *NotchFrequency<n°>* (Hz), bandwidths *NotchBandwidth<n°>* (Hz) and their gains *NotchGain<n°>*. The frequencies and bandwidths must be greater than or equal to zero (filter disabled) and less than or equal to half of the servo loop frequency (8 kHz). The gain must be greater than or equal to zero. The default value for these parameters is 0, which disables this feature.

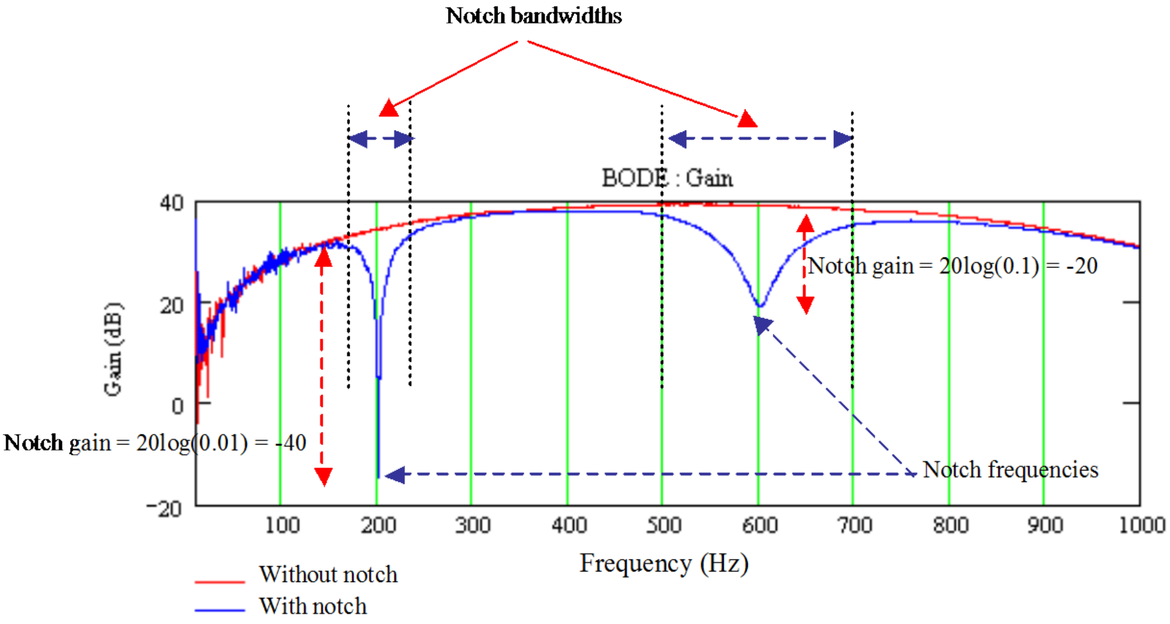


Figure 101: Notch filters.

23.8.5 PIDFFVelocity

Corrector Type: PIDFFVelocity

This servo loop type is used when a constant value applied to the driver results in constant velocity of the stage, for instance a DC motor with tachometer connected to a driver with internal speed loop. This servo loop type features a parallel PID servo loop with a feed forward velocity and two notch filters.

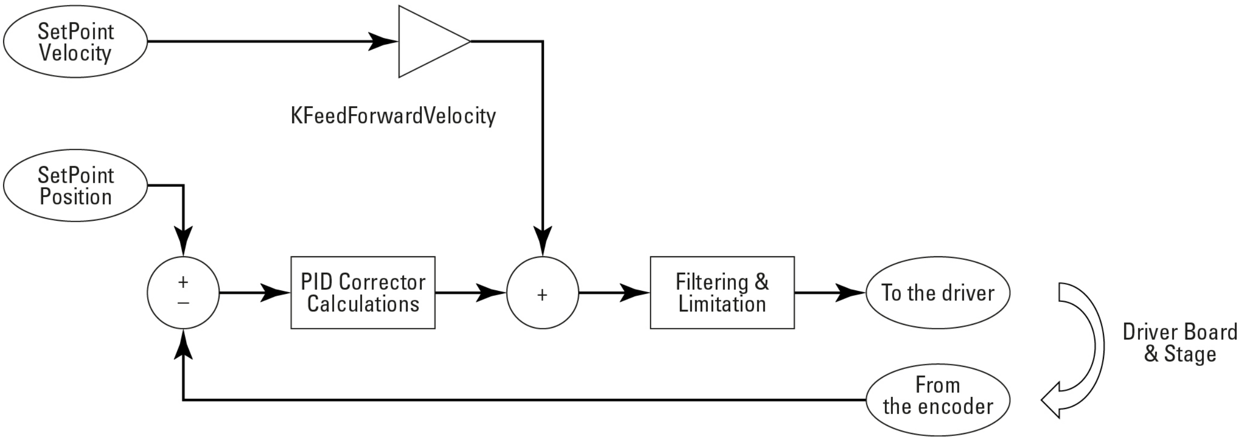


Figure 102: PIDFFVelocity corrector.

Closed Loop Status

The position servo loop status parameter sets the position servo loop either to open loop i.e. without feedback of a position encoder, or to closed loop, i.e. with feedback from a position encoder. The default value for this parameter is *closed*.

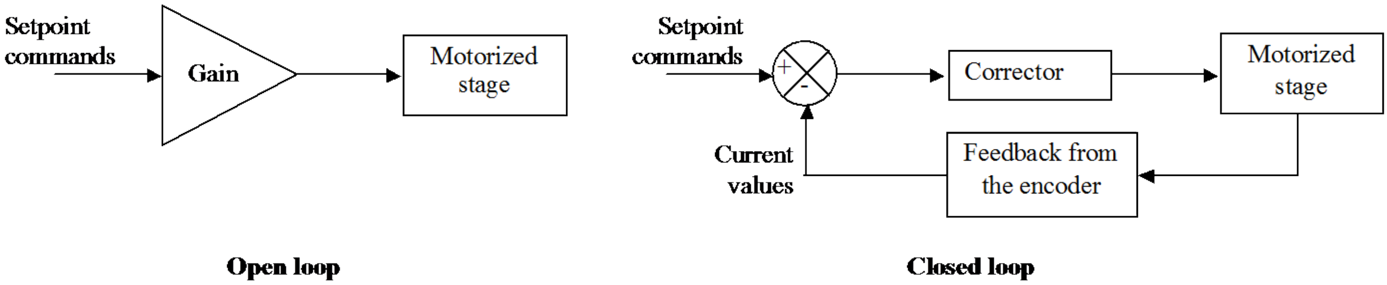


Figure 103: Open and closed loop.

Fatal Following Error

The value for the fatal following error sets the maximum allowed following error of the positioner before generating an error response from the controller. This error is defined as the absolute value of the difference between the setpoint position and the current position. This value is calculated each servo cycle. A following error that exceeds this value will generate the corresponding error code and action. It must be greater than zero.

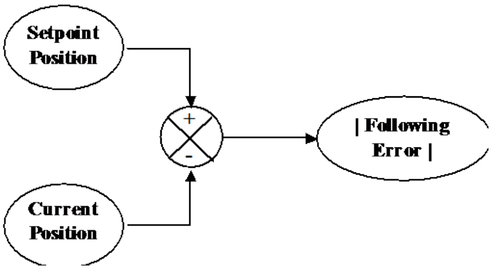


Figure 104: Following error.

Feed Forward (*PIDFFVelocity*) PID Parameters entries for the configuration file:

- ***K P*** — PID servo loop proportional gain
- ***K I*** — PID servo loop integral gain
- ***K D*** — PID servo loop derivative gain
- ***K S*** — PID integral saturation value
- ***Integration Time*** — PID integration time:
- ***Derivative Cut Off Frequency*** — PID derivative filter cut off frequency
- ***K Feed Forward Velocity*** — Velocity feedforward gain

The PID servo loop parameters *KP*, *KI*, *KD* and *KFeedForwardVelocity* define the bandwidth of the servo loop. They must be greater than or equal to zero.

The PID integral saturation value *KS* sets the limit of the integral part of the PID servo loop that is applied to the total servo loop output. The value for *KS* must be between 0 and 1.

The PID *IntegrationTime* (seconds) defines the time span for integration of the residual errors. A small value limits the effect of the integral gain *KI*. The value in seconds must be greater than or equal to the servo loop period (125 µs).

The PID *DerivativeFilterCutOffFrequency* (Hz) sets the cut-off frequency of the derivative filter. It can be used to reduce the noise introduced by the numerical derivation of the following error. It must be greater than or equal to zero (zero means filter is disabled) and less than or equal to half of the servo loop frequency (8 kHz).

All parameters will have an impact on system performance and should be set together. It should be noted that parameters are best set for desired performance according the requirements of the motion application (small following error during motion, short settling time after a displacement…). This document will present context specific information on setting PID parameters, but is not intended to be a tutorial on servo loops. Please refer to the common literature for a general treatment of servo loops.

Choice: PI servo loop (Kd = 0)

$C(p) = KP + \frac{KI}{p}$, where *p* is the Laplace variable.

Mechanical transfer function: $H(p) = \frac{1}{p \times (\tau_v \times p + 1)}$, where $\tau_v$is the time constant of the velocity transfer function (s). See section 23.10.6: “DRV01AnalogVelocity (with tachometer feedback)”.

Assumption

The position closed loop resonance time constant τ_p is much greater than τ_v. This allows disregarding the velocity servo loop transfer function. By experience, τ_p is about 10-20 ms for most screw driven stages and τ_p / τ_v ranges between 10 and 20. The damping factor of the closed loop $\zeta$is set to $\zeta = 1.25$ to avoid overshoots, see closed transfer function numerator.

Notice: $\tau = \frac{1}{2 \times \pi \times f}$

PID parameters

By taken into account that $\tau_p >> \tau_v$, the closed loop transfer function is:

$$T(p) \approx \frac{\frac{KP}{KI}\times p+1}{\frac{1}{KI}\times p^2+\frac{KP}{KI}\times p+1}$$
$$\approx \frac{2\times\zeta\times\tau_p\times p+1}{\tau_p^2\times p^2+2\times\zeta\times\tau_p\times p+1}$$

This results in:

- Closed loop cut off frequency: $Fc=\frac{\sqrt{1+2\times\zeta^2+\sqrt{\left(1+2\times\zeta^2\right)^2+1}}}{2\times\pi\times\tau_p}$
- PID servo loop proportional gain: $KP=\frac{2\times\zeta}{\tau_p}=156.25$ with $\tau_p=16ms$
- PID servo loop integral gain: $KI=\frac{1}{\tau_p^2}=3906.25$ with $\tau_p=16ms$
- PID servo loop derivative gain: $KD=0$
- PID integral saturation value: default $KS=0.8$
- PID integration time: $\text{IntegrationTime}=1e^{99}s$ (full integration)
- PID derivative filter cut off frequency: $\text{DerivativeCutOffFrequency}=0$ (disabled)
- velocity feedforward gain: $\text{KFeedForwardVelocity}=1$
- Variable PID parameters

Feed Forward (*PIDFFVelocity*) Variable PID parameters entries for the configuration file:

- ***G K P*** — variable PID proportional gain multiplier
- ***G K I*** — variable PID integral gain multiplier
- ***G K D*** — variable PID derivative gain multiplier
- ***K Form*** — variable PID form coefficient

In addition to the classical gains of the PID servo loop, the XPS controller PID position servo loop features variable gain factors *GKP*, *GKD*, and *GKI*. These gains can be used to reduce settling times of systems exhibiting non-uniform behavior or to tighten the servo loop during the final segment of a move. For example, a stage with a high level of friction will have a response which is dependent on the size of the move: friction is negligible for a large move, but becomes a predominant factor for small moves. For this reason, the required response of the system to reach the commanded position is not the same for small and large moves. The optimum values of PID parameters for small moves are often higher than the optimum values for large moves. Users that do not want to set individual PID gains for different size motions can benefit from variable PID gains. Variable gains are driven by the distance between the target position and the current position. They must be greater than -1.

The parameter PID form coefficient value *KForm* (units) defines the relationship between the distance to the target and the change of the PID gains:

$$Kx=\left(1+GKx\bullet\frac{KForm}{|TagetPosition-CurrentPosition|+KForm)}\right)\bullet Kx$$

It must be greater than or equal to zero.

The smaller the variable PID form coefficient value is, the sharper the change of the PID gains.

GKp = 10   Kp = 2
Target Position = 0
Encoder Position = -100 to 100

$$Kp_{(Kform,\ Encoder\ Position)} = \left[1 + GK \cdot \left(\frac{Kform}{|\,Target\ Position - Encoder\ Position\,| + Kform}\right)\right] \cdot Kp$$

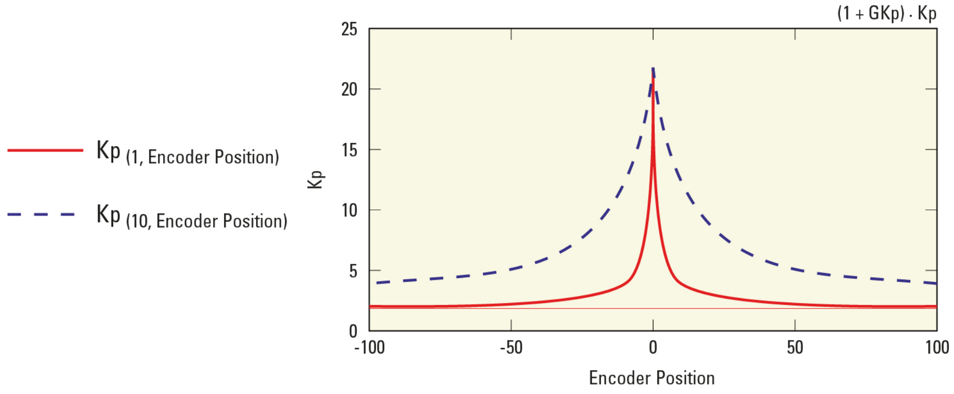


Figure 105: Influence of variable gains.

Dead Band Threshold

The servo loop dead band threshold sets the dead band value of the position control loop. When set to a value other than zero, the position loop is disabled when the following error is less than the value for the dead band threshold AND the theoretical motion is done. In some cases, this can avoid oscillations of stages with backlash or friction. It can also reduce stage settling times, but may result in a residual error from the target position. It must be greater than or equal to zero.

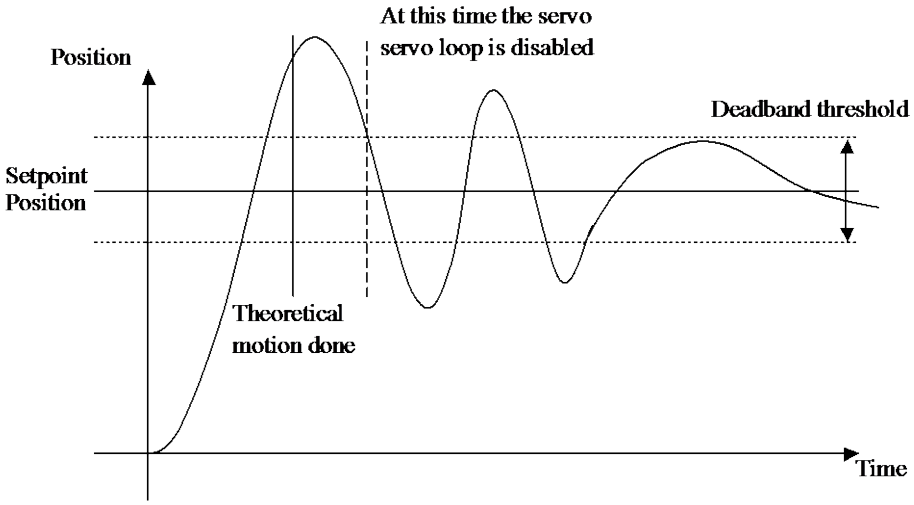


Figure 106: Deadband threshold.

Feed Forward (*PIDFFAcceleration*) Notch filters parameters entries for the configuration file:

- ***Notch Frequency 1*** and ***Notch Frequency 2*** — first and second notch filter center frequency
- ***Notch Bandwidth 1*** and ***Notch Bandwidth 2*** — first and second notch filter bandwidth:
- ***Notch Gain 1*** and ***Notch Gain 2*** — first and second notch filter gain

The output of the position servo loop is filtered by two notch filters. These filters can be used to avoid the excitation of specific frequencies. They are defined by their center frequencies *NotchFrequency<n°>* (Hz), bandwidths *NotchBandwidth<n°>* (Hz) and their gains *NotchGain<n°>*. The frequencies and bandwidths must be greater than or equal to zero (filter disabled) and less than or equal to half of the servo loop frequency (8 kHz). The gain must be greater than or equal to zero. The default value for these parameters is 0, which disables this feature.

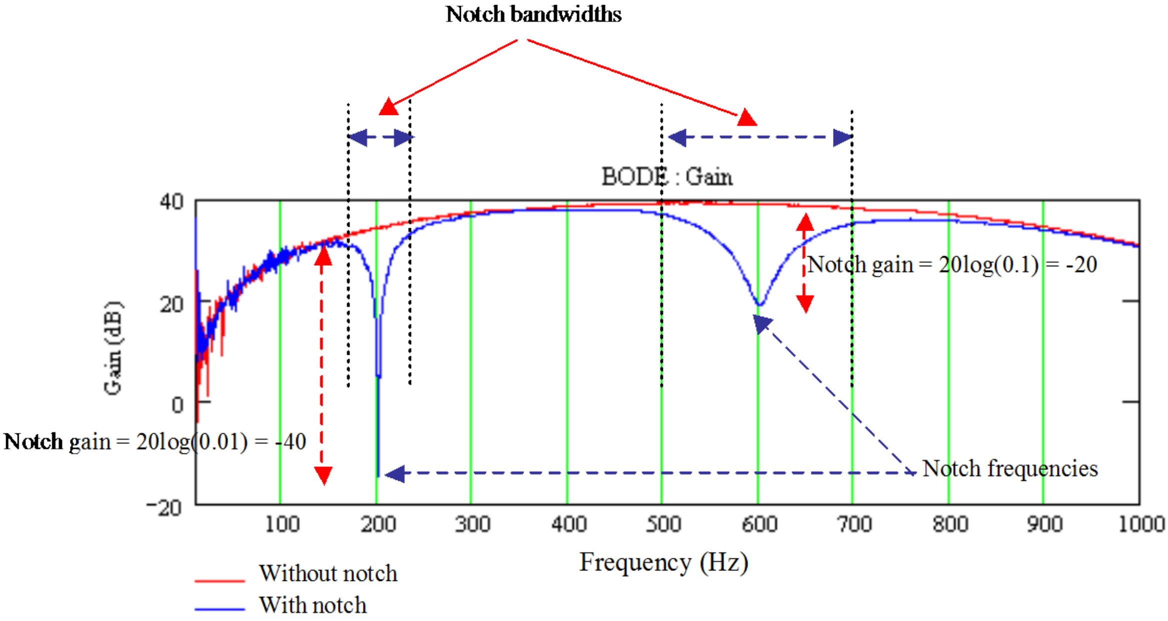


Figure 107: Notch filters.

23.8.6 PIPosition

Corrector Type: PIPosition

This servo loop type is used when the position servo loop directly outputs a position value. This servo loop type features a parallel PI servo loop with two notch filters.

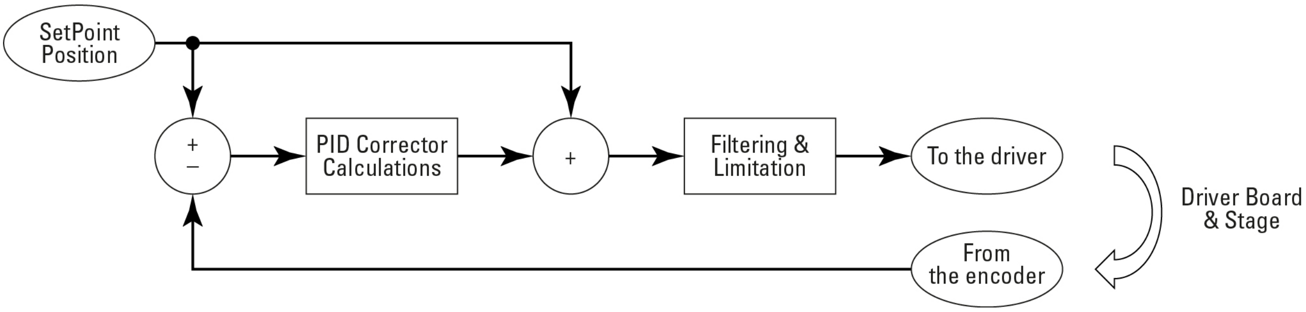


Figure 108: PIPosition corrector.

Closed Loop Status

The position servo loop status parameter sets the position servo loop either to open loop i.e. without feedback of a position encoder, or to closed loop, i.e. with feedback from a position encoder. The default value for this parameter is *closed*.

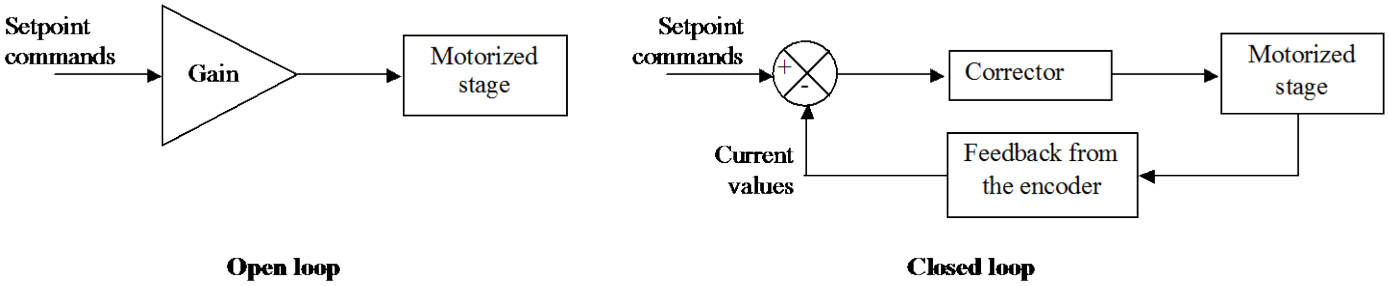


Figure 109: Open and closed loop.

Fatal Following Error

The value for the fatal following error sets the maximum allowed following error of the positioner before generating an error response from the controller. This error is defined as the absolute value of the difference between the setpoint position and the current position. This value is calculated each servo cycle. A following error that exceeds this value will generate the corresponding error code and action. It must be greater than zero.

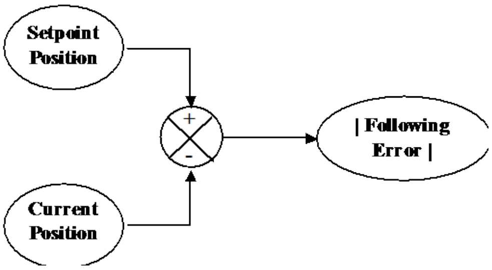


Figure 110: Following error.

Position (*PIPosition*) PI Parameters entries for the configuration file:

- ***K P*** — PI servo loop proportional gain:
- ***K I*** — PI servo loop integral gain
- ***Integration Time*** — PI integration time

The PI servo loop parameters *KP* and *KI* define the bandwidth of the servo loop. They must be greater than or equal to zero.

The PI *IntegrationTime* (seconds) defines the time span for integration of the residual errors. A small value limits the effects of the integral gain *KI*. It must be greater than or equal to the servo loop period (125 µs).

All parameters will have an impact on system performance and should be set together. It should be noted that parameters are best set for desired performance according the requirements of the motion application (small following error during trajectory motion, short settling time after a displacement, …). This document will present context specific information on setting PID parameters, but is not intended to be a tutorial on servo loops. Please refer to the common literature for a general treatment of servo loops.

Choice: I servo loop (no value for *KP*)

$C(p)=\frac{KI}{p}$, where p is the Laplace variable.

Mechanical transfer function: $H(p)=1$

Assumption

The position closed loop real time constant τ_p (s) is between 40 ms and 80 ms (2 Hz to 4Hz)

Notice: $\tau = \frac{1}{2 \times \pi \times f}$

PI parameters

The closed loop transfer function is:

$$T(p) = \frac{1}{\frac{1}{KI} \times p + 1}$$

This results in:

- PI servo loop proportional gain: $KP = 0$
- PI servo loop integral gain: $KI = \frac{1}{\tau_p} = 12.5$ with $\tau_p = 80ms$
- PID integration time: $\mathrm{IntegrationTime} = 1e^{99}s$ (full integration)

Dead Band Threshold

The servo loop dead band threshold sets the dead band value of the position control loop. When set to a value other than zero, the position loop is disabled when the following error is less than the value for the dead band threshold AND the theoretical motion is done. In some cases, this can avoid oscillations of stages with backlash or friction. It can also reduce stage settling times, but may result in a residual error from the target position. It must be greater than or equal to zero.

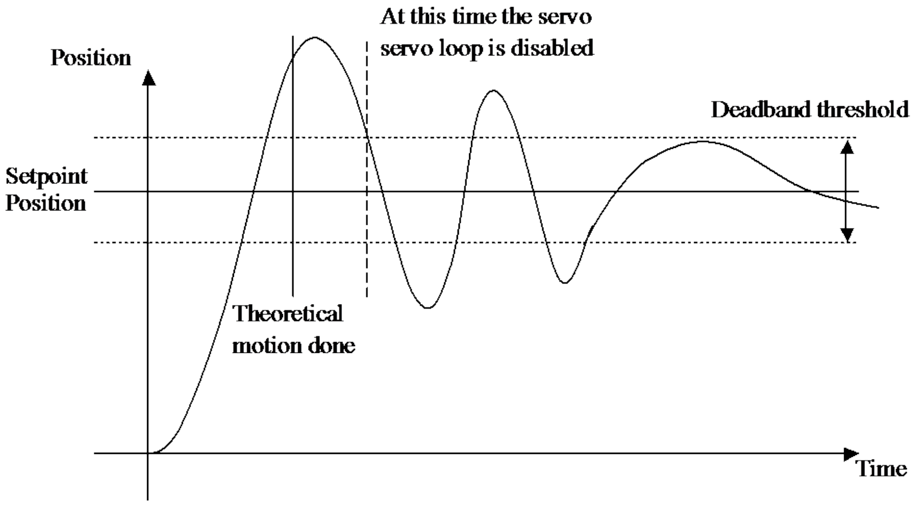


Figure 111: Deadband threshold.

Position (*PIPosition*) Notch Filters Parameters entries for the configuration file:

- ***Notch Frequency 1*** and ***Notch Frequency 2*** — first and second notch filter center frequency
- ***Notch Bandwidth 1*** and ***Notch Bandwidth 2*** — first and second notch filter bandwidth:
- ***Notch Gain 1*** and ***Notch Gain 2*** — first and second notch filter gain

The output of the position servo loop is filtered by two notch filters. These filters can be used to avoid the excitation of specific frequencies. They are defined by their center frequencies *NotchFrequency<n°>* (Hz), bandwidths *NotchBandwidth<n°>* (Hz) and gains *NotchGain<n°>*. The frequencies and bandwidths must be greater than or equal to zero (filter disabled) and less than or equal to half of the servo loop frequency (8 kHz). The gain must be greater than or equal to zero. The default setting for these parameters is 0 which disables the filters.

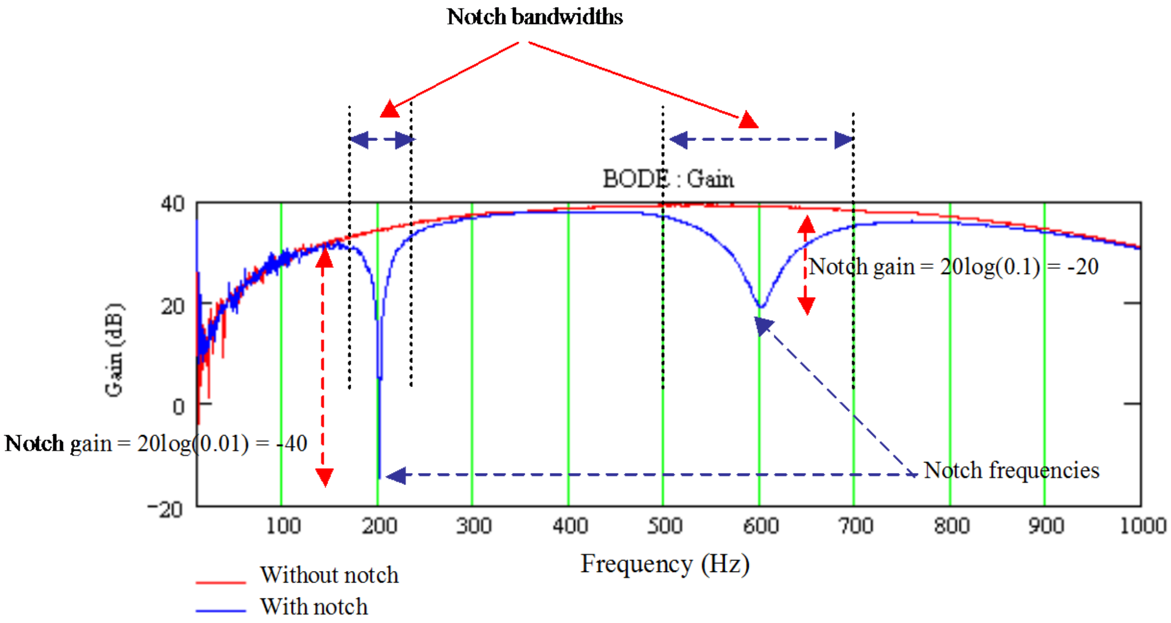


Figure 112: Notch filters.

23.9 Motor Driver Interface

In this configuration category, users are building the Drive command interface parameters section of the stages.ini file.

Example:

```
; --- Driver command interface parameters
; --- <MotorDriverInterface.AnalogSin120AccelerationLMI>
MotorDriverInterface = AnalogSin120AccelerationLMI
DelayAfterMotorOnToSetClosedLoop = 0.05 ; s
ScalingAcceleration = 30590 ; Unit/s²
AccelerationLimit = 13960 ; Unit/s²
MagneticTrackPeriod = 30 ; Unit
MagneticTrackPositionAtHomeMode = Disabled
MagneticTrackPositionAtHome = 0 ; Units
InitializationAccelerationLevel = 7 ; Percent
InitializationCycleDuration = 5 ; s
```

The XPS controller features 7 different motor driver interfaces:

- Velocity control
- Voltage control
- Acceleration control
- Sine/cosine position control
- Position control
- 60/90/120 deg UV phase acceleration control
- 60/90/120 deg UV phase dual output acceleration control

The choice of the motor driver interface depends on the position servo loop type, the driver type and the motor type.

23.9.1 NoMotorInterface

Motor Driver Interface Type: NoMotorInterface

This driver command interface is only used for Dummy configuration.

There are no additional parameters to set for this motor interface type.

23.9.2 AnalogAcceleration

Motor Driver Interface Type: AnalogAcceleration

This driver command interface is used when the output of the position servo loop refers to an acceleration value and when the driver input is an analog acceleration value.

Scaling Acceleration

The stage acceleration at maximum command, *ScalingAcceleration* (units/s²), scales the output of the controller. This value corresponds to the theoretical acceleration reached by the stage with a +10 V input signal to the driver. The value for the *ScalingAcceleration* is stage and driver dependent and must be greater than zero.

See section 23.10.10: "XPS-DRV03 for acceleration control (DRV03AnalogAcceleration)" for XPS-DRV03 driver board settings.

Delay After Motor On To Set Closed Loop (Not used by standard XPS-Q firmware)

- In seconds.

This delay enables to wait after a motor ON to give time with certain external motor amplifiers to stabilize before the controller closes the position control loop. The default value is 0.050 s.

Acceleration Limit

This parameter should not be confused with the *profile generator maximum acceleration*, which is the maximum acceleration a stage can be commanded to move (see section 23.3.1: "Type: Sgamma"). In order to decrease following errors, a positioner must be allowed to move at greater acceleration than the *profile generator maximum acceleration*. Its maximum value is defined by the maximum allowed stage acceleration, *AccelerationLimit* (units/s²). The higher the dynamic bandwidth of a system, the greater the margin between *maximum allowed stage acceleration* and the *profile generator maximum acceleration*.

The value for the *maximum allowed stage acceleration* must be less than the *stage acceleration at maximum command* and greater than or equal to the *profile generator maximum acceleration.* The recommended value is 1.5 times the value of the *profile generator maximum acceleration*.

Initialization Acceleration Level (percent)

The following relation: is the acceleration used during the stage auto-scaling process.

$$\text{InitializationAcceleration} = \frac{\text{ScalingAcceleration} \bullet \text{InitializationAccelerationLevel}}{100}$$

[InitializationAcceleration] = units/s²

The recommended starting value for this parameter is equal to 20% of the scaling acceleration. If the auto-scaling process does not work properly with this setting (for example, an acceleration that is too low during auto-scaling combined with bad efficiency of the drive chain, could result in failure of the auto-scaling), this value has to be increased step by step. If the displacement during auto-scaling is too large, the *stage initialization acceleration* can be decreased.

23.9.3 AnalogPosition

Motor Driver Interface Type: AnalogPosition

This driver command interface is used when the output of the position servo loop is a position value and when the driver input is an analog position value. This is the case for some drivers of piezo stages or galvanometric mirrors or voice coils.

Delay After Motor On To Set Closed Loop (Not used by standard XPS-Q firmware)

- In seconds.

This delay enables to wait after a motor ON to give time with certain external motor amplifiers to stabilize before the controller closes the position control loop.

Minimum Target Position Voltage

The command voltage at minimum target position, *MinimumTargetPositionVoltage* (V), sets the analog output voltage of the controller when the stage is at its minimum travel limit. This value must be between ±10 V and less than the maximum target position voltage.

Maximum Target Position Voltage

The command voltage at maximum target position, *MaximumTargetPositionVoltage* (V), sets the analog output voltage of the controller when the stage is at its maximum travel limit. This value must be between ±10 V and greater than the minimum target position voltage.

23.9.4 AnalogPositionPiezo

Motor Driver Interface Type: AnalogPositionPiezo

This type of motor driver interface is used specially for piezo stages (DRVP1, …). It is compatible only with *NoEncoderPosition* or *PIPosition* corrector type.

Delay After Motor On To Set Closed Loop (Not used by standard XPS-Q firmware)

- In seconds.

This delay enables to wait after a motor ON to give time with certain external motor amplifiers to stabilize before the controller closes the position control loop.

Default value: 0.050 s.

23.9.5 AnalogSin60Acceleration

Motor Driver Interface Type: AnalogSin60Acceleration

This motor driver interface is used when the output of the position servo loop refers to a modulated acceleration value and when the driver inputs are analog modulated signals. The modulation is based on the position and suitable for synchronous linear or rotary brushless motors. Use this Motor interface type for motors whose phase difference between the two output channels is 60° between phases.

Note: This interface type does not require Hall sensors for the motor phase initialization. This is done by a Newport patented process that determines the coil positions relative to the magnetic track solely based on the encoder feedback, and avoids major motion during initialization.

Delay After Motor On To Set Closed Loop (Not used by standard XPS-Q firmware)

- In seconds.

This delay enables to wait after a motor ON to give time with certain external motor amplifiers to stabilize before the controller closes the position control loop.

Default value: 0.050 s.

Scaling Acceleration

The stage acceleration at maximum command, *ScalingAcceleration* (units/s²), scales the analog output of the controller. This value corresponds to the theoretical acceleration reached by the stage with the maximum voltage input to the driver (+10 V for the amplitude of the sine signal). The *ScalingAcceleration* must be greater than zero and is stage and driver dependent. See also section 23.10.8 for parameter definition with the XPS driver board XPS-DRV02.

Acceleration Limit

This parameter should not be confused with the *profile generator maximum acceleration*, which is the maximum acceleration a stage can be commanded to move, see section 23.4.5: "Encoder Board (StandardLimitAndLimitEncoderPlug)". In order to decrease following errors, a positioner must be allowed to move at greater acceleration than the *profile generator maximum acceleration*. Its maximum value is defined by the maximum allowed stage acceleration, *AccelerationLimit* (units/s²). The higher the dynamic bandwidth of a system, the greater the margin between *maximum allowed stage acceleration* and the *profile generator maximum acceleration*.

The value for the *maximum allowed stage acceleration* should be set in relation to the application and not solely in relation to the motor capability. For correct parameter determination with the driver board XPS-DRV02, see also section 23.10.8: "DRV02". The value must be less than the *stage acceleration at maximum command* and greater than or equal to the *profile generator maximum acceleration.*

Magnetic Track Period

The stage displacement per motor period, *MagneticTrackPeriod* (units), is the magnetic pitch between two consecutive north poles of the magnetic track. Its value must be greater than zero.

Magnetic Track Position At Home Mode (Not used by standard XPS-Q firmware)

When *MagneticTrackPositionAtHome* is Enabled, the phasing of the motor commutation (previously detected during initialization sequence) is adjusted to the parameter *MagneticTrackPositionAtHome* during the home sequence.

Magnetic Track Position At Home (Not used by standard XPS-Q firmware)

The phasing of the motor commutation (previously detected during initialization sequence) is adjusted to the parameter *MagneticTrackPositionAtHome* during the home sequence, when *MagenticTrackPositionAtHome* is Enabled.

Initialization Acceleration Level (percent)

This parameter is the acceleration used during the motor initialization and the stage auto-scaling processes with the following relation:

$$\mathrm{InitializationAcceleration} = \frac{\mathrm{ScalingAcceleration} \bullet \mathrm{InitializationAccelerationLevel}}{100}$$

[InitializationAcceleration] = units/s²

The recommended starting value for this parameter is 20% of the scaling acceleration and must be greater than or equal to the maximum acceleration of the profile generator, see section 23.4.5: "Encoder Board (StandardLimitAndLimitEncoderPlug)". If the initialization or auto-scaling process does not work properly with this setting (for example, an acceleration that is too low during auto-scaling combined with bad efficiency of the drive chain, could result in failure of the initialization or auto-scaling), this value has to be increased step by step. If the displacement during initialization or auto-scaling is too large, the *stage initialization acceleration* can be decreased.

Initialization Frequency

Default value: 50 Hz.

23.9.6 AnalogSin60AccelerationLMI

Motor Driver Interface Type: AnalogSin60AccelerationLMI

This motor driver interface is used when the output of the position servo loop refers to a modulated acceleration value and when the driver inputs are analog modulated signals. The modulation is based on the position and suitable for synchronous linear or rotary brushless motors. Use this Motor interface type for motors whose phase difference between the two output channels is 60° between phases. Large Move Initialization (LMI) will be enabled under this configuration allowing for a larger, user defined move during the initialization process as compared to standard initialization process.

NOTE

This interface type does not require Hall sensors for the motor phase initialization. This is done by a Newport patented process that determines the coil positions relative to the magnetic track solely based on the encoder feedback, and avoids major motion during initialization.

Initialization Cycle Duration

The stage initialization cycle duration, InitializationCycleDuration (s), is the time period of the motor initialization process.

This parameter is used only with the LMI (Large Move Initialization) initialization process.

The recommended starting value of this parameter varies from 1 to 20 seconds and is inversely related to the stage friction. For example, a stage with minimal friction requires a longer stage initialization cycle to stabilize the stage during initialization.

For all other configuration file parameters refer to section 23.9.5:* “AnalogSin60Acceleration*”.

23.9.7 AnalogDualSin60Acceleration

Motor Driver Interface Type: AnalogDualSin60Acceleration

This motor driver interface is used when the output of the position servo loop refers to a modulated acceleration value balanced on two controller axes and when both driver inputs are analog modulated signals. This is a specific interface for driving two motors via two drivers in parallel. The modulation is based on the position and suitable for synchronous linear or rotary brushless motors. Use this Motor interface type for motors whose phase difference between the two output channels is 60° between phases.

NOTE

This interface does not require Hall sensors for the motor phase initialization. This is done by a Newport patented process that determines the coil positions relative to the magnetic track solely based on the encoder feedback, and avoids major motion during initialization.

Delay After Motor On To Set Closed Loop (Not used by standard XPS-Q firmware)

Default value: 0.050 s

Scaling Acceleration

The stage acceleration at maximum command value *ScalingAcceleration* (units/s²) is used to scale the analog output of the controller. This value corresponds to the theoretical acceleration reached by the stage with both motors when the maximum voltage (+10 V for the amplitude of the sine signal) is output by the controller. It must be greater than zero. See also section 23.10.8 for XPS-DRV02 driver board application.

Acceleration Limit

The stage acceleration at maximum command, ScalingAcceleration (units/s²), scales the analog output of the controller. This value corresponds to the theoretical acceleration reached by the stage with a maximum voltage input to the driver (+10 V for the amplitude of the sine signal). The ScalingAcceleration must be greater than zero and is stage and driver dependent. See also section 23.10.8 for parameter definition with the XPS driver board XPS-DRV02.

Magnetic Track Period

The stage displacement per motor period, *MagneticTrackPeriod* (units), is the magnetic pitch between two consecutive north poles of the magnetic track. This value must be greater than zero.

Magnetic Track Position At Home Mode (Not used by standard XPS-Q firmware)

When *MagneticTrackPositionAtHome* is Enabled, the phasing of the motor commutation (previously detected during initialization sequence) is adjusted to the parameter *MagneticTrackPositionAtHome* during the home sequence.

Magnetic Track Position At Home (Not used by standard XPS-Q firmware)

The phasing of the motor commutation (previously detected during initialization sequence) is adjusted to the parameter *MagneticTrackPositionAtHome* during the home sequence, when *MagenticTrackPositionAtHome* is Enabled.

Initialization Acceleration Level (percent)

This parameter is the acceleration used during the motor initialization and the stage auto-scaling processes with the following relation:

$$\mathrm{InitializationAcceleration} = \frac{\mathrm{ScalingAcceleration} \bullet \mathrm{InitializationAccelerationLevel}}{100}$$

[InitializationAcceleration] = units/s²

The recommended starting value for this parameter is 20% of the scaling acceleration and must be greater than or equal to the maximum acceleration of the profile generator (see section 23.3.1: “Type: Sgamma”). If the initialization or auto-scaling process does not work properly with this setting (for example, an acceleration that is too low during auto-scaling combined with bad efficiency of the drive chain, could result in failure of the initialization or auto-scaling), this value has to be increased step by step. If the displacement during initialization or auto-scaling is too large, the *stage initialization acceleration* can be decreased.

First Motor Balance and Second Motor Balance

The motor command input balance, *FirstMotorBalance* and *SecondMotorBalance*, scale the outputs of the two drivers to apply the acceleration to the center of gravity. The values for both parameters must be between 0 and 1.

Initialization Frequency

Default value: 50 Hz.

23.9.8 AnalogDualSin60AccelerationLMI

Motor Driver Interface Type: AnalogDualSin60AccelerationLMI

This motor driver interface is used when the output of the position servo loop refers to a modulated acceleration value balanced on two controller axes and when both driver inputs are analog modulated signals. This is a specific interface for driving two motors via two drivers in parallel. The modulation is based on the position and suitable for synchronous linear or rotary brushless motors. Use this Motor interface type for motors whose phase difference between the two output channels is 60° between phases. Large Move Initialization (LMI) will be enabled under this configuration allowing for a larger, user defined move during the initialization process as compared to standard initialization process.

Note: This interface does not require Hall sensors for the motor phase initialization. This is done by a Newport patented process that determines the coil positions relative to the magnetic track solely based on the encoder feedback, and avoids major motion during initialization.

Initialization Cycle Duration

The stage initialization cycle duration, *InitializationCycleDuration* (s), is the time period of the motor initialization process.

This parameter is used only with the LMI (Large Move Initialization) initialization process.

The recommended starting value of this parameter varies from 1 to 20 seconds and is inversely related to the stage friction. For example, a stage with minimal friction requires a longer stage initialization cycle to stabilize the stage during initialization.

For all other configuration file parameters refer to section 23.9.7: "*AnalogDualSin60Acceleration*".

23.9.9 AnalogSin90Acceleration

Motor Driver Interface Type: AnalogSin90Acceleration

This motor driver interface is used when the output of the position servo loop refers to a modulated acceleration value and when the driver inputs are analog modulated signals. The modulation is based on the position and suitable for synchronous linear or rotary brushless motors. Use this Motor interface type for motors whose phase difference between the two output channels are 90° between phases for two phase motors.

Note: This interface type does not require Hall sensors for the motor phase initialization. This is done by a Newport patented process that determines the coil positions relative to the magnetic track solely based on the encoder feedback, and avoids major motion during initialization.

For all other configuration file parameters refer to section 23.9.5: "*AnalogSin60Acceleration*".

23.9.10 AnalogSin90AccelerationLMI

Motor Driver Interface Type: AnalogSin90AccelerationLMI

This motor driver interface is used when the output of the position servo loop refers to a modulated acceleration value and when the driver inputs are analog modulated signals. The modulation is based on the position and suitable for synchronous linear or rotary brushless motors. Use this Motor interface type for motors whose phase difference between the two output channels are 90° between phases for two phase motors. Large Move Initialization (LMI) will be enabled under this configuration allowing for a larger, user defined move during the initialization process as compared to standard initialization process.

NOTE

This interface type does not require Hall sensors for the motor phase initialization. This is done by a Newport patented process that determines the coil positions relative to the magnetic track solely based on the encoder feedback, and avoids major motion during initialization.

Initialization Cycle Duration

The stage initialization cycle duration, *InitializationCycleDuration* (s), is the time period of the motor initialization process.

This parameter is used only with the LMI (Large Move Initialization) initialization process.

The recommended starting value of this parameter varies from 1 to 20 seconds and is inversely related to the stage friction. For example, a stage with minimal friction requires a longer stage initialization cycle to stabilize the stage during initialization.

For all other configuration file parameters refer to section 23.9.5:* “AnalogSin60Acceleration*”.

23.9.11 AnalogDualSin90Acceleration

Motor Driver Interface Type: AnalogDualSin90Acceleration

This motor driver interface is used when the output of the position servo loop refers to a modulated acceleration value balanced on two controller axes and when both driver inputs are analog modulated signals. This is a specific interface for driving two motors via two drivers in parallel. The modulation is based on the position and suitable for synchronous linear or rotary brushless motors. Use this Motor interface type for motors whose phase difference between the two output channels are 90° between phases for two phase motors.

NOTE

This interface type does not require Hall sensors for the motor phase initialization. This is done by a Newport patented process that determines the coil positions relative to the magnetic track solely based on the encoder feedback, and avoids major motion during initialization.

For all other configuration file parameters refer to section 23.9.7:* “AnalogDualSin60Acceleration*”.

23.9.12 AnalogDualSin90AccelerationLMI

Motor Driver Interface Type: AnalogDualSin90AccelerationLMI

This motor driver interface is used when the output of the position servo loop refers to a modulated acceleration value balanced on two controller axes and when both driver inputs are analog modulated signals. This is a specific interface for driving two motors via two drivers in parallel. The modulation is based on the position and suitable for synchronous linear or rotary brushless motors. Use this Motor interface type for motors whose phase difference between the two output channels are 90° between phases for two phase motors. Large Move Initialization (LMI) will be enabled under this configuration allowing for a larger, user defined move during the initialization process as compared to standard initialization process.

NOTE

This interface type does not require Hall sensors for the motor phase initialization. This is done by a Newport patented process that determines the coil positions relative to the magnetic track solely based on the encoder feedback, and avoids major motion during initialization.

Initialization Cycle Duration

The stage initialization cycle duration, *InitializationCycleDuration* (s), is the time period of the motor initialization process.

This parameter is used only with the LMI (Large Move Initialization) initialization process.

The recommended starting value of this parameter varies from 1 to 20 seconds and is inversely related to the stage friction. For example, a stage with minimal friction requires a longer stage initialization cycle to stabilize the stage during initialization.

For all other configuration file parameters refer to section 23.9.7:* “AnalogDualSin60Acceleration*”.

23.9.13 AnalogSin120Acceleration

Motor Driver Interface Type: AnalogSin120Acceleration

This motor driver interface is used when the output of the position servo loop refers to a modulated acceleration value and when the driver inputs are analog modulated signals. The modulation is based on the position and suitable for synchronous linear or rotary brushless motors. Use this Motor interface type for motors whose phase difference between the two output channels are 120° between phases for three phase motors.

NOTE

This interface type does not require Hall sensors for the motor phase initialization. This is done by a Newport patented process that determines the coil positions relative to the magnetic track solely based on the encoder feedback, and avoids major motion during initialization.

For all other configuration file parameters refer to section 23.9.5:* “AnalogSin60Acceleration*”.

23.9.14 AnalogSin120AccelerationLMI

Motor Driver Interface Type: AnalogSin120AccelerationLMI

This motor driver interface is used when the output of the position servo loop refers to a modulated acceleration value and when the driver inputs are analog modulated signals. The modulation is based on the position and suitable for synchronous linear or rotary brushless motors. Use this Motor interface type for motors whose phase difference between the two output channels are 120° between phases for three phase motors. Large Move Initialization (LMI) will be enabled under this configuration allowing for a larger, user defined move during the initialization process as compared to standard initialization process.

NOTE

This interface type does not require Hall sensors for the motor phase initialization. This is done by a Newport patented process that determines the coil positions relative to the magnetic track solely based on the encoder feedback, and avoids major motion during initialization.

Initialization Cycle Duration

The stage initialization cycle duration, *InitializationCycleDuration* (s), is the time period of the motor initialization process.

This parameter is used only with the LMI (Large Move Initialization) initialization process.

The recommended starting value of this parameter varies from 1 to 20 seconds and is inversely related to the stage friction. For example, a stage with minimal friction requires a longer stage initialization cycle to stabilize the stage during initialization.

For all other configuration file parameters refer to section 23.9.5: "*AnalogSin60Acceleration*".

23.9.15 AnalogDualSin120Acceleration

Motor Driver Interface Type: AnalogDualSin120Acceleration

This motor driver interface is used when the output of the position servo loop refers to a modulated acceleration value balanced on two controller axes and when both driver inputs are analog modulated signals. This is a specific interface for driving two motors via two drivers in parallel. The modulation is based on the position and suitable for synchronous linear or rotary brushless motors. Use this Motor interface type for motors whose phase difference between the two output channels are 120° between phases for three phase motors.

NOTE

This interface type does not require Hall sensors for the motor phase initialization. This is done by a Newport patented process that determines the coil positions relative to the magnetic track solely based on the encoder feedback, and avoids major motion during initialization.

For all other configuration file parameters refer to section 23.9.7: "*AnalogDualSin60Acceleration*".

23.9.16 AnalogDualSin120AccelerationLMI

Motor Driver Interface Type: AnalogDualSin120AccelerationLMI

This motor driver interface is used when the output of the position servo loop refers to a modulated acceleration value balanced on two controller axes and when both driver inputs are analog modulated signals. This is a specific interface for driving two motors via two drivers in parallel. The modulation is based on the position and suitable for synchronous linear or rotary brushless motors. Use this Motor interface type for motors whose phase difference between the two output channels are 120° between phases for three phase motors. Large Move Initialization (LMI) will be enabled under this configuration allowing for a larger move during the initialization process as compared to standard initialization process.

NOTE

This interface type does not require Hall sensors for the motor phase initialization. This is done by a Newport patented process that determines the coil positions relative to the magnetic track solely based on the encoder feedback, and avoids major motion during initialization.

Initialization Cycle Duration:

The stage initialization cycle duration, *InitializationCycleDuration* (s), is the time period of the motor initialization process.

This parameter is used only with the LMI (Large Move Initialization) initialization process.

The recommended starting value of this parameter varies from 1 to 20 seconds and is inversely related to the stage friction. For example, a stage with minimal friction requires a longer stage initialization cycle to stabilize the stage during initialization.

For all other configuration file parameters refer to section 23.9.7: "*AnalogDualSin60Acceleration*".

23.9.17 AnalogStepperPosition

Motor Driver Interface Type: AnalogStepperPosition

This driver command interface is used when the output of the position servo loop is a position value and when the driver inputs are two channels of analog sine/cosine signals. For example, this would be the case for a stepper motor connected to a driver board XPS-DRV01 and a position loop setting to either *PI with position output* or to *No servo loop with position output*.

Delay After Motor On To Set Closed Loop (Not used by standard XPS-Q firmware)

- In seconds.

This delay enables to wait after a motor ON to give time with certain external motor amplifiers to stabilize before the controller closes the position control loop.

Default value: 0.050 s

Scaling Current

The motor current at maximum command, *ScalingCurrent* (A), scales the output of the controller. Its value corresponds to the current output of the driver with a +10 V input signal. It must be greater than zero.

When used with driver board XPS-DRV01, this value must be set to 3 A.

Displacement Per Full Step

The stage displacement per motor full step, *DisplacementPerFullStep* (units), defines the stage displacement generated by one full step of the motor. Note: One full step displacement corresponds to ¼ of the electrical period.

The *DisplacementPerFullStep* defines the measurement units of the stage and many parameters are derived from this value, essentially all parameters with units of length, such as velocities or accelerations. Therefore, it is critical this parameter be set correctly for proper operation.

To calculate the *DisplacementPerFullStep* value all of the following must be taken into account: the number of steps per revolution, screw pitch and any gear reduction in the stage.

NOTE

The value for the *maximum velocity* (see section 23.3.1: "Type: Sgamma") must be less than or equal to the *DisplacementPerFullStep* multiplied by the servo loop frequency (8 kHz). This is a requirement for smooth operation; however, we recommend not exceeding one quarter of this maximum possible value.

Peak Current Per Phase:

The stepper motor peak current per phase, *PeakCurrentPerPhase* (A), sets the amplitude of the sine/cosine modulated output current of the driver. This corresponds either to the nominal current per phase (1 phase on) or to the nominal current per phases (2 phases on) multiplied by $\sqrt{2}$. This value can be less than the capability of the motor to reduce motor heating. It must be greater than zero and less than or equal to the *motor current at maximum command*.

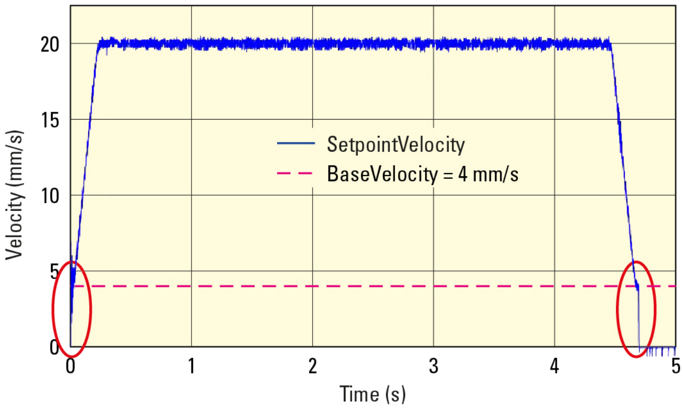
Standby Peak Current Per Phase

The stepper motor standby peak current per phase, *StandByPeakCurrentPerPhase* (A), sets the amplitude of the sine/cosine modulated output current of the driver when the stage has stopped for 5 s. This parameter allows further reduction of motor heating after a motion. It must be greater than zero and less than or equal to the *stepper motor peak current per phase*.

The default value for this parameter is one half of the *stepper motor peak current per phase*.

Base Velocity

The stepper motor start/stop velocity, *BaseVelocity* (units/s), sets the start/stop velocity of the stepper motor. It must be greater than or equal to zero and less than or equal to the maximum velocity, see section 23.4.5: "Encoder Board (StandardLimitAndLimitEncoderPlug)".



In this example, the *BaseVelocity* is set to 4 mm/s: during start/stop periods, the velocity passes through the 4 mm/s velocity step:

- Start period: Starts motion with 4 mm/s velocity and increases from 4 mm/s to the max velocity according to the settings of the motion profiler
- Stop period: Decreases from the max velocity to 4 mm/s velocity according to the settings of the motion profiler and then decreases to null velocity immediately.

The default value for the *BaseVelocity* is zero.

23.9.18 AnalogVelocity

Motor Driver Interface Type: AnalogVelocity

This driver command interface is used when the output of the position servo loop refers to a velocity value and when the driver input is an analog velocity value. For instance, this is the case with a DC motor with tachometer connected to a driver with internal speed loop and a Position servo loop type setting to *PID with a velocity output*.

This driver command interface also provides a configurable current limitation output.

Delay After Motor On To Set Closed Loop (Not used by standard XPS-Q firmware)

- In seconds.

This delay enables to wait after a motor ON to give time with certain external motor amplifiers to stabilize before the controller closes the position control loop.

Default value: 0.050 s.

Scaling Velocity

The stage velocity at maximum command, *ScalingVelocity* (units/s), scales the output of the controller. The value corresponds to the velocity of the positioner with a +10 V input signal to the driver. For the XPS-DRV03 driver board, it is recommended to set this value equal to the maximum allowed stage velocity. For XPS-DRV01 driver board, see section 23.10.6: “DRV01AnalogVelocity (with tachometer feedback)” with the driver board settings.

The value for the *ScalingVelocity* must be greater than zero.

Velocity Limit

This parameter should not be confused with the *profile generator maximum velocity*, which is the maximum velocity a stage can be commanded to move, see section 23.4.5: “Encoder Board (StandardLimitAndLimitEncoderPlug)”. In order to decrease following errors, a positioner must be capable of moving faster than the *profile generator maximum velocity*. The maximum value is defined by the maximum allowed stage velocity, *VelocityLimit* (units/s).

The higher the dynamic bandwidth of a system, the greater the margin between the *maximum allowed stage velocity* and the *profile generator maximum velocity*.

The value for the *maximum allowed stage velocity* must be less than the *stage velocity at maximum command* and greater than or equal to the *profile generator maximum velocity.* The recommended value is 1.2 times the value for the *profile generator maximum velocity*.

Scaling Current

The motor current at maximum command, *ScalingCurrent* (A), scales the output of the controller for the current limitation setting.

- When used with driver board XPS-DRV01, this value must be set to 3 A.
- When used with driver board XPS-DRV03, this value must be set to 5 A.
- When used with driver board XPS-DRV00P, this value scales the 10 V analog output of the output channel B, pin 13 on the XPS-DRV00P (see also next entry ***Current Limit***). This value must be greater than zero.

Current Limit

The maximum allowed motor current, *CurrentLimit* (A), defines the current limitation of the motor driver. This values must be less than or equal to the *motor current at maximum command* and greater than zero.

When used with the driver board XPS-DRV00, this value defines the voltage that gets output on the analog output channel B, pin 13, in relation to the *motor current at maximum command* as follows: Output voltage = 10 V * CurrentLimit (A)/ScalingCurrent (A).

The *CurrentLimit* can be determined as follows:

Motor data

- motor torque constant: Kt (N.m/A)
- maximum allowed motor current: *MotorCurrentLimit* (A)

Driver data:

- motor current at maximum command: *ScalingCurrent* (A)
  (for instance 3A for XPS-DRV01 or 5A for XPS-DRV03)

Stage data:

- ratio between motor rotation and stage displacement: G (revolution/units)
- total inertia on the motor axis: J (kg.m²)

Notice: $J = J_{motor} + J_{load} + J_{mec}$

with: J_{motor}: motor rotor inertia (kg.m²)

J_{load}: load inertia (kg.m²)

J_{mec}: bearing, lead screw, … inertia (kg.m²)

User Performance

- maximum stage acceleration (see section 23.3.1: "Type: Sgamma"):
  $MaximumAcceleration$ (units/s²)

Maximum Allowed Motor Current

- $MaximumCurrent = \frac{MaximumAcceleration \times J \times 2 \times \pi \times G}{Kt}$
- $CurrentLimit = \min(MaximumCurrent \times 1.5, MotorCurrentLimit, ScalingCurrent)$
  (A)

NOTE

It is recommended that the CurrentLimit be 1.5 times the motion profiler maximum current to meet the motion requirements of the default stage dynamics.

See also section 23.10.11: "DRV03AnalogVelocity" for XPS-DRV03 driver board with a RMS limitation.

23.9.19 AnalogVoltage

Motor Driver Interface Type: AnalogVoltage

This driver command interface is used when the output of the position servo loop refers to a motor voltage value and when the driver input is a motor voltage value. For instance, this is the case for a DC motor without tachometer connected to a voltage amplifying driver and a Position servo loop type setting to *PID with motor voltage output*.

This driver command interface also provides a configurable current limitation output.

Delay After Motor On To Set Closed Loop (Not used by standard XPS-Q firmware)

- In seconds.

This delay enables to wait after a motor ON to give time with certain external motor amplifiers to stabilize before the controller closes the position control loop.

Default value : 0.05 s.

Scaling Voltage

The motor voltage at maximum command, *ScalingVoltage* (V), scales the analog output of the controller. This value corresponds to the voltage output of the driver with a +10 V input signal. It must be greater than zero.

When used with the driver board XPS-DRV01 or XPS-DRV03, this value must be set to 48 V.

Voltage Limit

The maximum allowed motor voltage, *VoltageLimit* (V), sets the maximum allowed output voltage of the driver. This value must be less than or equal to the *motor voltage at maximum command* and greater than zero.

This parameter can be determined as follows:

Motor data

- motor winding resistance per phase: R_{mot} (Ω)
- motor torque constant: Kt (N.m/A)
- motor voltage constant: Kv (V/rpm)
- maximum allowed motor current: *MotorCurrentLimit* (A)
- maximum allowed motor voltage: *MotorVoltageLimit* (V)

Driver data

- motor current at maximum command: *ScalingCurrent* (A)
  (for instance 3A for XPS-DRV01 or 5A for XPS-DRV03)
- motor voltage at maximum command; *ScalingVoltage* (V)
  (for instance 48 V for XPS-DRV01 or XPS-DRV03)

Stage data

- ratio between motor rotation and stage displacement: G (revolution/units)
- total inertia on the motor axis: J (kg.m²)

Notice: $J = J_{motor} + J_{load} + J_{mec}$

with: J_{motor}: motor rotor inertia (kg.m²)

J_{load}: load inertia (kg.m²)

J_{mec}: bearing, lead screw, … inertia (kg.m²)

User performance:

- maximum stage velocity (see section 23.3.1: "Type: Sgamma"): $MaximumVelocity$ (units/s)
- maximum stage acceleration (see section 23.3.1: "Type: Sgamma"): $MaximumAcceleration$ (units/s²)

Maximum Allowed Motor Voltage

- $MaximumCurrent = \min\left(\frac{MaximumAcceleration \times J \times 2 \times \pi \times G}{Kt}, MotorCurrentLimit, ScalingCurrent\right)$
- $MaximumVoltage = R_{mot} \times MaximumCurrent + MaximumVelocity \times 60 \times G \times Kv$
- $VoltageLimit = \min(MaximumVoltage \times 1.5, MotorVoltageLimit, ScalingVoltage)$

NOTE

It is recommended that the VoltagaeLimit be 1.5 times the motion profiler maximum voltage to meet the motion requirements of the default stage dynamics.

Scaling Current

The motor current at maximum command, *ScalingCurrent* (A), scales the output of the controller for the current limitation setting.

- When used with the driver board XPS-DRV01, this value must be set to 3 A.
- When used with the driver board XPS-DRV03, this value must be set to 5 A.
- When used with the driver board XPS-DRV00, this value scales the 10 V analog output of the output channel B, pin 13 on the XPS-DRV00 (see also section 23.9.18: “AnalogVelocity”). Its value must be greater than zero.

Current Limit

The maximum allowed motor current, *CurrentLimit* (A), defines the current limit of the motor driver. This value must be less than or equal to the *motor current at maximum command* and greater than zero.

When used with the driver board XPS-DRV00, this value defines the voltage that gets output on the analog output channel B, pin 13, in relation to the *motor current at maximum command* as follows: Output voltage = 10 V * CurrentLimit (A)/ScalingCurrent (A).

The *CurrentLimit* can be determined as follows:

Motor data

- motor torque constant: Kt (N.m/A)
- maximum allowed motor current: *MotorCurrentLimit* (A)

Driver data

- motor current at maximum command: *ScalingCurrent* (A)
  (for instance 3A for XPS-DRV01 or 5A for XPS-DRV03)

Stage data

- ratio between motor rotation and stage displacement: G (revolution/units)
- total inertia on the motor axis: J (kg.m²)

Notice: $J = J_{motor} + J_{load} + J_{mec}$

with: J_{motor}: motor rotor inertia (kg.m²)

J_{load}: load inertia (kg.m²)

J_{mec}: bearing, lead screw, … inertia (kg.m²)

User performance

- maximum stage acceleration (see section 23.3.1: “Type: Sgamma”): $MaximumAcceleration$ (units/s²)

Maximum allowed motor current:

- $MaximumCurrent = \frac{MaximumAcceleration \times J \times 2 \times \pi \times G}{Kt}$
- $CurrentLimit = \min(MaximumCurrent \times 1.5, MotorCurrentLimit, ScalingCurrent)$ (A)

NOTE

It is recommended that the CurrentLimit be 1.5 times the motion profiler maximum current to meet the motion requirements of the default stage dynamics.

See also section 23.10.12: “DRV03AnalogVoltage” for XPS-DRV03 driver board with a RMS limitation.

23.9.20 DigitialStepperPosition

Motor Driver Interface Type: DigitalStepperPosition

Required entries in the configuration file: MotorInterfaceType = *PulseDir*

This motor interface drives external stepper motors. The output signals are named PLS_OUT and DIR_OUT. To modify the logic of these signals, two modes are available:

- PLS_OUT = pulses generation only.
- DIR_OUT = direction information only.

DigitalStepperDirectionLogic	**Negative**				**Positive**			
DigitalStepperPulseLogic	**Positive**		**Negative**		**Positive**		**Negative**	

DIR_OUT	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
direction	+	-	+	-	-	+	-	+

PLS_OUT	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
pulse	□	■	■	□	□	■	■	□

■ = pulse □ = no pulse + = positive direction - = negative direction

Delay After Motor On To Set Closed Loop (Not used by standard XPS-Q firmware)

- In seconds.

This delay enables to wait after a motor ON to give time with certain external motor amplifiers to stabilize before the controller closes the position control loop.

Default value: 0.050 s.

Digital Stepper Pulse Logic

Stages.ini file entry: *DigitalStepperPulseLogic*

The logic pulse can be **Negative** (PLS_OUT: 1 = no pulse and 0 = pulse) or **Positive** (PLS_OUT: 1 = pulse and 0 = no pulse).

PLS_OUT	**0**	**1**
DigitalStepperPulseLogic = **Positive**	No pulse	Pulse
DigitalStepperPulseLogic = **Negative**	Pulse	No pulse

Digital Stepper Direction Logic

Stages.ini file entry: *DigitalStepperDirectionLogic*

The logic direction can be **Negative** (DIR_OUT: 1 = negative direction and 0 = positive direction) or **Positive** (DIR_OUT: 1 = positive direction and 0 = negative direction).

The PLS_OUT represents the pulse generation.

DIR_OUT	**0**	**1**
DigitalStepperDirectionLogic **= Positive**	Direction -	Direction +
DigitalStepperDirectionLogic **= Negative**	Direction +	Direction -

Displacement Per Full Step

Stages.ini file entry: *DisplacementPerFullStep*

The stage displacement per motor full step, *DisplacementPerFullStep* (units), defines the stage displacement generated by one full step of the motor.

NOTE

One full step displacement corresponds to ¼ of the electrical period.

The *DisplacementPerFullStep* defines the measurement units of the stage and many parameters are derived from this value, essentially all parameters with units of length, such as velocities or accelerations. It is critical this value be set correctly for proper operation of the stage.

To calculate the *DisplacementPerFullStep* value all of the following must be taken into account: the number of steps per revolution, screw pitch and any gear reduction in the stage.

Micro Steps Per Full Step

Stages.ini file entry: *MicroStepsPerFullStep*

The *MicroStepsPerFullStep* defines the number of micro steps in the displacement per one full step of the stepper motor.

23.10 Driver

In this configuration category, users are building the Motor driver parameters section of the stages.ini file.

Example:

```
; --- Motor driver parameters
; --- <Driver.DRV02>
DriverName = XPS-DRV02
DriverMotorResistance = 5.5 ; Ohm
DriverMotorInductance = 0.0018 ; Henry
DriverMaximumPeakCurrent = 2.51 ; Amp
DriverMaximumRMSCurrent = 1.14 ; Amp
DriverRMSIntegrationTime = 15 ; s
DriverThermistanceThreshold = 1000 ; Ohm
DriverCutOffFrequency = 400 ; Hz
```

The XPS controller supports the following settings for the motor driver model:

- No Driver
- XPS-DRV01 with tachometer feedback
- XPS-DRV01 without tachometer feedback
- XPS-DRV01 for stepper motors
- XPS-DRV02 for linear/brushless motors
- XPS-DRV03 with tachometer feedback
- XPS-DRV03 for acceleration control
- XPS-DRV03 for voltage control
- XPS-DRV00 for non-configurable external drivers
- XPS-DRV00P for configurable external drivers
- NON_CONFIGURABLE_DRV for non-configurable external drivers (CIE and external driver are directly connected, without use of DRV00/DRV00P pass-through cards).
- XPS-DRVPx (x = 1, 2, …) for piezo stage drivers

The choice of a driver board setting depends on the driver board used, the driver command interface, position servo loop type, and motor type.

23.10.1 No Driver (NoDriver)

Motor Driver Type: NoDriver

This type of Driver Name is used when there is not a driver card in the XPS controller and the controller is not connected to a stage.

Driver Name: NO_DRIVER

There are no additional parameters to set for this motor interface type.

23.10.2 NonConfigurableDriver

Motor Driver Type: NonConfigurableDriver

This type of Driver Name is used when connecting the XPS controller to a non-configurable external motor driver without use of a XPS-DRV00 or XPS-DRV00P pass-through card. This setting of the motor driver type is compatible with all driver command interfaces.

Driver Name: NON_CONFIGURABLE_DRV

There are no additional parameters to set for this motor interface type.

23.10.3 DRV00 (for Non-Configurable External Driver)

Motor Driver Type: DVR00

The XPS-DRV00 is a pass-through board needed for connecting the XPS controller to an external motor driver. This setting of the motor driver type is compatible with all driver command interfaces.

Driver Name: XPS-DRV00

There are no additional parameters to set for this motor interface type.

23.10.4 DRV00P

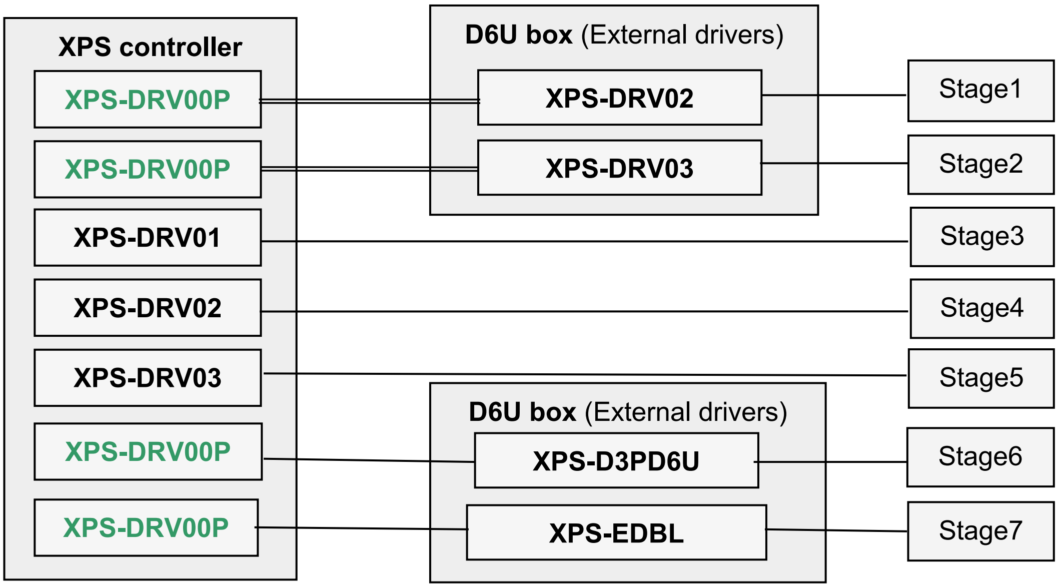
Motor Driver Type: DRV00P

The XPS-DRV00P (DRV00 Version 2) is a pass through board developed for the CIE05/CIE08 board. It is an interconnect board for an external amplifier to connect to the XPS controller. The XPS-DRV00P is similar to an XPS-DRV00 card with the added capability of Pulse/Direction outputs for stepper motor control (Pulse/Dir or Pulse+/Pulse- mode). The I2C communication link allows the user to configure the configurable external drivers (for example external XPS-DRV02, XPS-DRV03, XPS-D3PD6U or XPS-EDBL Newport drivers).

Driver Name: XPS-DRV00P

How to use a configurable external driver?

A configurable external driver must be connected to the XPS controller with an XPS-DRV00P board.



NOTE

In the stages.ini file, the "DriverName" of your stage configuration must be declared as XPS-DRV02, XPS-DRV03, XPS-D3PD6U or XPS-EDBL, but not XPS-DRV00P.

[Stage1]
DriverName = XPS-DRV02

[Stage2]
DriverName = XPS-DRV03

[Stage6]
DriverName = XPS-D3PD6U

[Stage7]
DriverName = XPS-EDBL

There are no additional parameters to set for this motor interface type.

23.10.5 DRV01AnalogStepperPosition (for stepper motors)

Motor Driver Type: DRV01AnalogStepperPosition

This type of motor driver model is used for stepper motors. The motor driver interface must be set to *sine/cosine position control* (see section 23.9.17: "AnalogStepperPosition") and the position servo loop type to either *PI with a position output* (see section 23.8.6: "PIPosition") or to *no servo loop with a position output* (see section 23.8.2: "NoEncoderPosition"). The XPS-DRV01 driver board supplies a maximum output of 3 A and 48 V.

Driver Name: XPS-DRV01

Driver PWM Frequency: *{int, 50, 52, 54, 57, 65, 69, 73, 78, 96, 104, 113, 125, 178, 200, 250, 300} Hz*

The pulse width modulation frequency, *DriverPWMFrequency* (Hz), sets the frequency of the pulse width modulation output of the XPS-DRV01 driver.

The default value is 50 kHz.

Driver Stepper Winding: {string, Full, Half}

The stepper motor winding connection, *DriverStepperWinding*, specifies the stepper wiring. If the stepper is wired between pin 1-4 and pin 5-8, the parameter must be set to *Full.* If the stepper is wired between pin 1-4 and 11-12 as well as pin 5-10, the parameter must be set to *Half.*

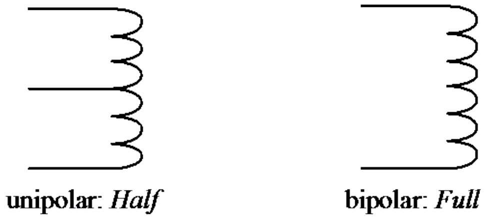


Figure 113: Stepper motor winding connection.

23.10.6 DRV01AnalogVelocity (with tachometer feedback)

Motor Driver Type: DRV01AnalogVelocity

This type of motor driver model is used for DC motors with tachometer. The motor driver interface must be set to *velocity control* (see section 23.9.18: "AnalogVelocity") and the position servo loop type to *PID with velocity output* (see section 23.8.5: "PIDFFVelocity"). The XPS-DRV01 driver board supplies a maximum output of 3 A and 48 V.

Driver Name: XPS-DRV01

Driver PWM Frequency: *{int, 50, 52, 54, 57, 65, 69, 73, 78, 96, 104, 113, 125, 178, 200, 250, 300} Hz*

The pulse width modulation frequency, *DriverPWMFrequency* (Hz), sets the frequency of the pulse width modulation output of the XPS-DRV01 driver.

The default value is 50 kHz.

- **Driver Error Amplifier Gain**: {int, 1, 5, 9, 13, 17, 21, 25, 29}
- **Driver Tachometer Gain**: {int, 0, 25, 27, 30, 33, 50, 60, 75, 100}

The velocity servo loop proportional gain and the tachometer gain should be set together to optimize the bandwidth of the velocity loop.

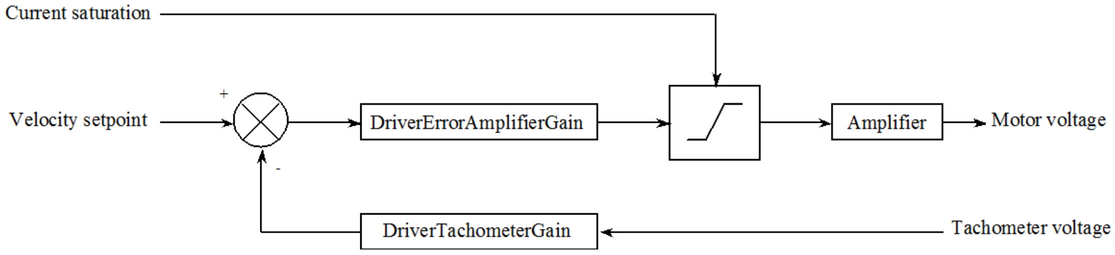


Figure 114: Velocity servo loop.

Motor data

- motor maximum allowed velocity: $V_{motor\ max}$ (rpm)
- motor winding resistance per phase: R_{mot} (Ω)
- motor torque constant: Kt (N.m/A)
- motor voltage constant: Kv (V/rpm)

Notice: $Kt = Kv$ when there are given in these units

Tachometer data

- tachometer voltage constant: K_{GT} (V/rpm)

Driver data

- DC power supply voltage: $U_{DC} = 48V$
- DAC maximum voltage: $U_{DAC\,\max} = 10V$
- maximum allowed current at maximum command: $ScalingCurrent = 3A$
- tachometer feedback maximum voltage: $U_{\text{tachometer max}} = 10V$
- tachometer gain (*DriverTachometerGain*): $K_{\text{tachometer}} \in \{100, 75, 60, 50, 33, 30, 27, 25, 0\}$
- velocity servo loop proportional gain (*DriverErrorAmplifierGain*): $Kp_{velocity} \in \{1, 5, 9, 13, 17, 21, 25, 29\}$

Stage data

- ratio between motor rotation and stage displacement: G (revolution/units)
- total inertia on the motor axis: J (kg.m²)

Notice: $J = J_{motor} + J_{load} + J_{mec}$

with: J_{motor}: motor rotor inertia (kg.m²)

J_{load}: load inertia (kg.m²)

J_{mec}: bearing, lead screw, … inertia (kg.m²)

- stage mechanical time constant: τ_m (s)

Notice: $\tau_m = \dfrac{R_{mot} \times J}{Kt^2}$

User performance

- maximum allowed stage velocity (see section 23.9.18: “AnalogVelocity”): $VelocityLimit$ (units/s)
- maximum stage acceleration (see section 23.3.1: “Type: Sgamma”): $MaximumAcceleration$ (units/s²)
- velocity servo loop cut off frequency: Fc (Hz)

Notice: Usually this cut off frequency is between 100 Hz and 200 Hz

Preliminary tachometer gain

- maximum motor velocity: $V_{motor\,\text{limit}} = 60 \times G \times V_{\text{limit}} \leq V_{motor\,\max}$ (rpm)
- raw tachometer gain: $K_{\text{tachometer}\,raw} = \dfrac{U_{\text{tachometer max}}}{V_{motor\,\text{limit}} \times K_{GT}} \times 100$
- tachometer gain: $K_{\text{tachometer}}$ closest lower value

Maximum allowed motor current

- $CurrentLimit = \dfrac{MaximumAcceleration \times 2 \times \pi \times J \times G}{Kt} \times 1.5 \leq ScalingCurrent$ (A)

Notice: The coefficient 1.5 is the margin for parameters uncertainty and servo loop transient.

Velocity servo loop proportional gain

- raw velocity servo loop proportional gain:

$$Kp_{velocit\,raw} = \frac{U_{DAC\max} \times Kv \times \left(\tau_m \times 2 \times \pi \times Fc - 1\right)}{U_{DC} \times K_{GT} \times \frac{K_{tachometer}}{100}}$$

- velocity servo loop proportional gain: $Kp_{velocity}$ closest higher value

Cut off frequency recalculation due to quantification

- raw cut off frequency: $Fc_{raw} = \dfrac{\dfrac{Kp_{velocity} \times U_{DC} \times K_{GT}}{U_{DAC\max} \times Kv} \times \dfrac{K_{tachometer}}{100} + 1}{\tau_m \times 2 \times \pi}$ (Hz)
- if the value is too far from the velocity servo loop cut off frequency, the tachometer gain $K_{tachometer}$ can be decreased.

Stage velocity at maximum command (see section 23.9.18: "AnalogVelocity"):

- $ScalingVelocity = \dfrac{\dfrac{Kp_{velocity} \times U_{DC}}{60 \times G \times Kv}}{\dfrac{Kp_{velocity} \times U_{DC} \times K_{GT}}{U_{DAC\max} \times Kv} \times \dfrac{K_{tachometer}}{100} + 1} \geq VelocityLimit$ (units/s)

23.10.7 DRV01AnalogVoltage (without tachometer feedback)

Motor Driver Type: DRV01AnalogVoltage

This type of motor driver model is used with DC motors without tachometer. The motor driver interface must be set to *voltage control* (see section 23.9.19: "AnalogVoltage") and the position corrector type to *PID with voltage output* (see section 23.8.3: "PIDDualFFVoltage"). The XPS-DRV01 driver board supplies a maximum output of 3 A and 48 V.

Driver Name: XPS-DRV01

Driver PWM Frequency: *{int, 50, 52, 54, 57, 65, 69, 73, 78, 96, 104, 113, 125, 178, 200, 250, 300}*

The pulse width modulation frequency, *DriverPWMFrequency* (Hz), sets the frequency of the pulse width modulation output of the XPS-DRV01 driver.

The default value is 50 kHz.

23.10.8 DRV02

Motor Driver Type: DRV02

This type of motor driver model is used for brushless linear or rotary motors. The motor driver interface must be set to *120° UV phase acceleration control* (see section 23.9.13: "AnalogSin120Acceleration") or *120° UV phase dual output acceleration control* (see section 23.9.15: "AnalogDualSin120Acceleration") and the position corrector type to *PID with acceleration output* (see section 23.8.4: "PIDFFAcceleration").

The XPS-DRV02 driver board supplies a maximum output of 5 A and 44 Vpp.

Driver Name: XPS-DRV02

Driver Motor Resistance:

The motor winding resistance per phase, *DriverMotorResistance* (Ω), is the motor resistance of each phase. It must be greater than zero and less than or equal to 65.535 Ω.

Driver Motor Inductance:

The motor winding induction per phase, *DriverMotorInductance* (H), is the motor induction of each phase. It must be greater than zero and less than or equal to 65.535 mH.

Driver Cut Off Frequency:

The current servo loop cut off frequency, *DriverCutOffFrequency* (Hz), sets the bandwidth of the internal driver current servo loop. The internal driver PI parameters are calculated automatically. This value has to be set relative to the bandwidth of the position servo loop (see section 23.8.4: "PIDFFAcceleration"). It must be greater than zero and less than or equal to 3 kHz.

For a stage with a position servo loop cut off frequency between 20 and 50 Hz, the *current servo loop cut off frequency* should be set between 200 and 500 Hz.

Current Monitoring Parameters entries in the configuration file:

- ***Driver Maximum Peak Current*** — peak current limit
- ***Driver Maximum RMS Current*** — RMS current limit
- ***Driver RMS Integration Time*** — RMS integration time

The peak current limit, *DriverMaximumPeakCurrent* (A), is the maximum allowed motor current. If the motor current goes beyond this value, a driver fault is generated. This value must be greater than zero and less than or equal to 5 A.

The RMS current limit, *DriverMaximumRMSCurrent* (A), is the maximum allowed RMS motor current. The RMS integration time *DriverRMSIntegrationTime* (s) defines the integration time span. If the RMS motor current goes beyond this value, a driver fault is generated. The *RMS Current limit* must be greater than zero and less than or equal to 5 A. The *RMS integration time* must be greater than zero and less than or equal to 480 s.

There are many different methods possible to define these values. Here is a description of one method used by Newport for standard products.

Application input:

- Due to the high accuracy requirements of Newport testing, a maximum temperature increase of the motor coils by 20°C is set to avoid thermal effects impacting the motion precision: $\Delta T = 20\ °\mathrm{C}$
- A ratio of 2 between the peak current limit and the RMS current limit is used. This has been found to be a realistic approach in numerous precision motion applications as a good balance between throughput and precision: r = 2.

Motor data

- motor force/torque constant: Kt (N/Arms) or (N.m/Arms)
- motor constant or steepness: S (N²/W) or (N².m²/W)
- thermal resistance: R_{th} (°C/W)
- thermal time constant: τ_{th} (s)

Driver data

- maximum driver current: $\mathbf{I_{max}} = 5\ \mathbf{A}$

Stage data

- load (linear direct drive stage only) : *Load* (kg)
- ratio between motor rotation and stage displacement: G (revolution/units)
- total inertia on the motor axis: J (kg.m²)

Notice: $J = J_{motor} + J_{load} + J_{mec}$

with: J_{motor}: motor rotor inertia (kg.m^2)

J_{load}: load inertia (kg.m^2)

J_{mec}: bearing, lead screw, … inertia (kg.m^2)

Calculations

- $Kt = \sqrt{3 \times Rf \times S}$, where Rf (Ω) is the motor resistance per phase
- $\tau_{th} = \dfrac{R_{th} \times F_p^2}{\theta\tau \times S}$ or $\tau_{th} = \dfrac{R_{th} \times C_p^2}{\theta\tau \times S}$,

  where: F_p (N) is the motor peak force, C_p (N.m) is the motor peak torque and $\theta\tau$(°C/s) is the temperature rise at motor peak force
- *DriverMaximumRMSCurrent*: $I_{RMS} = \min\left(\sqrt{\dfrac{\Delta T \times S}{R_{th} \times (1 + 0.004 \times \Delta t)}} \times \dfrac{\sqrt{2}}{Kt}, \mathrm{I}_{\max}\right)$
- *DriverMaximumPeakCurrent*: $I_{peak} = \min(I_{RMS} \times r \times 1.1, \mathrm{I}_{\max})$

Notice: The coefficient 1.1 is the margin for the servo loop transient.

- DriverRMSIntegrationTime: $\min(\tau_{th}, 15s)$
- ScalingAcceleration: $A_{scaling} = \dfrac{I_{\max}}{\sqrt{2}} \times \dfrac{Kt}{Load}$ (m/s^2) or

  $A_{scaling} = \dfrac{I_{\max}}{\sqrt{2}} \times \dfrac{Kt}{2 \times \pi \times J \times G}$ (units/s^2)
- AccelerationLimit: $A_{limit} = \min\left(\dfrac{I_{RMS}}{\sqrt{2}} \times r \times \dfrac{Kt}{Load}, \mathrm{A}_{scaling}\right)$ (m/s^2) or

  $A_{limit} = \min\left(\dfrac{I_{RMS}}{\sqrt{2}} \times r \times \dfrac{Kt}{2 \times \pi \times J \times G}, \mathrm{A}_{scaling}\right)$ (units/s^2)

Driver Thermistance Threshold

The thermistor threshold, *DriverThermistanceThreshold* (Ω), sets the threshold for the driver fault in the event of overheating. This value must be greater than 100 Ω and less than or equal to 9 kΩ.

For a 1 kΩ PTC temperature sensor, the thermistor threshold value is 1000, corresponding to the resistor value of the transition edge. The temperature that corresponds to that threshold is determined by the type of sensor (see color code of the wire).

23.10.9 DRV02P

Motor Driver Type: DVR02P

This type of motor driver model is used for brushless linear or rotary motors. The motor driver interface must be set to *120 deg UV phase acceleration control* (see section 23.9.13: "AnalogSin120Acceleration") or *120 deg UV phase dual output acceleration control* (see section 23.9.15: "AnalogDualSin120Acceleration") and the position corrector type to *PID with acceleration output* (see section 23.8.4: "PIDFFAcceleration").

The XPS-DRV02P driver board supplies a maximum output of 7 A and 44 Vpp.

Driver Name: XPS-DRV02P

For all other configuration file parameters refer to section 23.10.8: "*DRV02*".

23.10.10 XPS-DRV03 for acceleration control (DRV03AnalogAcceleration)

Type: DRV03AnalogAcceleration

This type of motor driver model is used for DC motors controlled by acceleration. The moto driver interface must be set to *acceleration control* (see section 23.9.2: “AnalogAcceleration”) and the position corrector type to *PID with acceleration output* (see section 23.8.4: “PIDFFAcceleration”). The XPS-DRV03 driver board supplies a maximum output of 5 A and 48 V. The XPS-DRV03H driver board supplies a maximum output of 1.58 A and 48 V.

Driver Name: XPS-DRV03

Driver Motor Resistance

The motor winding resistance, *DriverMotorResistance* (Ω), is the motor resistance. This value must be greater than zero and less than or equal to 65.535 Ω.

Driver Motor Inductance

The motor winding inductance, *DriverMotorInductance* (H), is the motor inductance. This value must be greater than zero and less than or equal to 65.535 mH.

Driver Current Cut Off Frequency

The current servo loop cut off frequency, *DriverCurrentCutOffFrequency* (Hz), sets the bandwidth of the internal driver current servo loop. The internal driver corrector parameters are calculated automatically. This value has to be set in relation to the bandwidth of the position servo loop (see section 23.8.4: “PIDFFAcceleration”). This value must be greater than zero and less than or equal to 3 kHz.

For a stage with a position servo loop cut off frequency between 20 and 50 Hz, the *current servo loop cut off frequency* should be set between 200 and 500 Hz.

Current Monitoring Parameters entries for the configuration file:

- ***Driver Maximum Peak Current*** — peak current limit
- ***Driver Maximum RMS Current*** — RMS current limit
- ***Driver RMS Integration Time*** — RMS integration time

The peak current limit, *DriverMaximumPeakCurrent* (A), is the maximum allowed motor current. If the motor current exceeds this value, a driver fault is generated. This value must be greater than zero and less than or equal to 5 A.

The RMS current limit, *DriverMaximumRMSCurrent* (A), is the maximum allowed RMS motor current. The RMS integration time *DriverRMSIntegrationTime* (s) defines the integration time span. If the RMS motor current exceeds this value, a driver fault is generated. The *RMS Current limit* must be greater than zero and less than or equal to 5 A. The *RMS integration time* must be greater than zero and less than or equal to 60 s.

There are many different methods to define these values. Here is a description of a method used by Newport for standard products.

Application input:

- Because high accuracy needs, a maximum temperature raise of the motor coils by 20°C is set to avoid thermal effects impacting the motion precision:
- A ratio of 2 between the peak current limit and the RMS current limit is used. This has been found to be a good approach in numerous precision motion applications offering a good balance between throughput and precision: = 2

Motor data

- motor torque constant: Kt (N.m/Arms)
- motor constant or steepness: S (N²m²/W)
- thermal resistance: R_{th} (°C/W)

- thermal time constant: τ_{th} (s)

Driver data

- maximum driver current: $\mathbf{I}_{\max} = 5\ \mathbf{A}$

Stage data

- ratio between motor rotation and stage displacement: G (revolution/units)
- total inertia on the motor axis: J (kg.m²)

Notice: $J = J_{motor} + J_{load} + J_{mec}$

with: J_{motor}: motor rotor inertia (kg.m²)

J_{load}: load inertia (kg.m²)

J_{mec}: bearing, lead screw, … inertia (kg.m²)

Calculations

- $Kt = \sqrt{Rf \times S}$, where Rf (Ω) is the motor resistance per phase
- $\tau_{th} = \dfrac{R_{th} \times C_p^2}{\theta\tau \times S}$,

  where: C_p (N.m) is the motor peak torque and

  $\theta\tau$(°C/s) is the temperature rise at motor peak torque
- *DriverMaximumRMSCurrent*: $I_{RMS} = \min\left(\sqrt{\dfrac{\Delta T \times S}{R_{th} \times (1 + 0.004 \times \Delta t)}} \times \dfrac{1}{Kt}, \mathrm{I}_{\max}\right)$
- *DriverMaximumPeakCurrent*: $I_{peak} = \min(I_{RMS} \times r \times 1.1, \mathrm{I}_{\max})$

Notice: The coefficient 1.1 is the margin for the servo loop transient.

- DriverRMSIntegrationTime: $\min(\tau_{th}, 3s)$
- ScalingAcceleration: $A_{scaling} = I_{\max} \times \dfrac{Kt}{J \times G \times 2\pi}$ (units/s²)
- AccelerationLimit: $A_{limit} = \min\left(I_{RMS} \times r \times \dfrac{Kt}{J \times G \times 2\pi}, \mathrm{A}_{scaling}\right)$ (units/s²)

Driver Maximum Motor Voltage

The maximum allowed motor voltage, *DriverMaximumMotorVoltage* (V), sets the maximum allowed output voltage of the driver.

23.10.11 DRV03AnalogVelocity

Motor Driver Type: DRV03AnalogVelocity

This type of motor driver model is used for DC motors with tachometer. The driver command interface must be set to *velocity control* (see section 23.9.18: “AnalogVelocity”) and the position servo loop type to *PID with velocity output* (see section 23.8.5: “PIDFFVelocity”). The XPS-DRV03 driver board supplies a maximum output of 5 A and 48 V. The XPS-DRV03H driver board supplies a maximum output of 1.58 A and 48 V.

Driver Name: XPS-DRV03

Driver Motor Resistance

The motor winding resistance, *DriverMotorResistance* (Ω), is the motor resistance. This value must be greater than zero and less than or equal to 655.35 Ω.

Driver Motor Inductance

The motor winding induction, *DriverMotorInductance* (H), is the motor inductance. This value must be greater than zero and less than or equal to 65.535 mH.

Driver Motor Voltage Constant

The motor voltage constant parameter, *DriverMotorVoltageConstant* (Volt/rpm), sets the back EMF constant of the motor. This value must be greater than zero and less than or equal to 65.535e^{-3} V/rpm.

Driver Tacho Generator Voltage

The tachometer generator voltage parameter, *DriverTachoGeneratorVoltage* (Volt/rpm), set the voltage constant of the tachometer generator. This value must be greater than zero and less than or equal to 65.535e^{-3} $^{\text{V}}$/rpm.

Driver Stage Inertia

The stage inertia, *DriverStageInertia* (kg.m²), is the total inertia (*J*) on the motor axis. It must be greater or equal to 10^{-9} kg.m² and less than or equal to 1 kg.m².

Notice: $J = J_{motor} + J_{load} + J_{mec}$

with: J_{motor}: motor rotor inertia (kg.m²)

J_{load}: load inertia (kg.m²)

J_{mec}: bearing, lead screw, … inertia (kg.m²)

Driver Gear Ratio

The gear ratio, *DriverGearRatio* (revolution/unit), sets the ratio between the motor rotation and the stage displacement. It must be greater than 0.

Driver Current Cut Off Frequency

The current servo loop cut off frequency, *DriverCurrentCutOffFrequency* (Hz), sets the bandwidth of the internal driver current servo loop. The driver internal corrector parameters are calculated automatically. This value has to be set in relation to the bandwidth of the velocity servo loop (see section 23.8.5: “PIDFFVelocity”). This value must be greater than zero and less than or equal to 3 kHz.

For a stage with a velocity servo loop cut off frequency between 100 and 200 Hz, the current servo loop cut off frequency should be set between 500 and 1000 Hz.

Driver Velocity Cut Off Frequency

The velocity servo loop cut off frequency, *DriverVelocityCutOffFrequency* (Hz), sets the bandwidth of the internal driver velocity servo loop. The driver internal corrector parameters are calculated automatically. This value has to be set in relation to the bandwidth of the velocity servo loop (see section 23.8.5: “PIDFFVelocity”). This value must be greater than zero and less than or equal to 300 Hz.

For a stage with a velocity servo loop cut off frequency between 20 and 50 Hz, the velocity servo loop cut off frequency should be set between 100 and 200 Hz.

Current Monitoring Parameters entries for the configuration file:

- ***Driver Maximum RMS Current*** — RMS current limit
- ***Driver RMS Integration Time*** — RMS integration time

The RMS current limit, *DriverMaximumRMSCurrent* (A), is the maximum allowed RMS motor current. The RMS integration time *DriverRMSIntegrationTime* (s) defines the integration time span. If the RMS motor current goes beyond this value, a driver fault is generated. The RMS Current limit must be greater than zero and less than or equal to 5 A. The RMS integration time must be greater than zero and less than or equal to 60 s.

There are many different methods to define these values. Here is a description of a method used by Newport for standard products.

Application input:

- Due to the high accuracy requirements of Newport testing, a maximum temperature increase of the motor coils by 20°C is set to avoid thermal effects impacting the motion precision: $\Delta T = 20°C$.
- A ratio of 2 between the current limit (defined in the motor interface section) and the RMS current limit is used. This has been found to be a good approach in numerous precision motion applications offering a good balance between throughput and precision: r = 2.

Motor data

- motor torque constant: Kt (N.m/Arms)
- motor constant or steepness: S (N².m²/W)
- thermal resistance: R_{th} (°C/W)
- thermal time constant: τ_{th} (s)

Driver data

- maximum driver current: $\mathrm{I}_{\max} = 5$ **A**

Calculations

- $Kt = \sqrt{Rf \times S}$, where Rf (Ω) is the motor resistance per phase
- $\tau_{th} = \dfrac{R_{th} \times C_p^2}{\theta\tau \times S}$,

  where: C_p (N.m) is the motor peak torque and

  $\theta\tau$(°C/s) is the temperature rise at motor peak torque
- *DriverMaximumRMSCurrent*: $I_{RMS} = \min\left(\sqrt{\dfrac{\Delta T \times S}{R_{th} \times (1 + 0.004 \times \Delta t)}} \times \dfrac{1}{Kt}, \mathrm{I}_{\max}\right)$
- $CurrentLimit = \min(I_{RMS} \times r, \mathrm{I}_{\max})$
- *DriverRMSIntegrationTime:* $\min(\tau_{th}, 3s)$

Driver Maximum Motor Voltage

The maximum allowed motor voltage, *DriverMaximumMotorVoltage* (V), sets the maximum allowed output voltage of the driver. It must be greater than zero and less than or equal to 48 V.

This parameter can be determined as follows:

Motor data

- motor winding resistance per phase: R_{mot} (Ω)
- motor torque constant: Kt (N.m/A)
- motor voltage constant: Kv (V/rpm)
- maximum allowed motor current: *MotorCurrentLimit* (A)
- maximum allowed motor voltage: *MotorVoltageLimit* (V)

Driver data

- motor current at maximum command: ScalingCurrent (A) (5A for XPS-DRV03)
- motor voltage at maximum command; ScalingVoltage (V) (48 V for XPS-DRV03)

Stage data

- ratio between motor rotation and stage displacement: G (revolution/units)
- total inertia on the motor axis: J (kg.m²)

Notice: $J = J_{motor} + J_{load} + J_{mec}$

with: J_{motor}: motor rotor inertia (kg.m²)

J_{load}: load inertia (kg.m²)

J_{mec}: bearing, lead screw, … inertia (kg.m²)

User performance

- maximum stage velocity (see section 23.3.1: “Type: Sgamma”): $MaximumVelocity$ (units/s)
- maximum stage acceleration (see section 23.3.1: “Type: Sgamma”): $MaximumAcceleration$ (units/s²)

Maximum allowed motor voltage

- $MaximumCurrent = \min\left(\frac{MaximumAcceleration \times J \times 2 \times \pi \times G}{Kt}, MotorCurrentLimit, ScalingCurrent\right)$
- $MaximumVoltage = R_{mot} \times MaximumCurrent + MaximumVelocity \times 60 \times G \times Kv$
- $VoltageLimit = \min(MaximumVoltage \times 1.5, MotorVoltageLimit, ScalingVoltage)$

NOTE

It is recommended that the VoltageLimit be 1.5 times the motion profiler maximum voltage to meet the motion requirements of the default stage dynamics.

23.10.12 DRV03AnalogVoltage

Motor Driver Type: DRV03AnalogVoltage

This setting of the motor driver model is used for DC motors controlled directly by the motor voltage. The motor driver interface must be set to *voltage control* (see section 23.9.19: “AnalogVoltage”) and the position corrector type to *PID with voltage output* (see section 23.8.3: “PIDDualFFVoltage”). The XPS-DRV03 driver board supplies a maximum output of 5 A and 48 V.

Driver Name: XPS-DRV03

Current Monitoring Parameters entries for the configuration file:

- ***Driver Maximum RMS Current*** — RMS current limit
- ***Driver RMS Integration Time*** — RMS integration time

The RMS current limit, *DriverMaximumRMSCurrent* (A), is the maximum allowed RMS motor current. The RMS integration time *DriverRMSIntegrationTime* (s) defines the integration time span. If the RMS motor current goes beyond this value, a driver fault is generated. The *RMS Current limit* must be greater than zero and less than or equal to 5 A. The *RMS integration time* must be greater than zero and less than or equal to 60 s.

There are many different methods to define these values. Here is a description of a method used by Newport for standard products.

Application input:

- Because high accuracy needs, a maximum temperature raise of the motor coils by 20°C is set to avoid thermal effects impacting the motion precision: $\Delta T = 20°C$.

- A ratio of 2 between the current limit (defined in the motor interface section) and the RMS current limit is used. This has been found to be a good approach in numerous precision motion applications offering a good balance between throughput and precision: $r = 2$.

Motor data

- motor torque constant: Kt (N.m/Arms)
- motor constant or steepness: S (N².m²/W)
- thermal resistance: R_{th} (°C/W)
- thermal time constant: τ_{th} (s)

Driver data

- maximum driver current: $\mathrm{I}_{\max} = 5\ \mathrm{A}$

Calculations

- $Kt = \sqrt{Rf \times S}$,

  where Rf (Ω) is the motor resistance per phase
- $\tau_{th} = \dfrac{R_{th} \times C_p^2}{\theta\tau \times S}$,

  where: C_p (N.m) is the motor peak torque and

  $\theta\tau$(°C/s) is the temperature rise at motor peak torque
- *DriverMaximumRMSCurrent*: $I_{RMS} = \min\left(\sqrt{\dfrac{\Delta T \times S}{R_{th} \times (1 + 0.004 \times \Delta t)}} \times \dfrac{1}{Kt}, \mathrm{I}_{\max}\right)$
- $CurrentLimit = \min\left(I_{RMS} \times r, \mathrm{I}_{\max}\right)$
- DriverRMSIntegrationTime: $\min(\tau_{th}, 3s)$

23.10.13 DRV03HAnalogAcceleration

Motor Driver Type: DRV03HAnalogAcceleration

This type of motor driver model is used for DC motors controlled by acceleration. The motor driver interface must be set to *acceleration control* (see section 23.9.2: "AnalogAcceleration") and the position corrector type to *PID with acceleration output* (see section 23.8.4: "PIDFFAcceleration"). The XPS-DRV03H driver board supplies a maximum output of 1.58 A and 48 V.

Driver Name: XPS-DRV03H

For all other configuration parameters, refer to section 23.10.10: "*XPS-DRV03 for acceleration control (DRV03AnalogAcceleration)*".

23.10.14 DRV03HAnalogVelocity

Motor Driver Type: DRV03HAnalogVelocity

This type of motor driver model is used for DC motors with tachometer. The driver command interface must be set to *velocity control* (see section 23.9.18: "AnalogVelocity") and the position servo loop type to *PID with velocity output* (see section 23.8.5: "PIDFFVelocity").The XPS-DRV03H driver board supplies a maximum output of 1.58 A and 48 V.

Driver Name: XPS-DRV03H

For all other configuration parameters, refer to section 23.10.11: "*DRV03AnalogVelocity*".

23.10.15 DRV03HAnalogVoltage

Motor Driver Type: DRV03HAnalogVoltage

This setting of the motor driver model is used for DC motors controlled directly by the motor voltage. The motor driver interface must be set to *voltage control* (see section 23.9.19: "AnalogVoltage") and the position corrector type to *PID with voltage output* (see section 23.8.3: "PIDDualFFVoltage"). The XPS-DRV03H driver board supplies a maximum output of 1.58 A and 48 V.

Driver Name: XPS-DRV03H

For other configuration file parameters, refer to section 23.10.12: "*DRV03AnalogVoltage*".

23.10.16 DRVM1AnalogVelocity

Motor Driver Type: DRVM1AnalogVelocity

NO LONGER SUPPORTED.

Driver Name: XPS-DRVM1

23.10.17 DRVM2AnalogVelocity

Motor Driver Type: DRVM2AnalogVelocity

NO LONGER SUPPORTED.

Driver Name: XPS-DRVM2

23.10.18 DRVM3AnalogVelocity

Motor Driver Type: DRVM3AnalogVelocity

NO LONGER SUPPORTED.

Driver Name: XPS-DRVM3

23.10.19 DRVM4AnalogAcceleration

Motor Driver Type: DRVM4AnalogAcceleration

NO LONGER SUPPORTED.

Driver Name: XPS-DRVM4

23.10.20 DRVM5AnalogVelocity

Motor Driver Type: DRVM5AnalogVelocity

NO LONGER SUPPORTED.

Driver Name: XPS-DRVM4

23.10.21 DRVP1AnalogPositionPiezo

Motor Driver Type: DRVP1AnalogPositionPiezo

This type of DriverName is used with a piezo driver card. This setting of the motor driver model is compatible only with:

- AnalogPositionPiezo driver command interfaces,
- NoEncoderPosition or PIPosition corrector type.

Driver Name: XPS-DRVP1

Driver parameters

- DriverNotchFrequency
- DriverNotchBandwidth
- DriverNotchGain
- DriverLowpassFrequency
- DriverKI
- DriverFatalFollowingError
- DriverStagePositionOffset
- DriverTravelCorrection

For more information, refer to the XPS-DRVP1 User's Manual.

23.10.22 EDBL

Motor Driver Type: EDBL

This type of motor driver model is used for brushless linear or rotary motors. The motor driver interface must be set to *120° UV phase acceleration control* (see section 23.9.13: "AnalogSin120Acceleration") or *120° UV phase dual output acceleration control* (see section 23.9.15: "AnalogDualSin120Acceleration") and the position corrector type to *PID with acceleration output* (see section 23.8.4: "PIDFFAcceleration"). The XPS-EDBL, external driver module, supplies a maximum output of 25 A and 96 Vpp.

Driver Name: XPS-EDBL

Driver Supply Voltage

For the XPS-EDBL driver, the parameter "Driver Supply Voltage" is fixed to 96 V and is not modifiable.

For all other configuration file parameters refer to section 23.10.8: "DRV02".

23.11 Stage

In this configuration category, users are building the Global stage parameters section of the stages.ini file.

Example:

```
; --- Global stage parameters
; --- <Stage.GenericInformation>
SmartStageName = XMS160
Unit = mm
ConfigurationComment = No load
```

23.11.1 Type: GenericInformation

Configuration Comment

Use this parameter to write a comment in the stages.ini file

Unit

Use this parameter to display the unit (such as mm) that is reference in the configuration file for parameters with units of length.

Smart Stage Name

Use this parameter of the stage as a Newport's exclusive ESP technology to define the EEPROM smart stage name.

Service Form

Your Local Representative

Tel.: ____________________

Fax: ____________________

Name: ____________________

Company: ____________________

Address: ____________________

Country: ____________________

P.O. Number: ____________________

Item(s) Being Returned: ____________________

Model#: ____________________

Return authorization #: ____________________
(Please obtain prior to return of item)

Date: ____________________

Phone Number: ____________________

Fax Number: ____________________

Serial #: ____________________

Description: ____________________

Reasons of return of goods (please list any specific problems): ____________________

Newport®

Visit Newport Online at:

www.newport.com

North America & Asia
Newport Corporation
1791 Deere Ave.
Irvine, CA 92606, USA

Sales
Tel.: (800) 222-6440
e-mail: sales@newport.com

Technical Support
Tel.: (800) 222-6440
e-mail: tech@newport.com

Service, RMAs & Returns
Tel.: (800) 222-6440
e-mail: service@newport.com

Europe
MICRO-CONTROLE Spectra-Physics S.A.S
9, rue du Bois Sauvage
91055 Évry CEDEX
France

Sales
Tel.: +33 (0)1.60.91.68.68
e-mail: france@newport.com

Technical Support
e-mail: tech_europe@newport.com

Service & Returns
Tel.: +33 (0)2.38.40.51.55

mks

Newport® Ophir® Spectra-Physics®