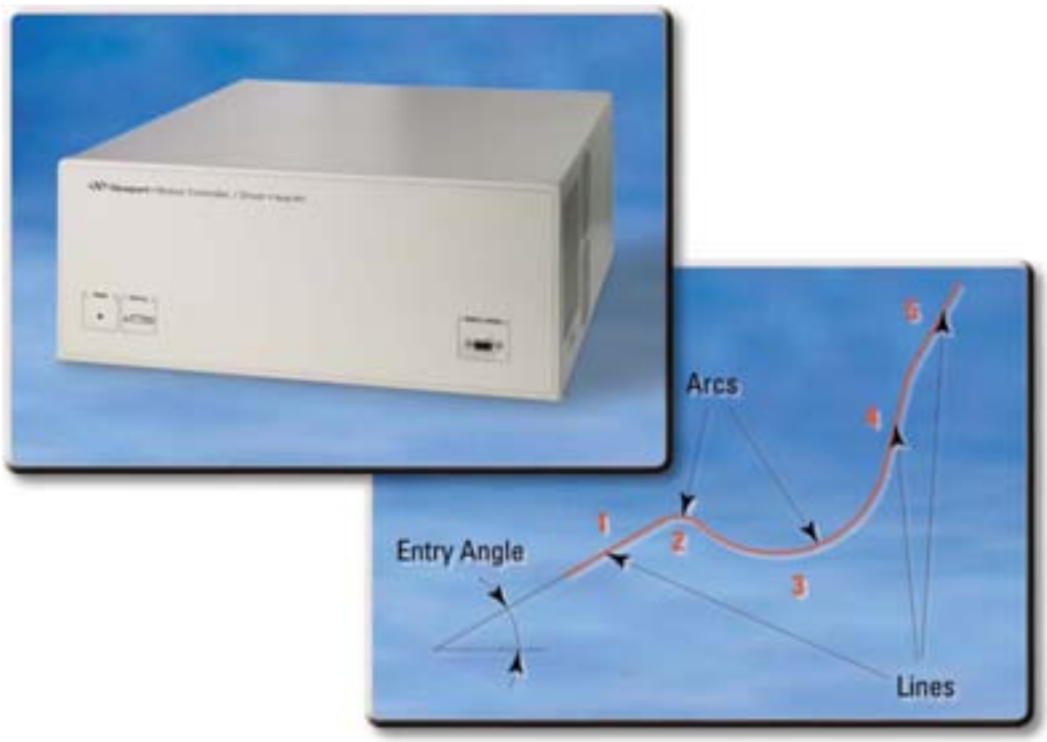
XPS Controller

Universal High-Performance Motion Controller/Driver



User's Manual
Software Tools
Tutorial
V2.6.x

Newport®
Experience | Solutions

For Motion, Think Newport

Table of Contents

User's Manual

Software Tools

Motion Tutorial

Appendices

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.

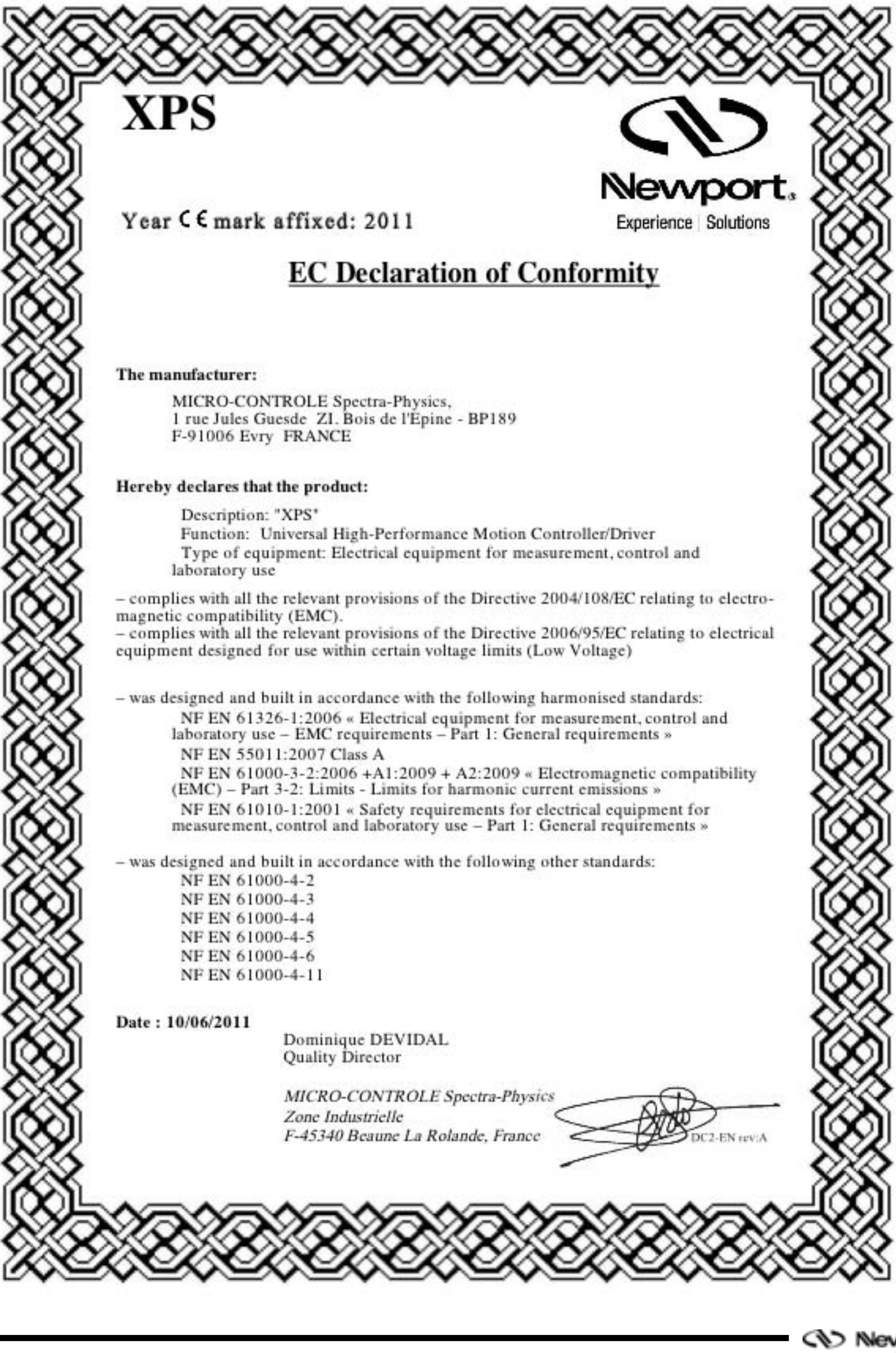
Limitation of Warranty

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.

First printing 2003

EU Declaration of Conformity



XPS

Newport.
Experience | Solutions

Year CE mark affixed: 2011

EC Declaration of Conformity

The manufacturer:

MICRO-CONTROLE Spectra-Physics,
1 rue Jules Guesde ZI. Bois de l'Epine - BP189
F-91006 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 2004/108/EC relating to electro-magnetic compatibility (EMC).
– complies with all the relevant provisions of the Directive 2006/95/EC relating to electrical equipment designed for use within certain voltage limits (Low Voltage)

– was designed and built in accordance with the following harmonised standards:
NF EN 61326-1:2006 « Electrical equipment for measurement, control and laboratory use – EMC requirements – Part 1: General requirements »
NF EN 55011:2007 Class A
NF EN 61000-3-2:2006 +A1:2009 + A2:2009 « Electromagnetic compatibility (EMC) – Part 3-2: Limits - Limits for harmonic current emissions »
NF EN 61010-1:2001 « 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 : 10/06/2011

Dominique DEVIDAL
Quality Director

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

[illegible]

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.: (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: rma.service@newport.com

Europe
MICRO-CONTROLE Spectra-Physics S.A.S
1, rue Jules Guesde – Bât. B
ZI Bois de l'Épine – BP189
91006 Evry Cedex
France

Sales France
Tel.: +33 (0)1.60.91.68.68
e-mail: france@newport-fr.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 have been assigned an RMA number by Newport. Assignment of the RMA requires the item serial number.

Packaging

XPS Series Controller/Driver being returned under an RMA must be securely packaged for shipment. If possible, re-use the original factory 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 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** and **XPS User’s Manual** are delivered as paper copies with the controller.

Programmer’s manual, TCL manual, Software Drivers manual and Stage Configuration manual are PDF files accessible from the XPS web site.

DLLs and corresponding sources are available from the controller disk in the folder Public/Drivers/DLL. DLLs can also be downloaded through FTP.

LabView VIs with examples are also available on the controller disk in the folder Public/Drivers/LabView. They can be downloaded through FTP.

To connect through FTP, please see chapter 5: “FTP connection”.

The first part of this manual is the getting-started part of the system. It serves as an introduction and 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 includes also an introduction to FTP connections and some general guidelines for troubleshooting, maintenance and service:

4. Software Tools

5. FTP connection

6. Maintenance and Service

The third part provides an exhaustive description of the XPS architecture, its features and capabilities. Different than 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 more complete list of specifications for the different features. It includes:

7. XPS Architecture

8. Motion

9. Trajectories

10. Compensation

11. Event Triggers

12. Data Gathering

13. Triggers

14. Control Loops

15. Analog Encoder Calibration

16. 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

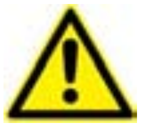


Figure 1: General Warning or Caution Symbol.

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

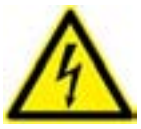


Figure 2: Electrical Shock Symbol.

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 voltage. Any mishandling could result in damage to the equipment, personal injury, or death.

1.2.3 European Union CE Mark

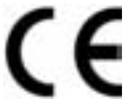


Figure 3: CE Mark.

The presence of the CE Mark on Newport Corporation equipment means that it has been designed, tested and certified as complying with all 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

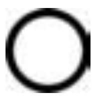


Figure 5: "OFF" Symbol.

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.

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 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 selection of the proper input voltage.

Only plug the Controller/Driver unit into a grounded power outlet.

Assure 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.

The system must be installed in such a way that power switch and power inlet remains accessible to the user.

Disconnect or do not plug in the AC power cord in the following circumstances:

– 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.

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 8 axes of stepper, DC brush or DC brushless motors using internal drives Other motion devices using external third-party drives
Communication Interfaces	Internet protocol TCP/IP One Ethernet 10/100 Base-T (RJ45 connector) with fixed IP address for local communication One Ethernet 10/100 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 Optional XPS-RC remote control
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. 2.5 MHz rate, programmable filter Time spaced trigger output pulses, 0.02 Hz to 2.5 MHz rate, 50 ns accuracy Trigger output on trajectories with 100 μs resolution Data gathering at up to 10 kHz rate, up to 1,000,000 data entries User-defined "actions at events" monitored by the controller autonomously at a rate of 10 kHz 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-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) Gantry mode including XY gantries with variable load ratio 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	10 kHz
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

I/O	30 TTL inputs and 30 TTL outputs (open-collector) 4 synch. analog inputs ±10 V, 14 Bit 4 synch. uncommitted analog outputs, 16 Bit Watchdog timer and remote interlock
Trigger In	Hardware latch of all positions and all analog I/O's; 10 kHz max. frequency <50 ns latency on positions <100 μs time jitter on analog I/O's
Trigger Out	One high-speed position compare output per axes that can be either configured for position synchronized pulses or for time synchronized pulses : <50 ns accuracy/latency, 2.5 MHz max. rate
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 x32768 interpolation used for servo; amplitude, phase and offset correction; additional 2nd hardware interpolator used for synchronization; up to x200 interpolation 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) Analog position (used with external drives for example for piezo control) 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-DRV00 and external stepper motor driver) 500 W @ 230 VAC and 425 W @ 115 VAC total available power
Dimensions (W x D x H)	19" – 4U, L: 508 mm
Weight	15 kg max

2.2 Drive Options

The XPS controller is capable of driving up to 8 axes of most Newport positioners with internal drives that slide through the back of the chassis. These factory tested drives are powered by an internal 500 W power supply which is independent of the controller power supply.

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. When used with Newport stages, the configuration of the amplifier is easy using the auto-configuration utility software. Advanced users can also manually develop their own configuration files specifically optimized for each application.

The XPS-DRV01 motor driver supplies a maximum current of 3 Amps and 48 Volts. 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 XM2000 and IMS-LM linear motor 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 encoder input signals used also for motor commutation. Motor initialization is done by a special routine that forces the motor to a known magnetic position without the need for Hall or other sensors.

The XPS-DRV03 is a fully numerical, programmable PWM-Amplifier that has been optimized for the 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 Amps and 48 Volts. 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 free-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-DRV00 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 brushless motors, voice coils and piezoelectric stages.

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 32,768-fold. For example, when used with a scale with 4 μm signal period the resolution can be as fine as 0.122 nm. This interpolator can be used for accurate position feedback on the servo corrector of the system. An additional hardware interpolator with 40 MHz clock frequency and programmable signal subdivision up to 200-fold is used for synchronization purposes. This fast interpolator latches directly the position with less than 50 ns latency and provides a much higher level of precision for synchronization than alternative time based systems. And unlike most high-resolution multiplication devices the XPS interpolators do not compromise positioning speed. With a maximum input frequency ranging from 180 kHz to 400 kHz (depending on the interpolation factor) the maximum speed of a stage with a 20 μm signal period scale can be up to 3.6 m/s.

2.3 Compatible Newport Positioners and Drive Power Consumption

The list of all Newport positioners that are directly compatible with the XPS controller and the corresponding drive module needed are available from the Newport catalog or the web site.

2.4 XPS Hardware Overview

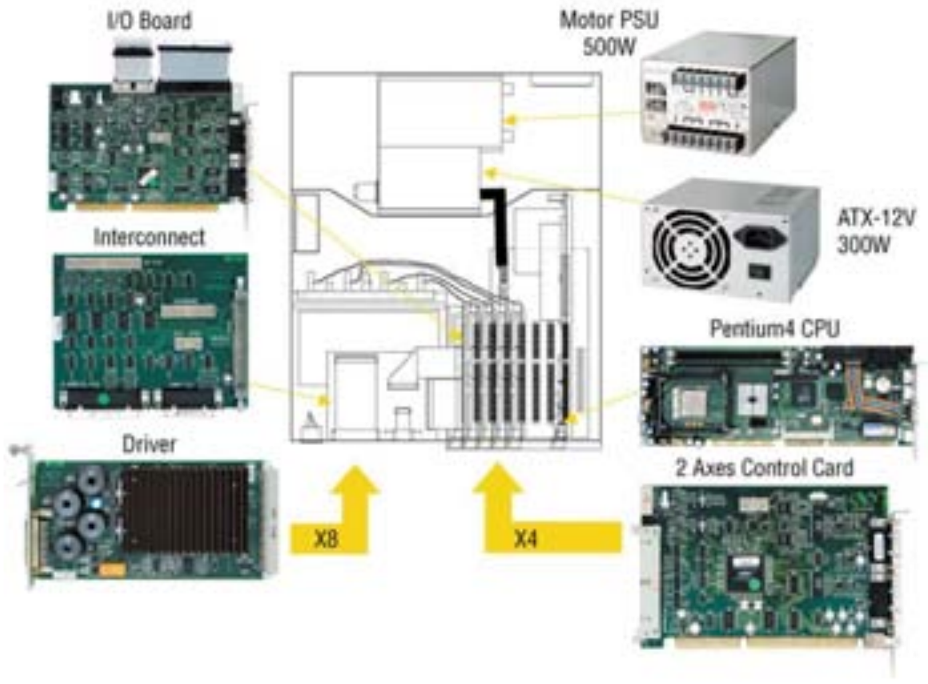


Figure 6: XPS Hardware Overview.

2.5 Front Panel Description

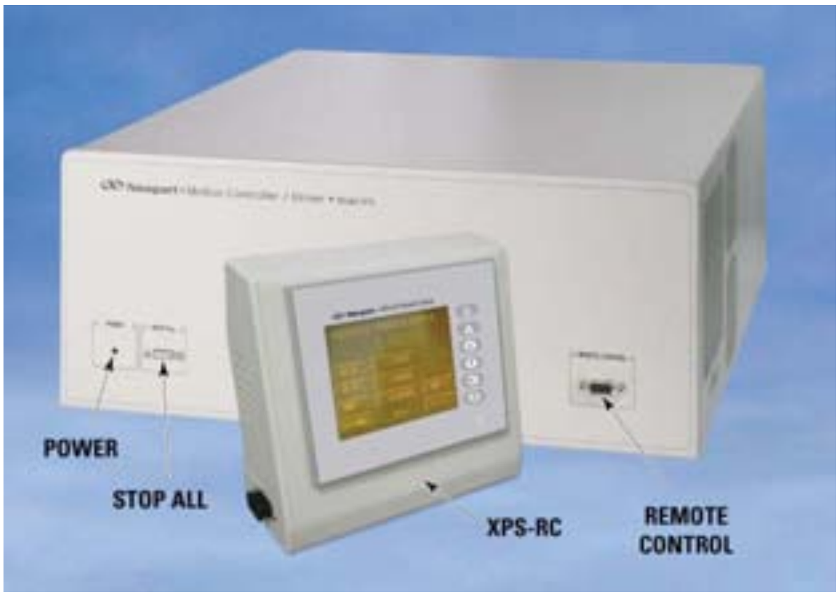


Figure 7: Front Panel of XPS Controller/Driver.

The XPS-RC Remote Control plugs into the front panel of the XPS controller to enable computer-independent motion and basic system diagnostic. For more information, refer to the XPS data sheet and the manual that is supplied with the XPS-RC.

2.6 Rear Panel Description

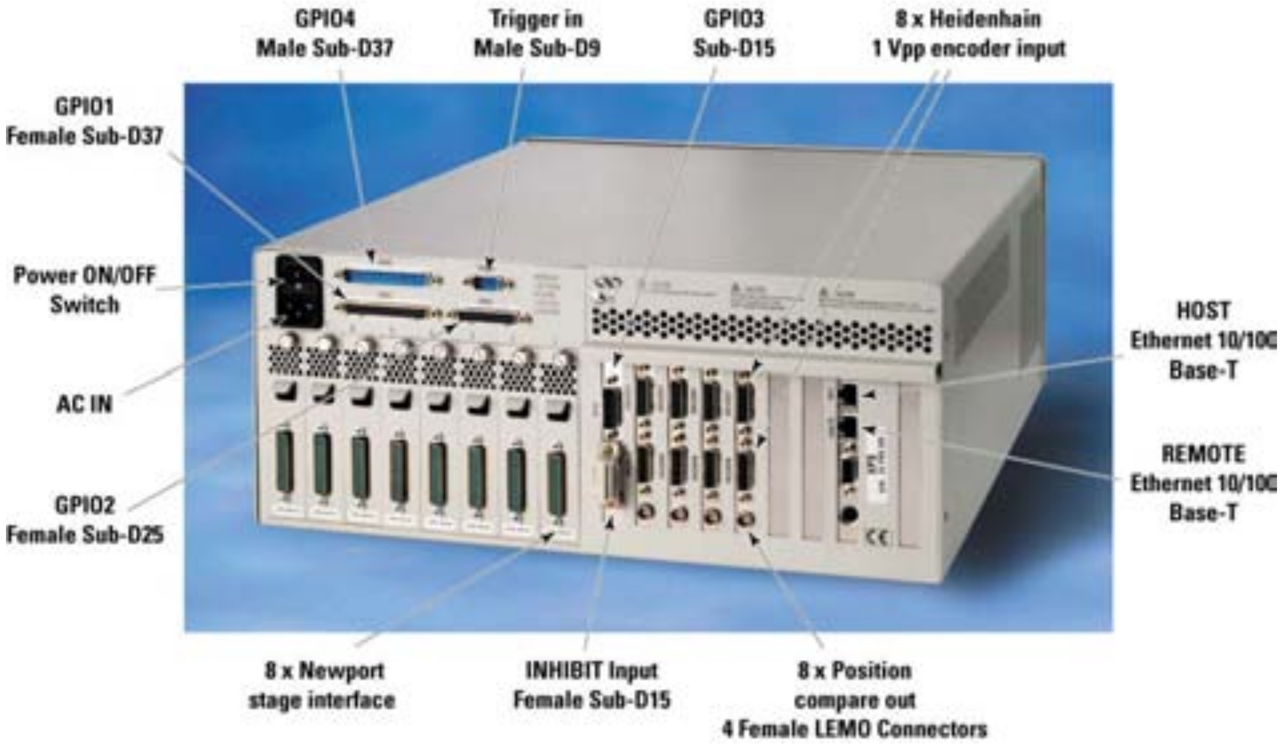


Figure 8: Rear Panel of XPS Controller/Driver.

NOTE

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

2.6.1 Axis Connectors (AXIS 1 – AXIS 8)

Each installed axis driver card features a connector to attach a cable 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 for the motion devices you intend to connect. Serious damage could occur if a stage is connected to the wrong driver card.

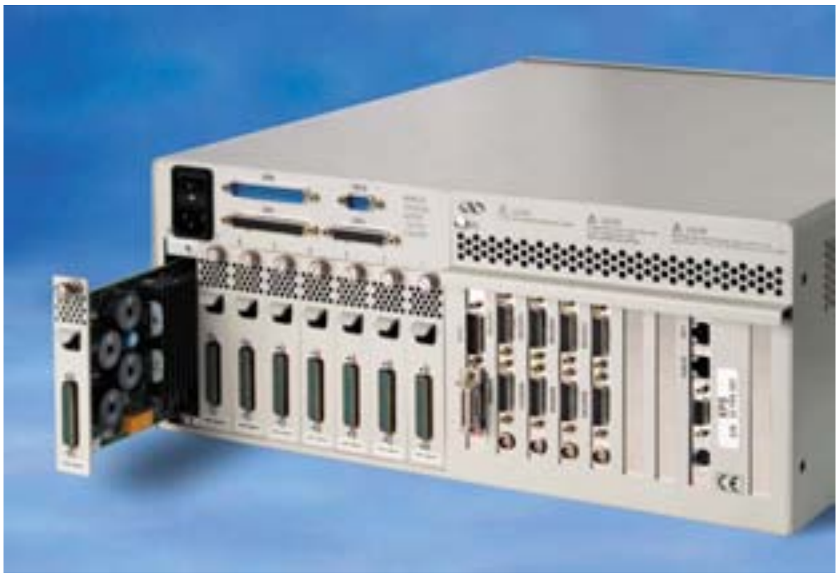


Figure 9: Axis Driver Card.

Please see next section for installation instructions.

NOTE

Power Input: 100–240 V, 50–60 Hz, 11 A–5.5 A

2.7 Ethernet Configuration

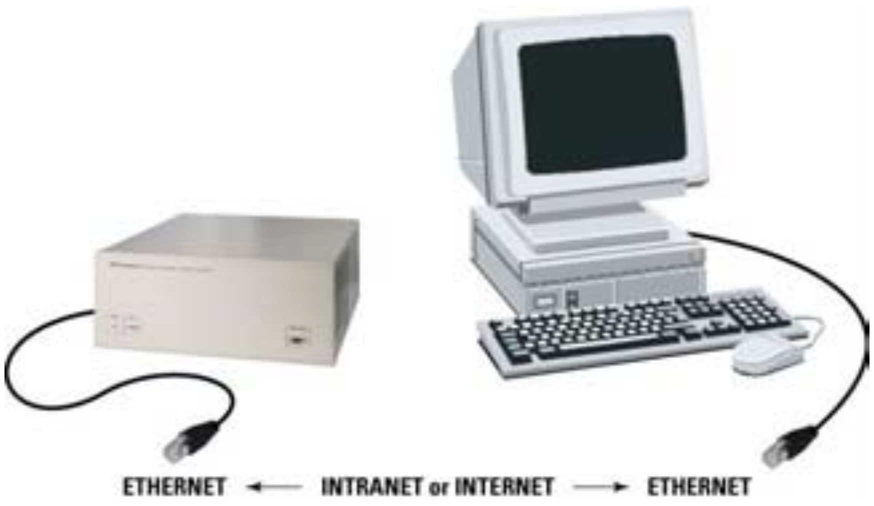


Figure 10: Ethernet Configuration.

2.7.1 Communication Protocols

The Ethernet is 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. 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.

2.7.2 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").

2.8 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 manages independently and concurrently 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 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 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 independently from different workstations without any delays or cross-talk.

The XPS controller uses TCP/IP blocking sockets, which means that commands to the same socket are "blocked" until the XPS gives a 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 sockets as needed. Please refer to section 16.4: "Running Processes in Parallel" for further information about sockets and parallel processing.

2.9 Programming with TCL

TCL documentation is available in a PDF file accessible from the XPS controller web site.

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 XPS allows them to include any function to 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 VxWorks hardware real time multitasking 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, communication tasks or when the maximum available time of 20 ms for each TCL program is over.

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 XPS provides 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 web-site, and to confirm 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 hazards to you. 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:

User's Manual and Motion Tutorial.

XPS controller.

Cross-over cable, gray, 3 meters.

Straight-through cable, black, 5 meters.

Power cord.

Rack mount ears and handles.

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

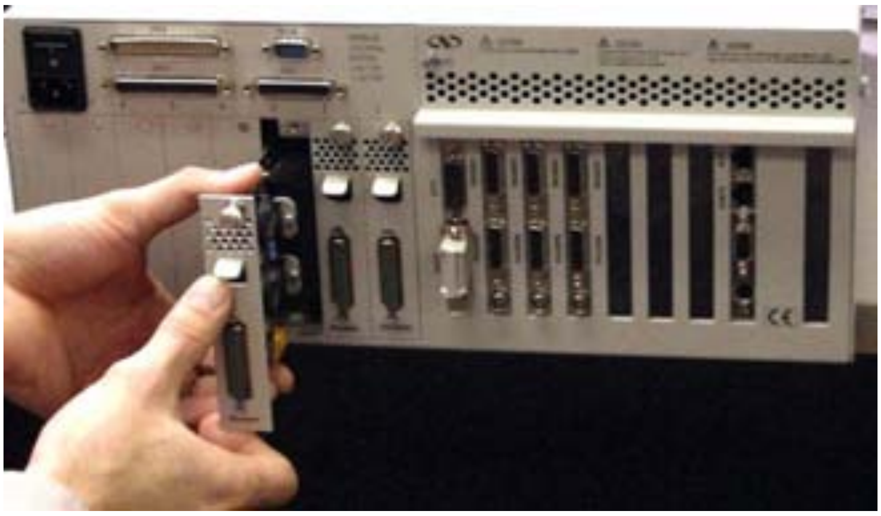


Figure 11: Installing Driver cards.

Due to the high power available in the XPS controller (300 W for the CPU and 500 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 the right (driver 1) to the left (driver 8) when looking at the controller from the back.
3. If 8 boards or less are used, the remaining slots must disabled with the appropriate covers that were delivered with the controller.
4. The surrounding ventilation holes at the sides and back of the XPS rack must be free from obstructions that prevent 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 remain accessible by the user.

After the main power is switched on, the LED on the front panel of the XPS will turn green.

There is an initial beep after power on and a second beep when the controller has finished booting. The time between the first and the second beep can be 1-2 minutes.

There is also a STOP ALL button on the front panel that is used to shut down all the motors.

3.5 Connecting to the XPS

The Newport's XPS Controller/Driver is a multi-axis motion controller system that is based on a high performance 10/100 base T Ethernet connection using CAT5 cable.

The controller can be connected in 2 different ways:

1. Direct connection-PC to XPS through a cross over cable (gray).
2. Corporate Network connection – requires input from Network Administrator (black).

Two cables are provided with the motion controller:

Cross-over cable – used when connecting XPS directly to a PC.

Straight Ethernet cable – used when connecting 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.

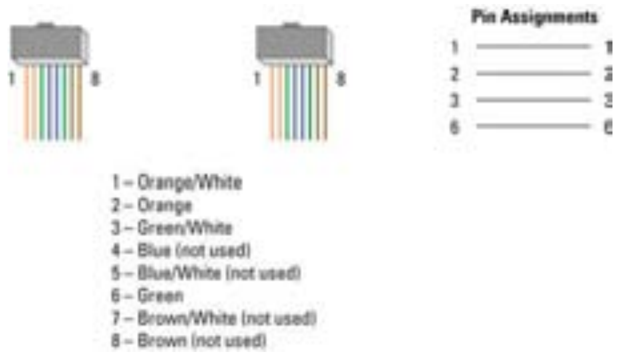


Figure 12: Straight through cables.

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.

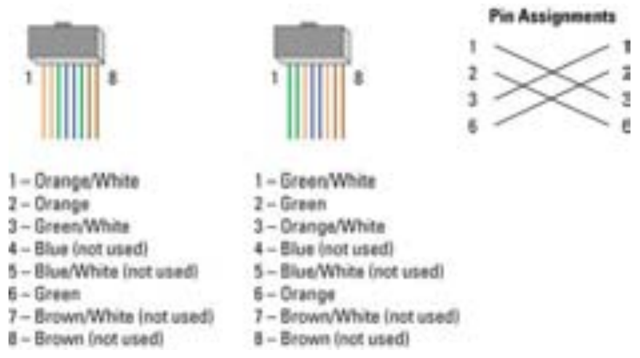


Figure 13: Ethernet Cross Over Cables.

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 gray cross-over cable and the HOST connector at the back of the XPS mainframe.

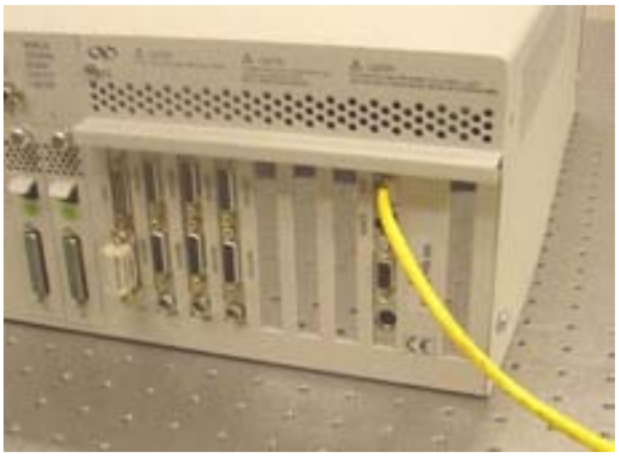
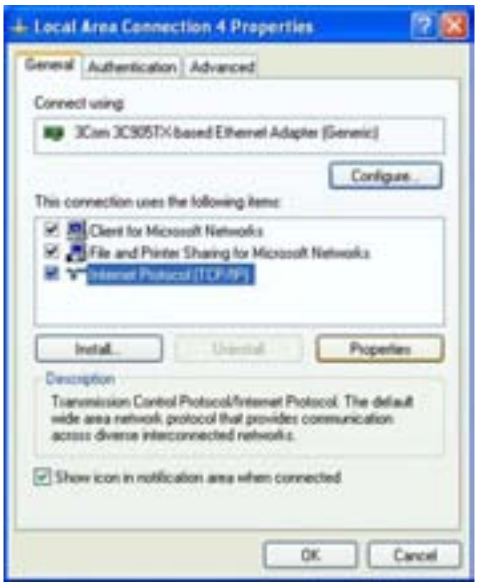


Figure 14: Direct Connection to the XPS using cross-over cable.

First, the IP address on the PC's Ethernet card has to be set to match the default factory XPS's IP address (192.168.0.254). Here is the procedure to set the Ethernet card address.

This procedure is for the Windows XP operating system:

1. Start Button > Control Panel > Network Connections.
2. Right Click on Local Area Connection Icon and select Properties.



Local Area Connection 4 Properties

General | Authentication | Advanced

Connect using:

3Com 3C905TX-based Ethernet Adapter (Generic)

This connection uses the following items:

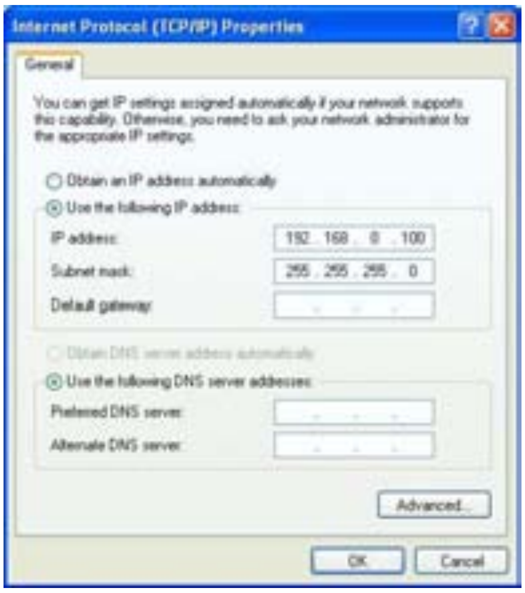
- [✓] Client for Microsoft Networks
- [✓] File and Printer Sharing for Microsoft Networks
- [✓] Internet Protocol (TCP/IP)

Description

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

[✓] Show icon in notification area when connected

3. Highlight Internet Protocol (TCP/IP) and click on Properties.
4. Type the following IP address and Subnet Mask as shown in the next window.


Internet Protocol (TCP/IP) Properties

General

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:	192 . 168 . 0 . 100
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	. . .

○ Obtain DNS server address automatically
◉ Use the following DNS server addresses:

Preferred DNS server:	. . .
Alternate DNS server:	. . .

5. Click "OK".

NOTE

The Last number of the IP address can be set to any number between 2 to 253: 100 for example.

NOTE

When configuring the controller to be on the network, the settings for the PC's Ethernet card will have to be set back to default under "Obtain an IP address automatically".

Once the Ethernet card address is set, you are ready to connect to the XPS controller. Following is the procedure for connecting to the controller:

6. Open Internet Browser and connect to http://192.168.0.254

Login:

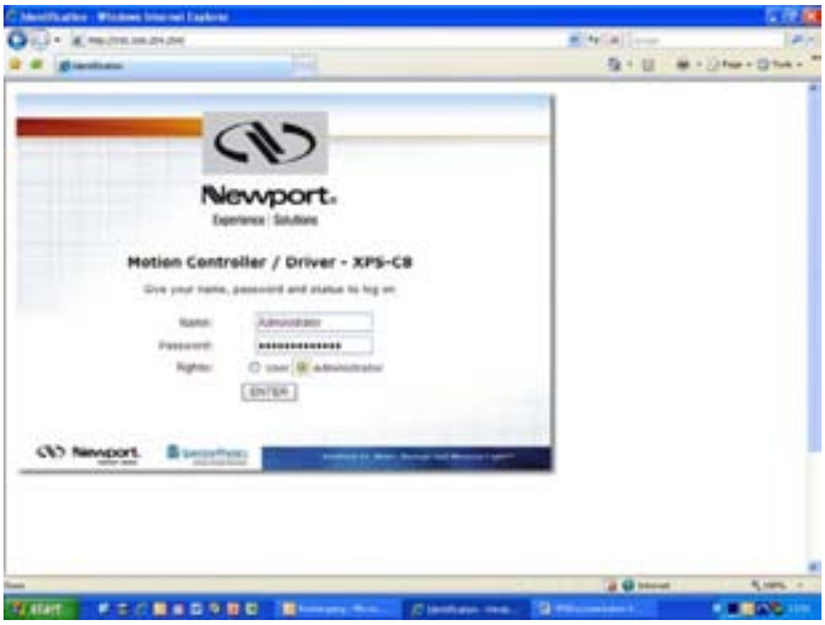
Name: **Administrator**

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

Rights: **Administrator**

NOTE

Please note that the login is case sensitive.


Motion Controller / Driver - XPS-C8

Give your name, password and status to log on

Name: Administrator

Password: **************

Rights: ○ User ● Administrator

ENTER

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

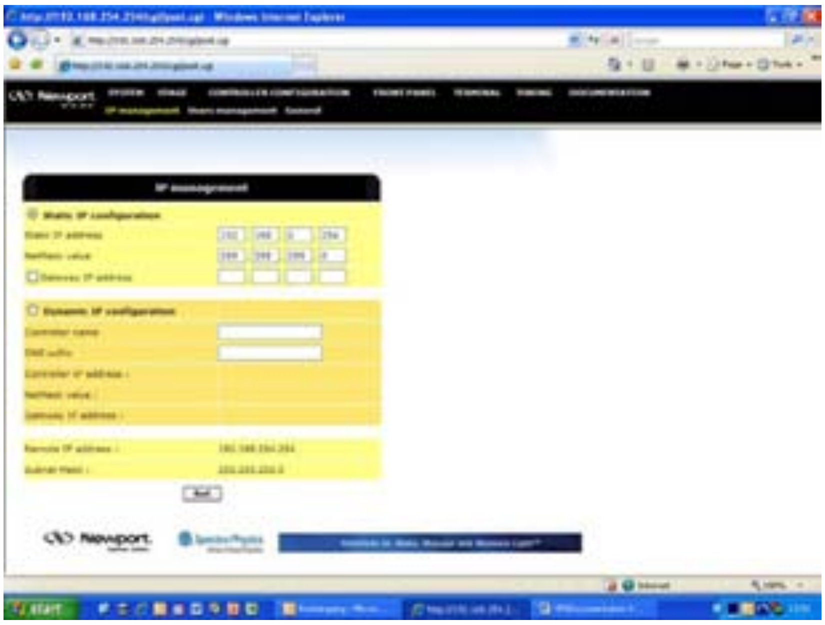
If you don't want to connect the XPS controller through a Corporate Network you may skip to section 3.7: "Connecting the Stages".

NOTE

If you want to change the IP address of the XPS controller, follow the next section explanations, but 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 previous 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-CONFIGURATION" of the web-site and select the sub-menu "IP-Management".



The static IP address, the subnet mask and the Gateway IP address have to be provided by your Network Administrator to avoid network conflicts. Once you have obtained 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 than 192.168.254…

NOTE

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

Once the appropriate addresses for the Static IP configuration are set, click on SET and switch off controller.

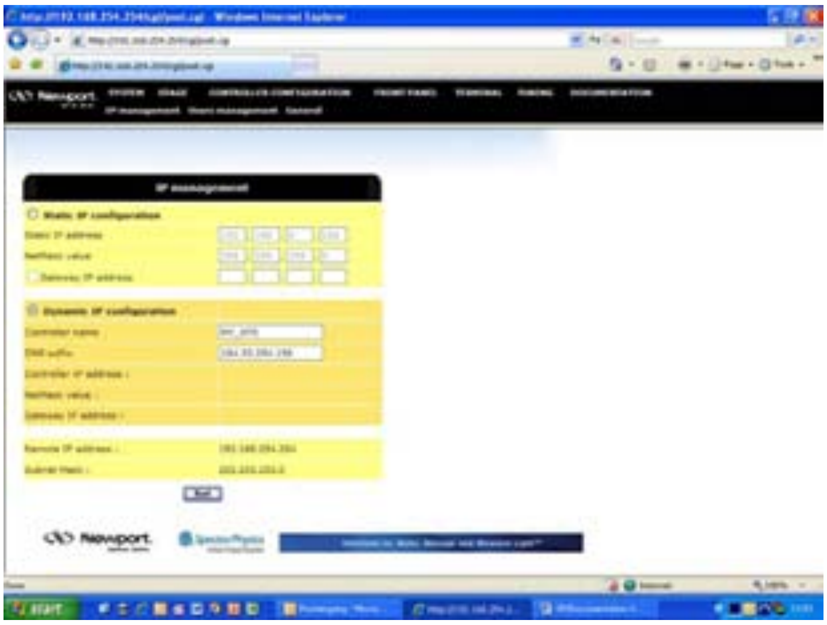
Connect now CAT-5 network cable (black) to the HOST connector of the XPS controller and to your network.

After restarting controller and restoring your PC's Ethernet card default configuration, open the Internet browser and connect using your given Static IP address.

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

3.5.5 Connecting The XPS to a Corporate Network Using Dynamic IP Configuration

Obtain DNS suffix from network administrator for the "DNS suffix" field.



IP management

Static IP configuration
Static IP address
NetMask value
Gateway IP address

Dynamic IP configuration
Controller name
DNS suffix
Controller IP address :
NetMask value :
Gateway IP address :

Remote IP address :
Subnet mask :

Assign any name for the "Controller name" field.

Click on SET and switch off the controller.

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

After restarting the controller, open the internet browser and connect using your controller name. You may skip to section 3.7: "Connecting the Stages".

NOTE

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

3.5.6 Recovering lost IP configuration

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

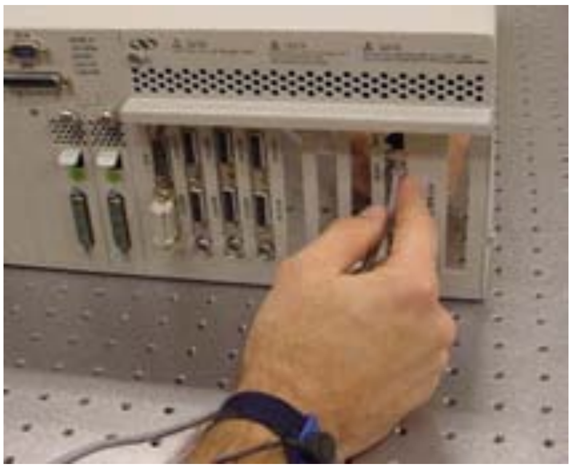
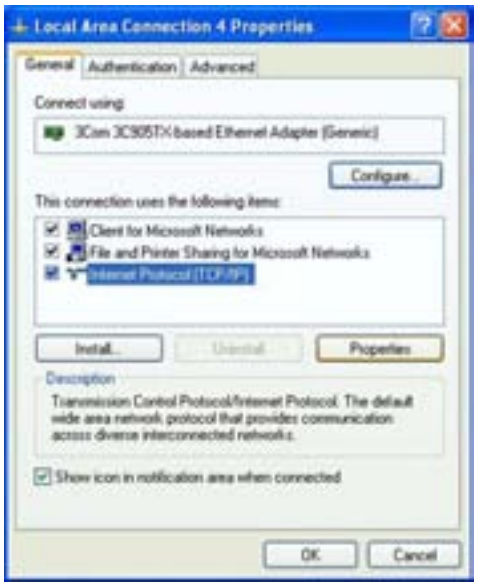


Figure 15: Direct Connection to the XPS using cross-over cable and REMOTE connector.

First, the IP address on the PC's Ethernet card has to be set to match the XPS's fixed IP address for the REMOTE plug (192.168.254.254). Here is the procedure to set the Ethernet card address.

This procedure is for the Windows XP operating system:

1. Start Button > Control Panel > Network Connections.
2. Right Click on Local Area Connection Icon and select Properties.



Local Area Connection 4 Properties

General | Authentication | Advanced

Connect using:

3Com 3C905TX-based Ethernet Adapter (Generic)

Configure...

This connection uses the following items:

- ☑ Client for Microsoft Networks
- ☑ File and Printer Sharing for Microsoft Networks
- ☑ Internet Protocol (TCP/IP)

Install... | Uninstall | Properties

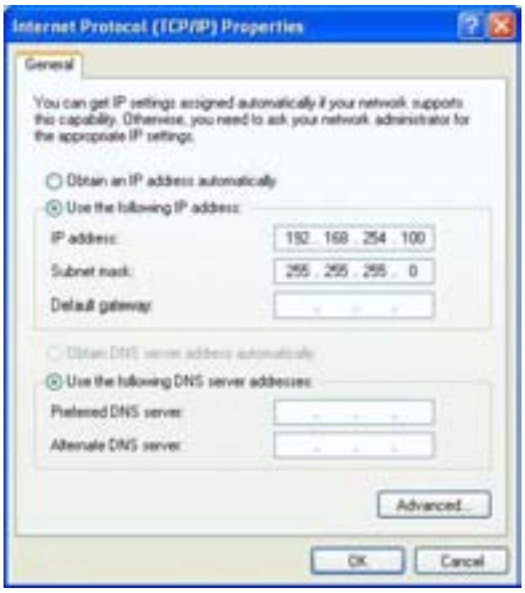
Description

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

☑ Show icon in notification area when connected

OK | Cancel

3. Highlight Internet Protocol (TCP/IP) and click on Properties.
4. Type the following IP address and Subnet Mask as shown in the next window.


Internet Protocol (TCP/IP) Properties

General

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: 192 . 168 . 254 . 100
Subnet mask: 255 . 255 . 255 . 0
Default gateway: . . .

○ Obtain DNS server address automatically
◉ Use the following DNS server addresses:

Preferred DNS server: . . .
Alternate DNS server: . . .

5. Click "OK".

NOTE

The last number of the IP address can be set to any number between 2 to 253: 100 in this example.

NOTE

When configuring the controller to be on the network, the settings for the PC's Ethernet card will have to be set back to default which is "Obtain an IP address automatically".

Once the Ethernet card address is set, you are ready to connect to the XPS controller. Following is the procedure for connecting to the controller:

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

Login:

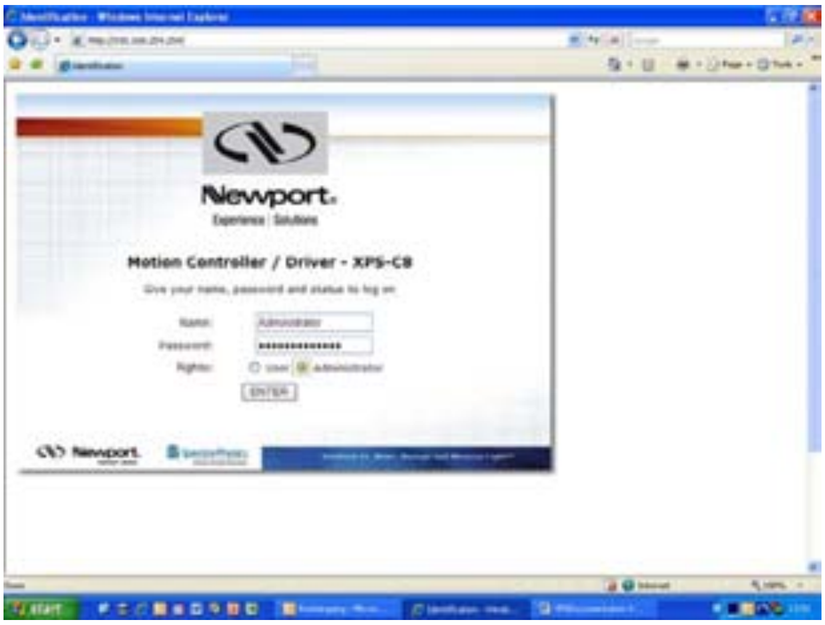
Name: **Administrator**

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

Rights: **Administrator**

NOTE

Please note that the login is case sensitive.



Motion Controller / Driver - XPS-C8

Give your name, password and status to log on

Name: Administrator
Password: ••••••••••••••
Rights: ○ User ● Administrator

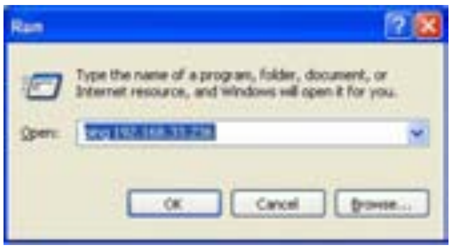
Once you are logged, you can change the IP configuration by following the steps described in section 3.5.4 or 3.5.5 depending on your configuration.

NOTE

If you want to reset the IP address to the default factory setting, follows the section 3.5.4 to set the IP address back to 192.168.0.254.

3.6 Testing your XPS-PC Connection

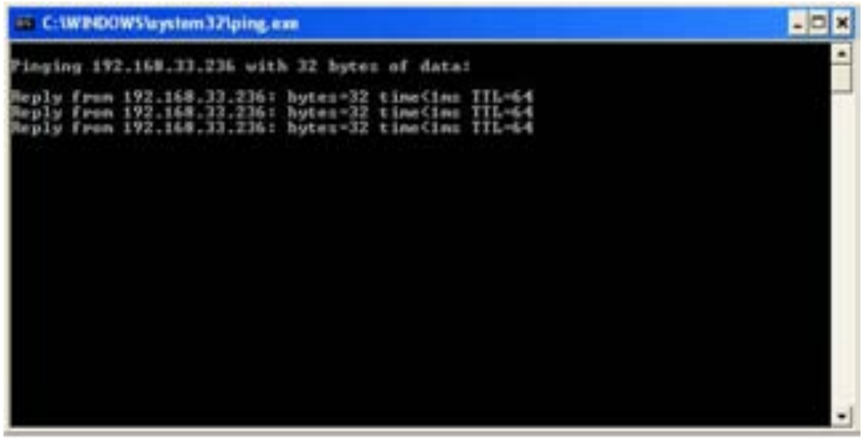
To check if the XPS is connected to the host computer, you could send a ping message from the computer to the XPS. This is done by using the menu: Start->Run->then type: ping + IP address of the XPS. See the example below for the IP address 192.168.33.236:



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

Open: ping 192.168.33.236

If the XPS is connected, it replies in the terminal window that appears when users click on the OK button:



```
Pinging 192.168.33.236 with 32 bytes of data:

Reply from 192.168.33.236: bytes=32 time<1ms TTL=64
Reply from 192.168.33.236: bytes=32 time<1ms TTL=64
Reply from 192.168.33.236: bytes=32 time<1ms TTL=64
```

If the XPS controller is not connected, the window displays that the time delay of the request is exceeded.

3.7 Connecting the Stages

CAUTION

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

CAUTION

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

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

When using stages with analog encoder interface, a separate encoder cable has to be connected to the appropriate connector of the control board labeled "Encoder 1" to "Encoder 8".

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

CAUTION

It is strongly recommended that the user read the section 3.4: "System Setup", before attempting to turn the controller or the motors 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 power and control signals (encoder, limits, and home signals). The cable is terminated with a standard 25-pin D-Sub connector.

"Dummy stages" might be used to simulate a stage. They allow users to configure and test system behavior without real stages connected.

For a dummy stage, use a male 25-pin D-Sub connector with signals for + and - travel limit connected to ground, and plug this device to the Newport stage interface (see pinout description of the motor driver connectors in appendix F). Configure your system with a number of those dummy stages. Dummy stages can be found in the stages.ini file (see Admin/Config folder of the controller) under [DUMMY_STAGE].

3.8 Configuring the Controller

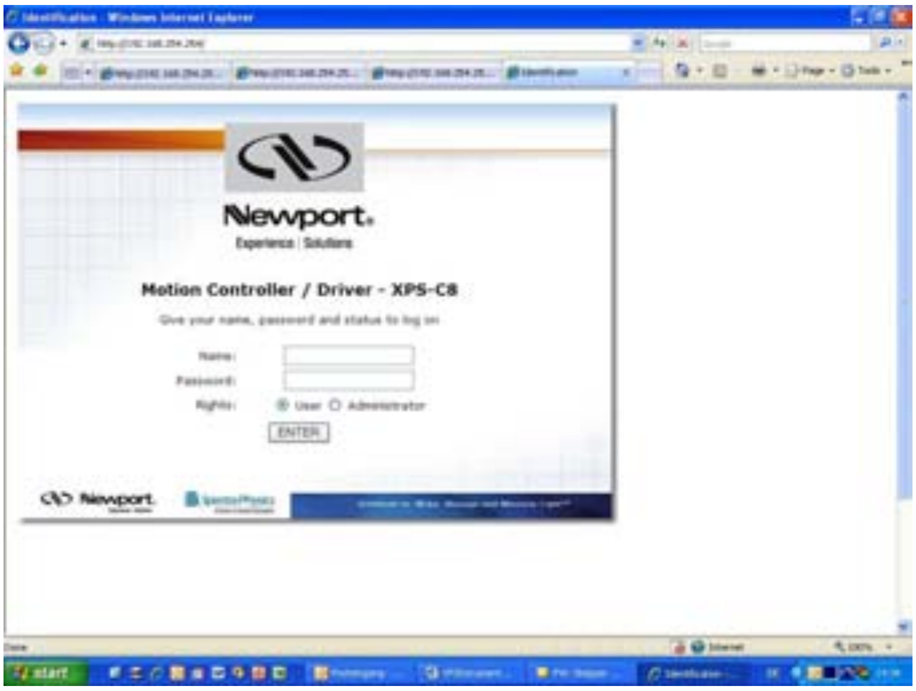
When the driver boards are installed and the IP address is configured the controller can be configured:

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. There is an initial beep a few seconds after the power on and a second beep when the controller has finished booting. The time between the first beep and the second beep is approx. 1-2 minutes.

Open an internet browser and connect to http://<your fixed IP address>


Newport
Experience | Solutions
Motion Controller / Driver - XPS-C8
Give your name, password and status to log on
Name:
Password:
Rights:
User
Administrator
ENTER

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

There are two possibilities to configure the controller: Auto configuration and manual configuration. Auto Configuration is the simplest method to configure the controller, but has some limitations:

Auto configuration works only with Newport ESP compatible positioners.

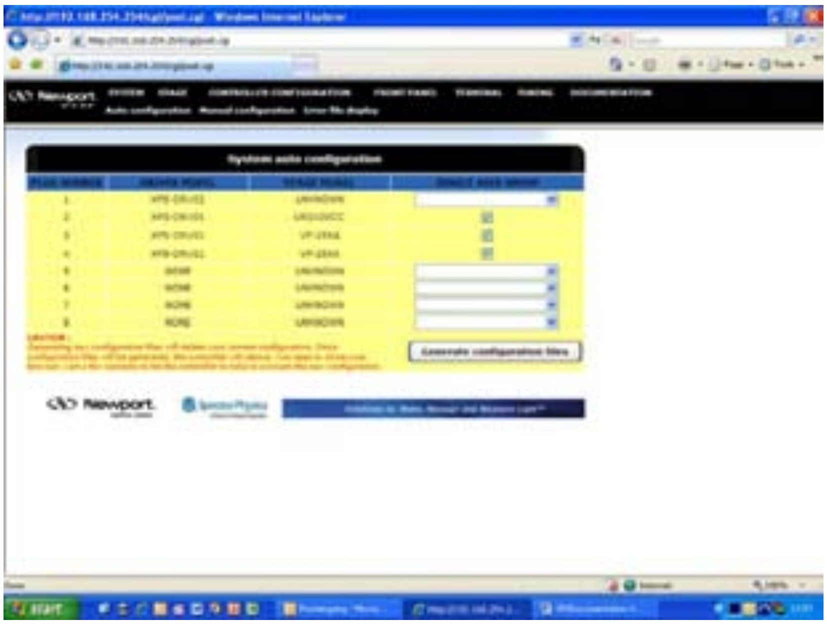
Auto configuration configures all detected positioners as single axis groups. But, single axis groups provide only limited functionality (no synchronized motion, no trajectories, no XY or XYZ compensation). To take full benefit of the capabilities of the XPS controller, a manual configuration is needed.

For Non-Newport stages or very old Newport stages, manual configuration is required. See document ConfigurationWizard.pdf for details. This document is accessible from the XPS web tools under the tag DOCUMENTATION.

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 Auto Configuration

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



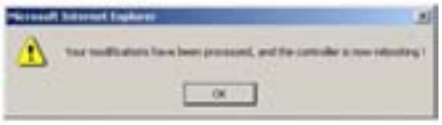
System auto configuration

PLUG NUMBER	DRIVER MODEL	STAGE MODEL	SINGLE AXIS GROUP
1	XPS-DRV01	UNKNOWN	
2	XPS-DRV01	URS100CC	✓
3	XPS-DRV01	VP-25XA	✓
4	XPS-DRV01	VP-25XA	✓
5	NONE	UNKNOWN	
6	NONE	UNKNOWN	
7	NONE	UNKNOWN	
8	NONE	UNKNOWN	

CAUTION : [illegible]

Check, if all connected stages are recognized by the system. If yes, click on "GENERATE CONFIGURATION FILES".

The controller reboots and the following screen appears (this action may take up to two minutes):

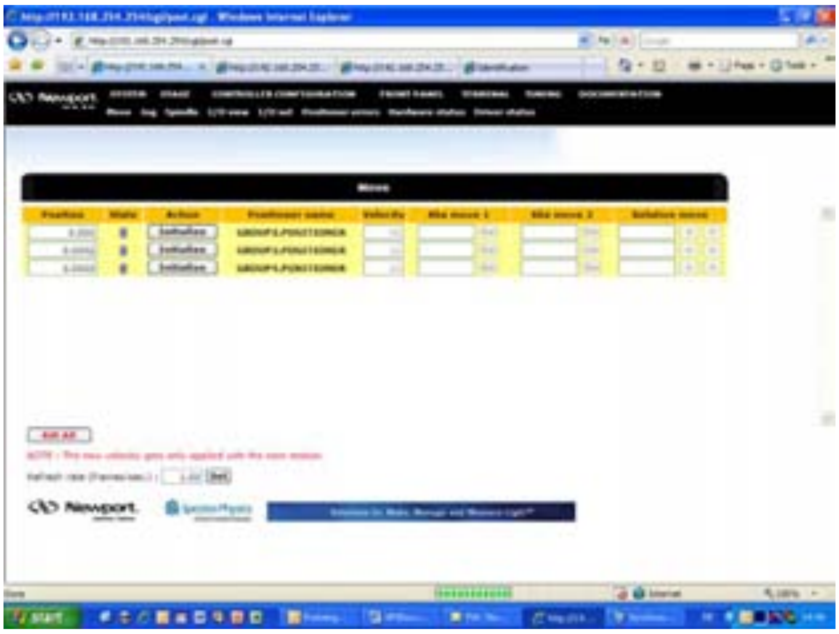


Microsoft Internet Explorer

Your modifications have been processed, and the controller is now rebooting !

Click "OK".

When the controller has finished booting (hear second beep after 1-2 minutes), press "F5" to reload the page, select **FRONT PANEL**, and then select **"Move"**. The following screen appears:



Position	State	Action	Positioner name	Velocity	Abs move 1	Abs move 2	Relative move
[illegible]	0	Initialize	[illegible]				
[illegible]	0	Initialize	[illegible]				
[illegible]	0	Initialize	[illegible]				

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 proceed reading the rest of this manual.

NOTE

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

3.8.2 Manual Configuration for Newport Positioners

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

For a manual configuration, first users need to build their personal stage data base using the web tool "**Add from Database**" under the main tag **STAGE**. 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 using a different name. That ways, for instance, it is possible adding 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.

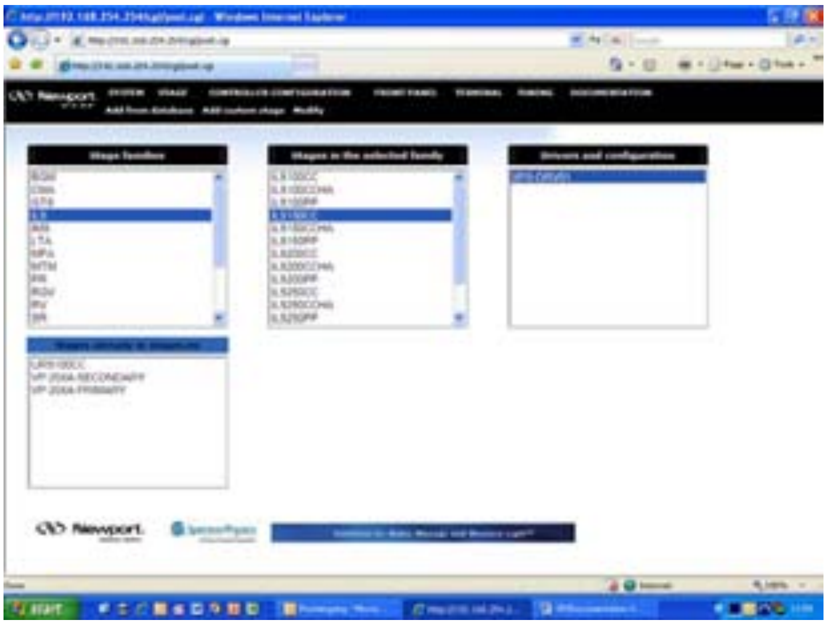
All stage parameters can be modified using the Web Tool **"Modify"** under the main tag **STAGE.** Alternative, the stage parameters can be modified directly in the stages.ini file using a text editor. The stages.ini file is located in the Config folder of the XPS controller. This folder is accessible via ftp, see chapter 5 for details.

When all stages are added to the stages.ini file, build the system using the web tool "**Manual Configuration**" under the main tag **SYSTEM**. In this tool, the stages get assigned to positioners and the positioners get assigned to motion groups. Please refer to chapter 7.3 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 controller.

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 **STAGE** and then click on **"Add from database"**.

1. The following screen appears:



2. Select a family name from the list. Double click.
3. Select the part number corresponding to your hardware. Double click.

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 with a periodic position reset at 360°, means 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 7.3.

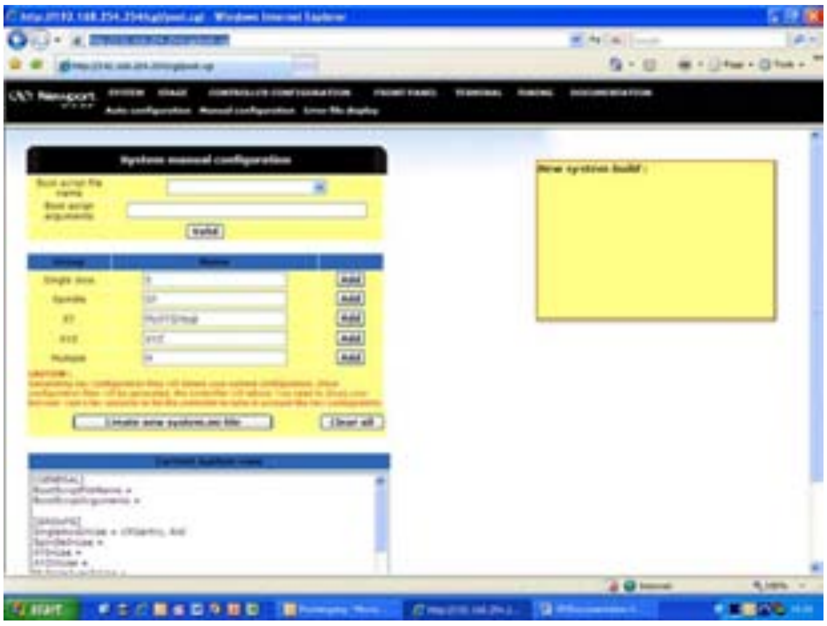
5. Once the stage name appears, you can modify it if you like (see comments above).

6. The box "Use ESP Compatibility for Hardware detection" is checked by default. If your stage has an ESP chip inside (see blue ESP compatible sticker on the stage) this box shall remain checked. Otherwise, with vacuum compatible stages or with old Newport stages, uncheck this box.

7. Click on "Add new stage" to add the stage.

   Once all stages have been added, you can review or modify these parameters from the screen **"Modify"** under the main tag **STAGE**. NOTE: From this screen you have access to all stage parameter. Only experienced users should modify these parameters. For the exact meaning of the different parameters, please refer to the document ConfigurationWizard.pdf, accessible from the main tag **DOCUMENTATION.**

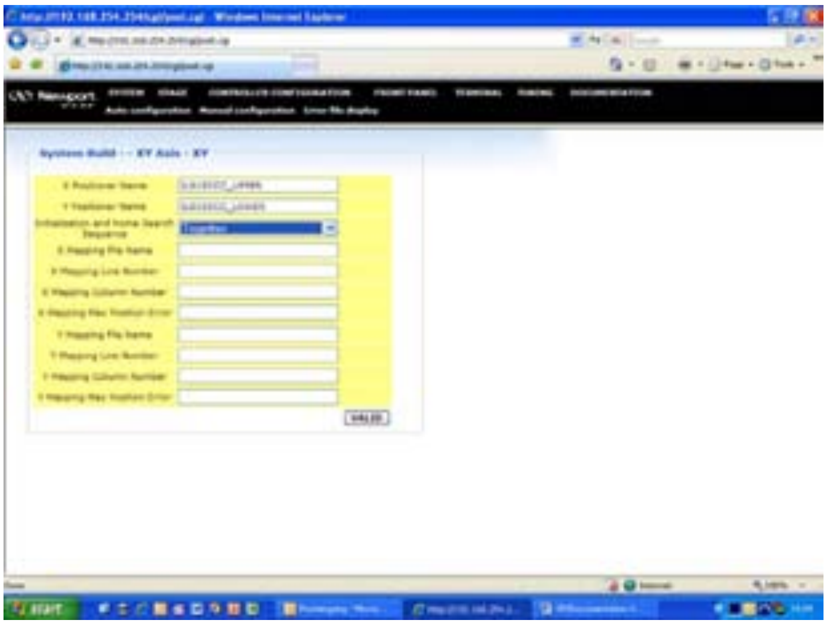
8. When done with all stages, click on **"Manual Configuration"** under **SYSTEM**. The following screen appears:



9. Enter a group name.

   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. Any group name can be given. In the example the name of the XY group is MyXYGroup.

10. Click on "ADD" to get to the next screen:



System Build : - XY Axis : XY

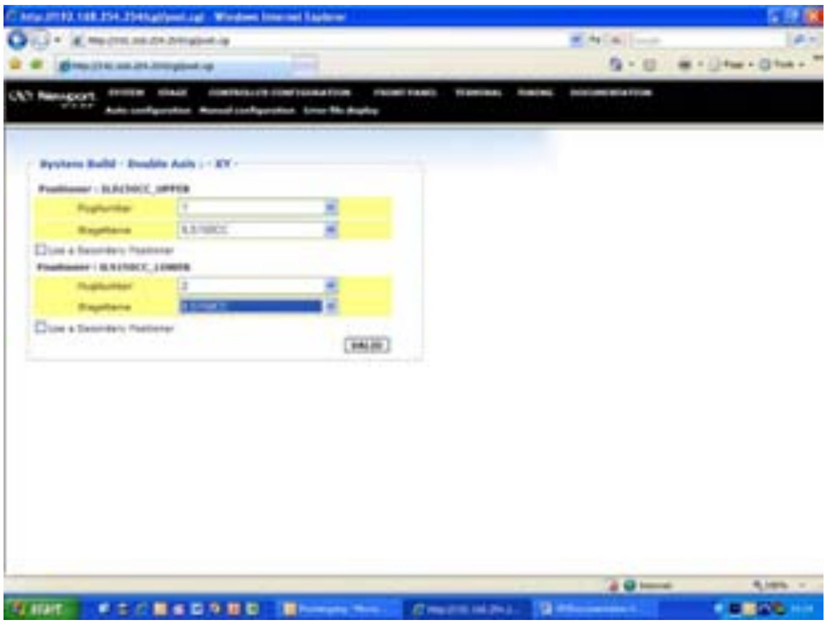
X Positioner Name	ILS150CC_UPPER
Y Positioner Name	ILS150CC_LOWER
Initialization and home Search Sequence	Together
X Mapping File Name	
X Mapping Line Number	
X Mapping Column Number	
X Mapping Max Position Error	
Y Mapping File Name	
Y Mapping Line Number	
Y Mapping Column Number	
Y Mapping Max Position Error	

11. Enter the positioner names.

Any positioner name can be used. In this example the X positioner name is ILS150CC_UPPER. 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 a first configuration, don't enter anything in these fields.

12. Click on "VALID" to get following screen:



System Build - Double Axis : - XY -

Positioner : ILS150CC_UPPER

PlugNumber	1
StageName	ILS150CC

☐ Use a Secondary Positioner

Positioner : ILS150CC_LOWER

PlugNumber	2
StageName	ILS150CC

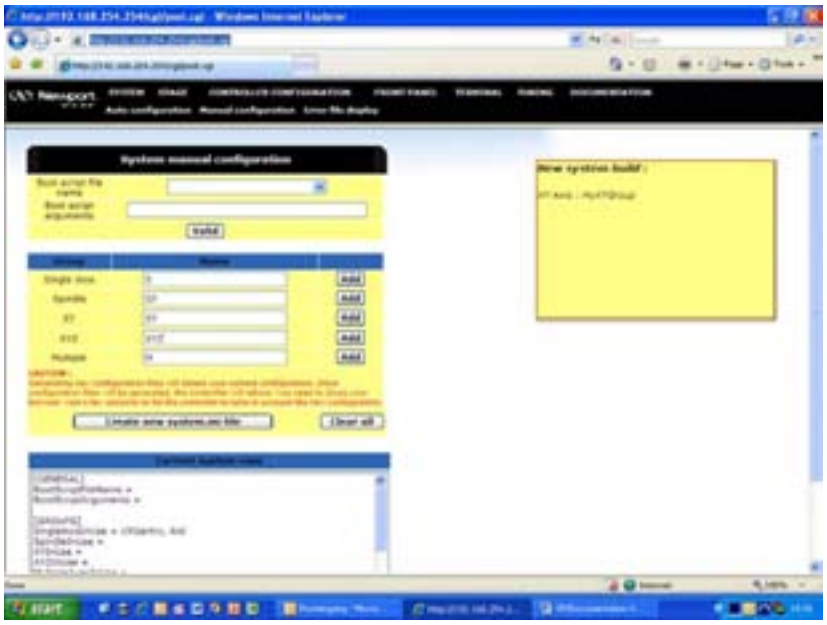
☐ Use a Secondary Positioner

13. Enter the appropriate PlugNumber. The plug number is the axis number where the stage is physically connected to the XPS controller. Plug number 1 is the first plug on the right and the number increases to the left, looking at the back of the controller.

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

15. Checking the box "Use a secondary Positioner" assigns a secondary positioner for a gantry configuration. For details about gantries, please refer to section 4.8. Don't check this box for a regular XY group or for a regular SingleAxis group.

16. Click on "VALID" to get back to the initial screen.

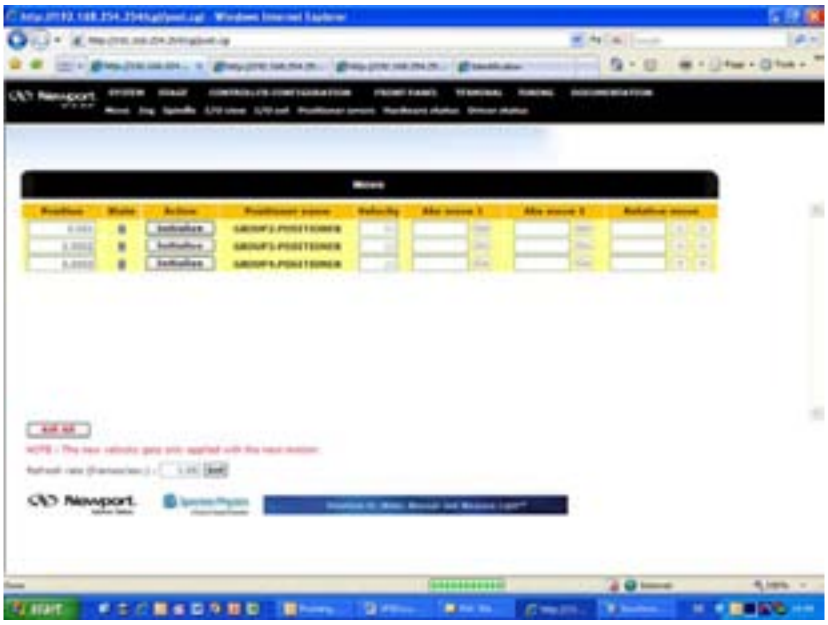


17. Continue the same way with the other motion groups.

18. When done, click on "Create new system.ini file" to complete the System configuration. The controller re-boots and the following message appears:

Click on "OK".

When the controller has finished booting (hear second beep after 1-2 minutes), press "F5" to reload the page, select **FRONT PANEL**, then select **"Move"**. The following screen appears (Group names will be different according to your definition):



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 proceed reading the rest of this manual.

3.8.3 Manual Configuration for stages not made by Newport

For configuring the XPS controller to stages or positioning devices not made by Newport, use the tool **"Add Custom Stage"** under the main tag **STAGE**. For detailed information about this tool, please refer to the document ConfigurationWizard.pdf provided under the main tag **DOCUMENTATION**.

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, see power switch above the power cord, at the back of the controller.

Software Tools

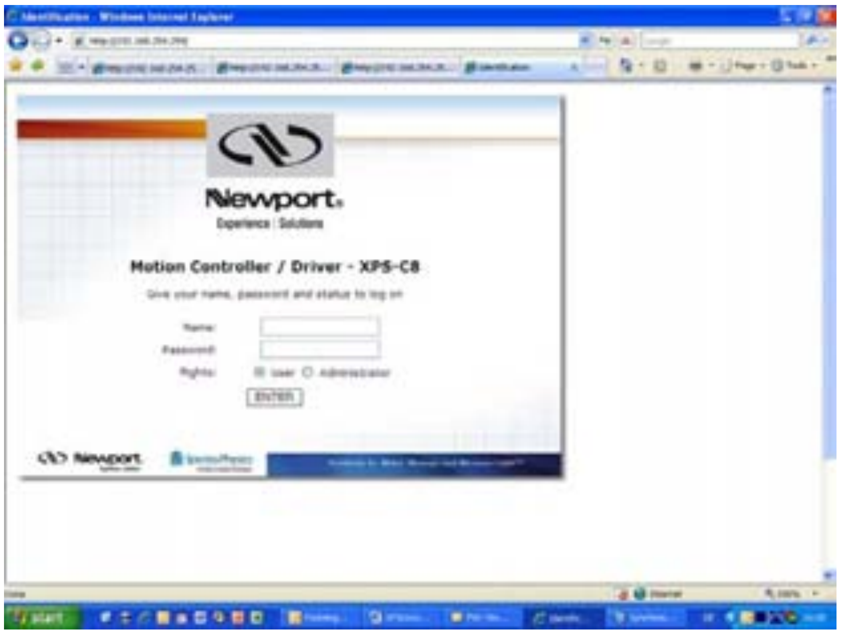
4.0 Software Tools

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. When logged-in with Administrator rights, you have an extended set of tools available.

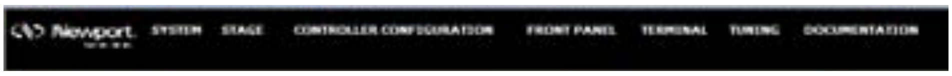
The predefined user has the log-in name **Anonymous**, Password **Anonymous**. The predefined Administrator has the log-in name **Administrator**, Password **Administrator**. Both, the Log-in name and the password are case sensitive.


Newport
Experience | Solutions
Motion Controller / Driver - XPS-C8
Give your name, password and status to log on
Name:
Password:
Rights: User Administrator
ENTER

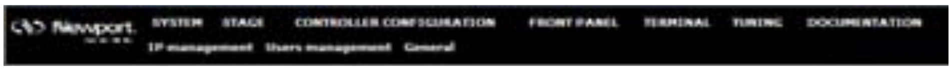
The main tag is displayed across the top of the XPS Motion Controller/Driver main program window, and lists each primary interface option. Each interface option has its own pull-down menu that allows the user to select various options by clicking the mouse's left button.

On the following pages we provide a brief description of all tools.

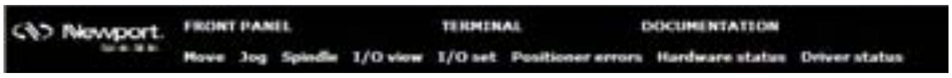
Administrator Menus



Sub-Menu for CONTROLLER CONFIGURATION



Restricted set of User Menus



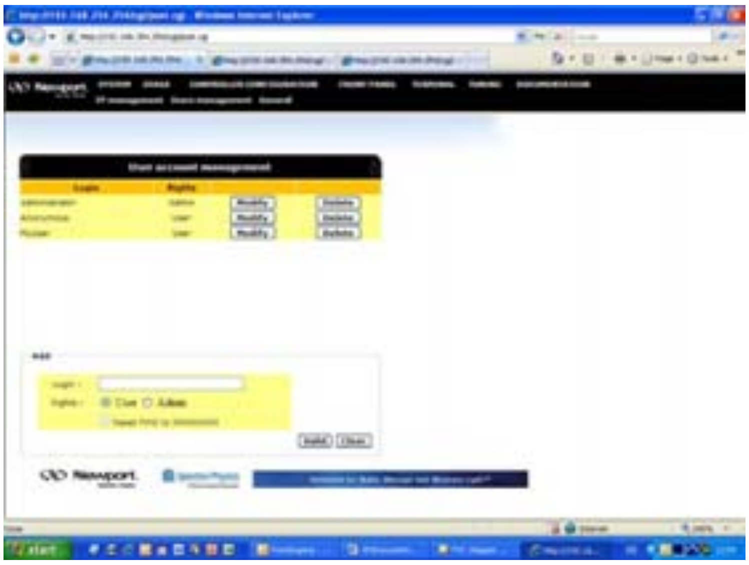
4.2 CONTROLLER CONFIGURATION – Users Management

This tool allows managing user accounts. There are two type of users defined: Administrators and Users. Administrators have configurations rights. Users have only restricted rights to use the system.

The following steps are needed to create a new user:

1. Enter a new user name in the “login” field.
2. Choose the access rights: “User” or “Admin”.
3. Check the box “Reset PWD to XXXXXXXX”:

   Your password is reset to XXXXXXXX.
4. Select the “VALID” button to add the new access account.

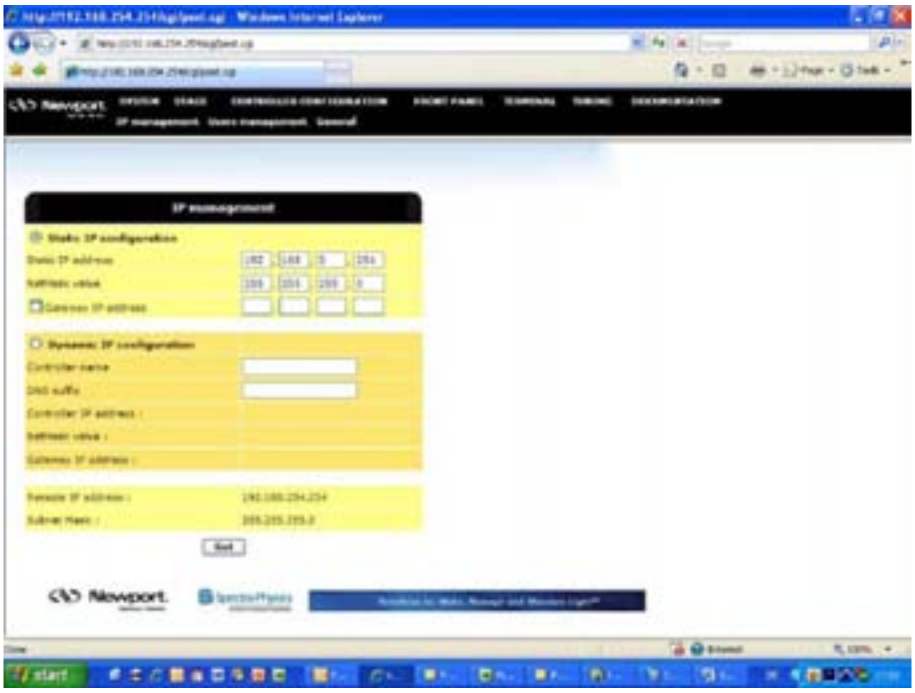


NOTE

The default password is XXXXXXXX and must be changed after the first log in.

4.3 CONTROLLER CONFIGURATION – IP Management

See chapter 3.5 for details.



IP management

Static IP configuration

Static IP address	192 . 168 . 5 . 254
Subnet value	255 . 255 . 255 . 0
Gateway IP address	

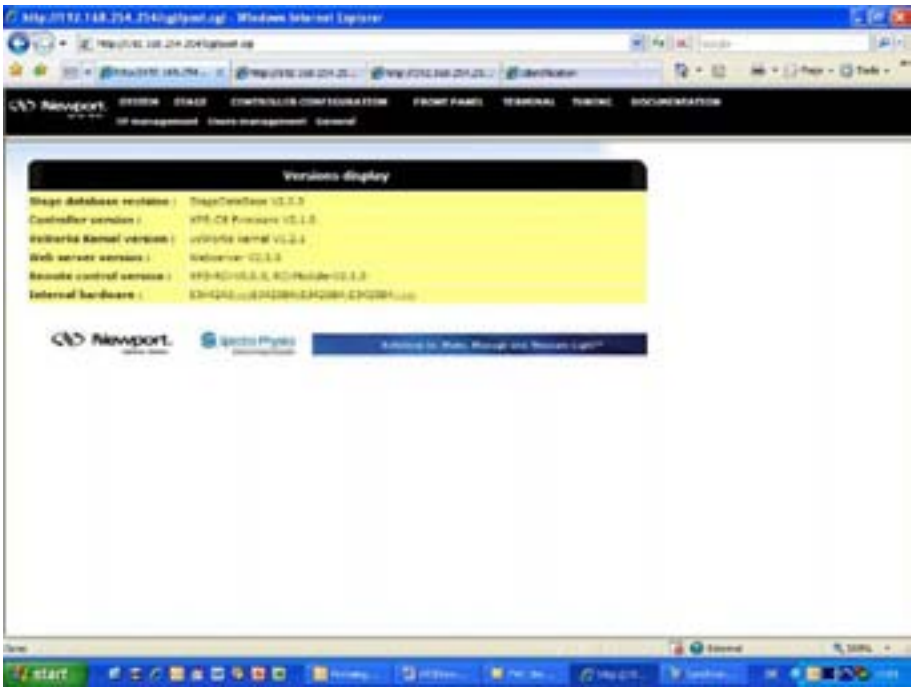
Dynamic IP configuration

Controller name	
DNS suffix	
Controller IP address :	
Subnet value :	
Gateway IP address :	

Remote IP address :	192.168.254.254
Subnet Mask :	255.255.255.0

4.4 CONTROLLER CONFIGURATION – General

This screen provides valuable information about the firmware and the hardware of the controller. It is an important screen for troubleshooting the controller.



Versions display

Stage database revision :	[illegible]
Controller version :	[illegible]
VxWorks Kernel version :	[illegible]
Web server version :	[illegible]
Remote control version :	[illegible]
Internal hardware :	[illegible]

4.5 SYSTEM – Error File Display

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

When no error has been detected during the system boot, this file is empty.

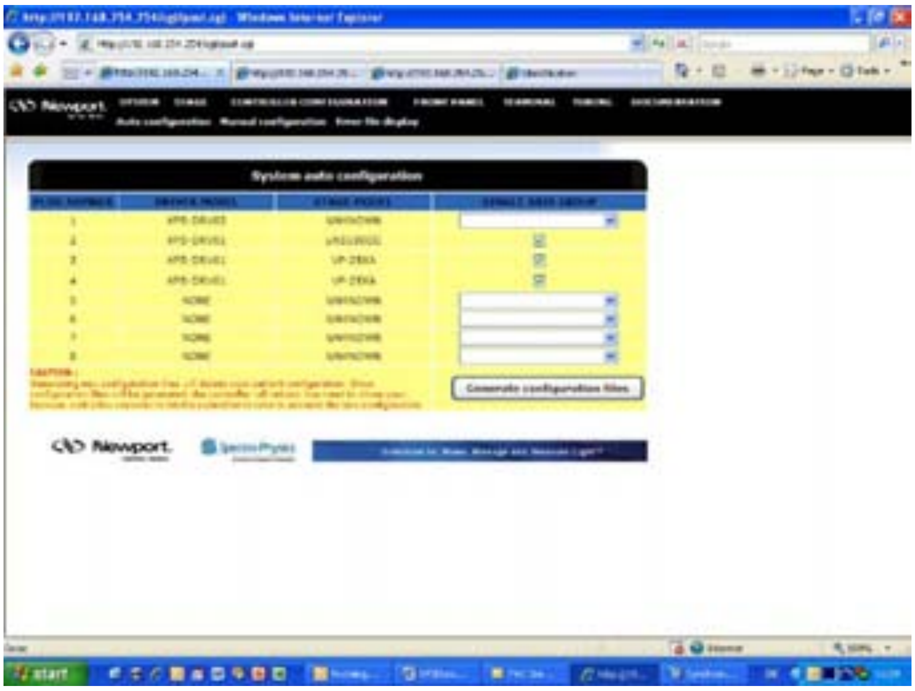
4.6 SYSTEM – Auto Configuration

With the help of this screen, a quick basic configuration of the XPS controller can be done. Check/un-check, those stage models that you want/don't want the XPS controller to configure to. When done, click "Generate Configuration Files". The XPS controller reboots. After re-booting, you are able to use the XPS controller in this basic configuration. For further information, refer also to chapter 3.8.1.

NOTE

"Generate Configuration Files" deletes your current system.ini configuration file. For troubleshooting a system, make sure that you have a copy of your system.ini file for recovery.

Under Driver Model and Stages model, all motor drivers and Newport ESP compatible stages seen by the XPS controller are listed. Hence, this screen provides also very valuable information for diagnosing or troubleshooting the system.



System auto configuration

PLUG NUMBER	DRIVER MODEL	STAGE MODEL	SINGLE AXIS GROUP
1	XPS-DRV03	UNKNOWN	
2	XPS-DRV01	URS100CC	✓
3	XPS-DRV01	UP-28XA	✓
4	XPS-DRV01	UP-28XA	✓
5	NONE	UNKNOWN	
6	NONE	UNKNOWN	
7	NONE	UNKNOWN	
8	NONE	UNKNOWN	

CAUTION : [illegible]

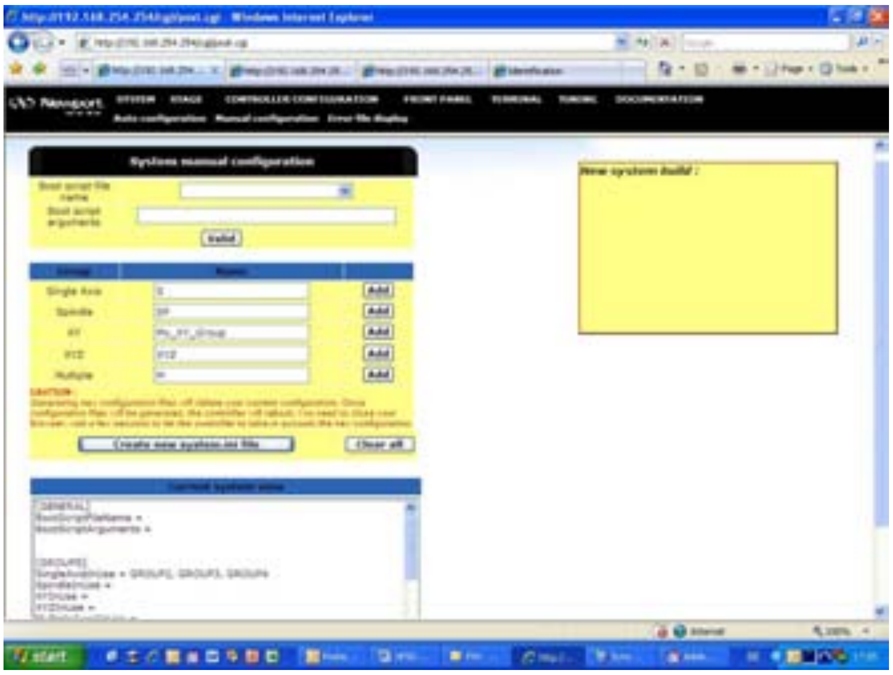
Generate configuration files

4.7 SYSTEM – Manual Configuration

This tool allows you to review the current system configuration and to define a new one. See also chapter 3.8.2 for further information.

To define a new system, you need to define all motion groups that should belong to that system. There is no possibility of appending a motion group to an existing configuration from this tool. To define a new motion group, do the following:

1. Enter the name of the new group (My_XY_Group in this case). Click on "ADD" to confirm the new group.



System manual configuration

Boot script file name

Boot script arguments

Valid

Group	Name	
Single Axis	S	Add
Spindle	SP	Add
XY	My_XY_Group	Add
XYZ	XYZ	Add
Multiple	M	Add

CAUTION: [illegible]

Create new system.ini file Clear all

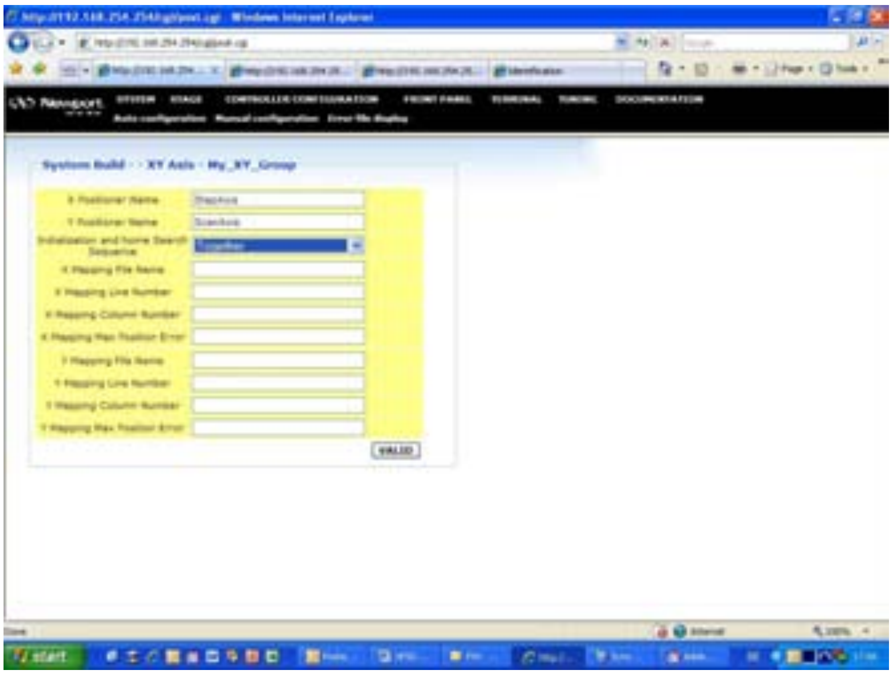
Current system view

```
[GENERAL]
BootScriptFileName =
BootScriptArguments =

[GROUPS]
SingleAxisInUse = GROUP2, GROUP3, GROUP4
SpindleInUse =
XYInUse =
XYZInUse =
```

New system build :

2. Enter the name for each positioner associated with the motion group (StepAxis and ScanAxis in this case). Define the home sequence ("Together" or "XThenY" or "YThenX"). For error compensation you need to define the name and structure of the correction data, otherwise leave these fields blank. For details about error

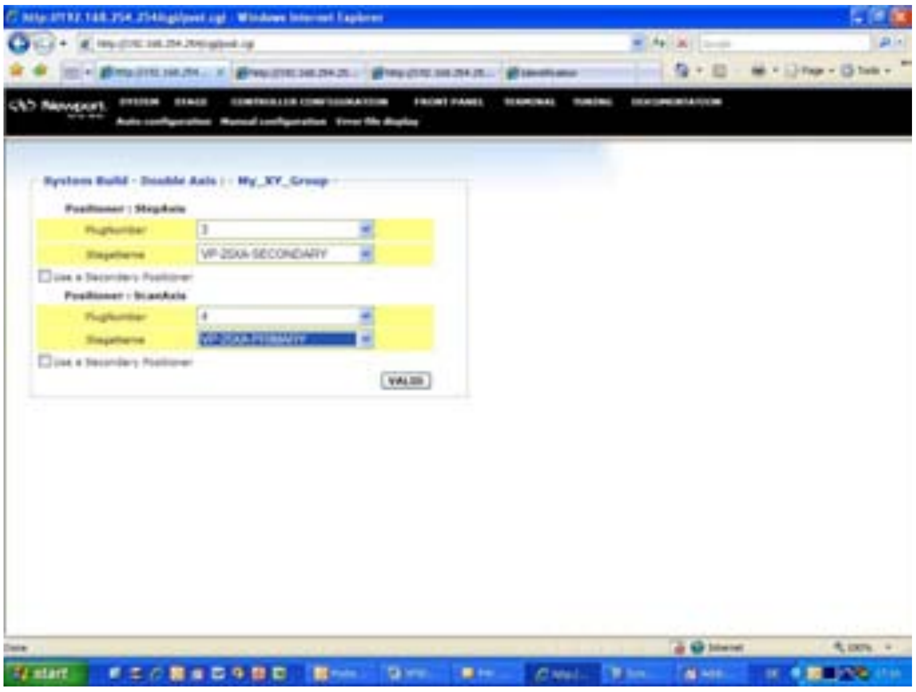
compensation, see chapter 10.0. When done, click on “VALID” to accept the configuration.



System Build - : XY Axis : My_XY_Group

X Positioner Name	StepAxis
Y Positioner Name	ScanAxis
Initialization and home Search Sequence	Together
X Mapping File Name	
X Mapping Line Number	
X Mapping Column Number	
X Mapping Max Position Error	
Y Mapping File Name	
Y Mapping Line Number	
Y Mapping Column Number	
Y Mapping Max Position Error	

3. Specify the plug number. The plug number is the number of the drive card (1 to 8) where the stage is physically connected to the XPS controller (see back of XPS controller). Select the name of the Stage from your stage data base (scroll down menu).
   Checking the box “Use a secondary Positioner” assigns a secondary positioner for a gantry configuration. A gantry is a motion device where two positioners, each of them having a motor, an encoder, limits, etc., are used for a motion in one direction. With most gantries, the two positioners are rigidly attached to each others. Hence, all motions, including motor initialization, homing, and emergency stops must be done in perfect synchronization. For details about Gantries and their configuration, please refer to section 4.8.
   When all positioners are configured, click on “VALID” to confirm the group configuration.



System Build - Double Axis : : My_XY_Group

Positioner : StepAxis

PlugNumber	3
StageName	VP-25XA-SECONDARY

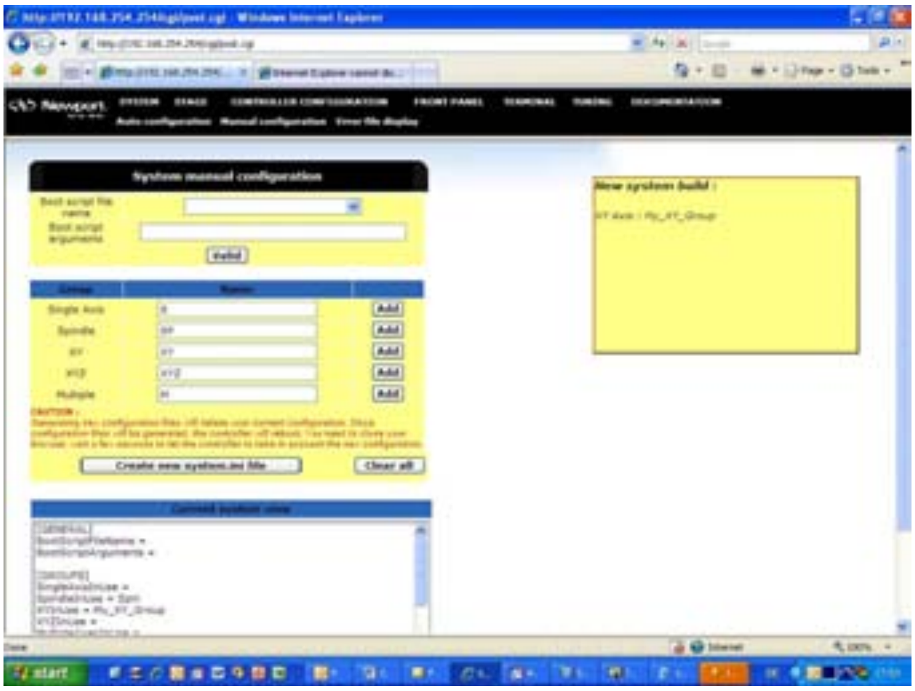
☐ Use a Secondary Positioner

Positioner : ScanAxis

PlugNumber	4
StageName	VP-25XA-PRIMARY

☐ Use a Secondary Positioner

4. When the configuration of each positioner is validated, the new group gets listed in the “New system build” window.



5. Continue the same way with all other motion group. When done, click on “Create new system.ini file” to apply the new configuration.

NOTE

“Create new system.ini file” deletes the current system.ini file. To keep a copy of the current system.ini file, get a copy from the “..admin\config” folder of the XPS controller.

The following screen appears:

Microsoft Internet Explorer
Your modifications have been processed, and the controller is now rebooting !
OK

Click on “OK”.

6. When the controller has finished booting (hear second beep after 1-2 minutes), select **SYSTEM** tag, then **“Error File Display”**. When there is no entry in the error file, your system is configured correctly and ready to use. When not, this file provides some valuable information for trouble shooting, see also chapter 4.5.

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
```

4.8 SYSTEM – Manual Configuration – Gantries (Secondary Positioners)

This section refers to experienced users of the XPS controller and addresses the configuration of a gantry via a secondary positioner.

A gantry is a motion device where two positioners, each of them having a motor, an encoder, limits, etc., are used for a motion in one direction. With most gantries, the two positioners are rigidly attached to each others, see example below. Hence, all motions, including motor initialization, homing, and emergency stops must be done in perfect synchronization.

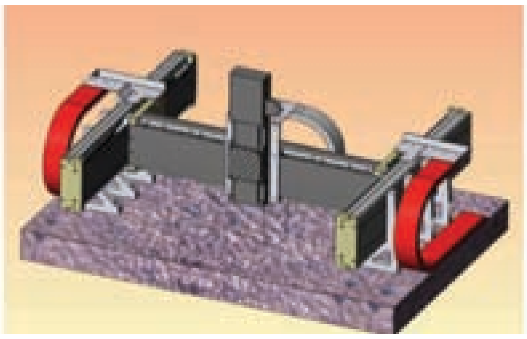
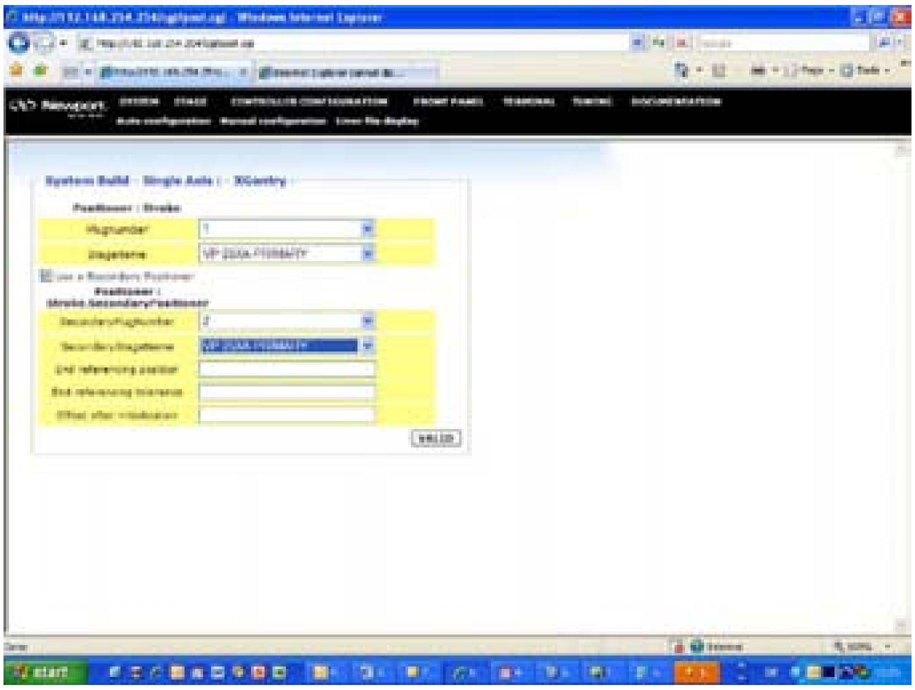


Figure 16: Example of a gantry.

The XPS controller allows configuring single axis gantries (Xx configuration) and XY gantries. For XY gantries, it is possible to define XxY, XYy and XxYy configurations. Here, X and Y refer to the primary positioner and x and y to an assigned secondary positioner.

To define a gantry, check the box “Use a secondary positioner” during the definition of a Single Axis group or XY group. See chapter 4.7 for further instructions how to define a new motion group. When done, the following screen appears (example Single Axis group):


System Build - Single Axis - XGantry

Positioner : Stroke

PlugNumber	1
StageName	[illegible]

Use a Secondary Positioner

Positioner : Stroke.SecondaryPositioner

SecondaryPlugNumber	2
SecondaryStageName	[illegible]
End referencing position	
End referencing tolerance	
Offset after initialization	

Define the plug number for the secondary positioner and the name from the stage data base. The secondary positioner must have at least common values with the primary positioner for the following parameters:

MaximumVelocity
MaximumAcceleration
HomeSearchMaximumVelocity
HomeSearchMaximumAcceleration
MinimumTargetPosition
MaximumTargetPositioner

The parameters "End referencing position" and "End referencing tolerance" refer to the homing process of the gantry, see chapter 4.8.1 for details.

The parameter "Offset after initialization" is relevant only for gantries with linear motors. See chapter 4.8.2 for details. For all other gantries, enter 0 for this parameter.

Furthermore, for certain XY gantries, it is also possible to apply a variable force ratio for the two X positioner. This variable force ratio accounts for the different forces required by the primary and the secondary X-axes positioners depending on the position of the Y axis to ensure a torque-free motion. For details, see chapter 4.8.3.

NOTE

When done with your gantry configuration, the secondary positioner is almost invisible for the application. All functions are sent directly to the motion group or to the (primary) positioner of that group. However, it is possible to get information from the secondary positioner by data gathering by appending "SecondaryPositioner" to the positioner name. Example:

MySingleGantry.S1.SecondaryPositioner.FollowingError

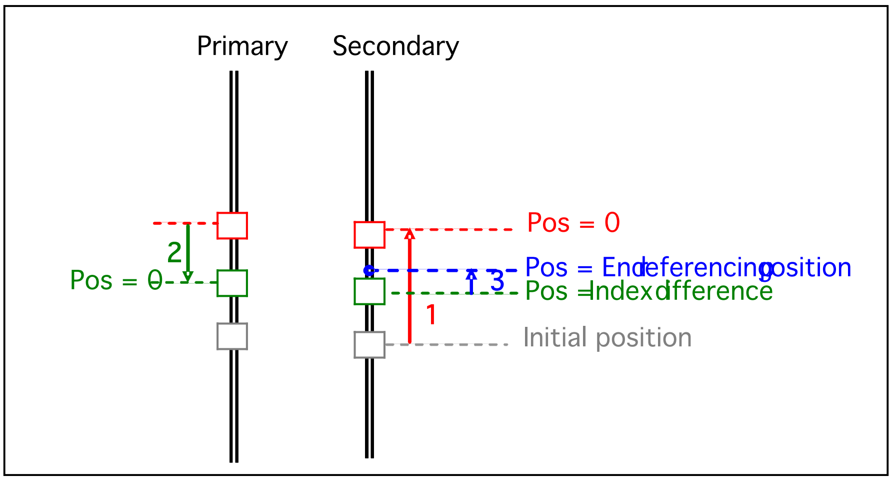
For further details about data gathering, see chapter 12.0.

4.8.1 Home search of gantries

During the home search of a gantry, first the secondary positioner is homed and the primary positioner follows the motion. Then, the primary positioner is homed and the secondary positioner follows the motion. At the end, the primary positioner is at his home position, but the secondary positioner will be off his home position due to the tolerances in the assembly of the gantry. The parameter "End referencing position" defines the "ideal" position of the secondary positioner when the primary positioner is at his home position. The parameter "End referencing tolerance" defines the maximum allowed distance between the secondary positioner's position, when the primary positioner is at his home position, and the "end referencing position".

When the actual distance is greater than the value for the "End referencing tolerance", the homing is aborted. When the actual distance is less than the value for the "End referencing tolerance", then the secondary positioner moves to the "End referencing position" whiles the primary positioner stays at his home position. Hence, this parameter allows correcting the angle of the gantry.

The below sketch illustrates this process:



1) Search home of the Secondary positioner. The primary positioner follows
2) Search home of the Primary positioner. The secondary positioner follows.
3) If the distance of the secondary positioner's position to the "End referencing position" is greater than the value for the "End referencing tolerance", the homing is aborted. If not, the Secondary positioner moves to the "End referencing position" while the primary positioner stays at his home position.

Index difference refers to the difference of the secondary positioner's position when the primary positioner is at his home position to the home position of the secondary positioner. The value for the Index difference can be queried by the function PositionersEncoderIndexDifferenceGet().

When no other metrology tools are available, the following method can be used to determine a value for the "End referencing position" of an assembled gantry:

Set the value for "End referencing position" and "End referencing tolerance" to 0. Complete the configuration of your system. After reboot, initialize and home the gantry group. With high probability the homing will fail with error -85 due to the zero value for the "End referencing tolerance". Query the index difference with the function PositionersEncoderIndexDifferenceGet(). Repeat the initialization, homing and querying of the index difference several times and build the average and the standard deviation from all values. Now, configure a new system with the same gantry. For "End referencing position" apply the average value of the index difference. For "End referencing tolerance" apply a value that is approximately equal to 6 times of the standard deviation of the index difference. Complete your configuration and reboot your system. Initialize and home your gantry group several times. Your system should now work properly.

4.8.2 Gantries with linear motors

The parameter "Offset after initialization" defines the offset of the magnetic tracks of the linear motors between the primary and the secondary positioner. This parameter is important to provide optimum performance of a gantry with linear motors. It ensures a correct sinusoidal commutation of the two motor signals. An accurate measurement of the offset can be done only with dedicated metrology tools.

For gantries NOT driven in acceleration mode, e.g. gantries with NO linear motors, this value can be put to 0.

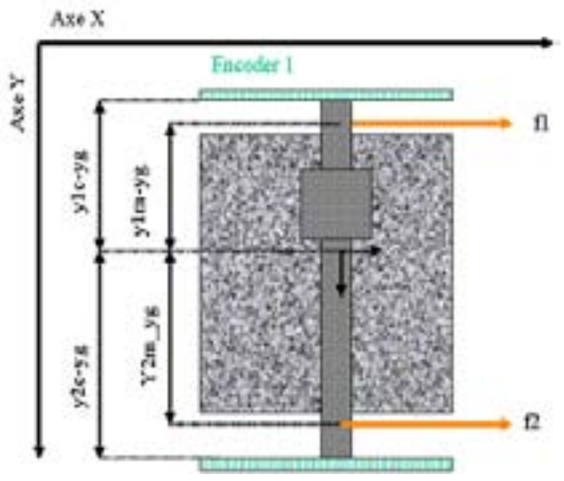
Also: With all stages driven in acceleration mode and that are configured for gantries, it is recommended to "force the initialization position" using the LMI mode (Large Move Initialization). To do so, append **LMI** to the line MotorDriverInterface=AnalogSinXAcceleration**LMI** (X = 60, 90 or 120) and add a line InitializationCycleDuration=5 at the end of the section driver command interface parameters of the stages.ini. Example:

```
;--- Driver command interface parameters
MotorDriverInterface=AnalogSin120AccelerationLMI
ScalingAcceleration=30641;--- units / s²
AccelerationLimit=27856;--- units / s²
MagneticTrackPeriod=24;--- units
InitializationAccelerationLevel=20;--- percent
InitializationCycleDuration=5;--- seconds
```

With the LMI setting, during initialization, the motor gets energized and the stage moves to the closest stable magnetic position. The result is a quick motion of the stage by maximum plus or minus half of the length of the magnetic track. This behavior might be undesired, but provides a more failure proof method for initialization than the Newport default process, that applies only very small oscillations to the stage during the initialization.

4.8.3 Gantries with linear motors and variable force ratio

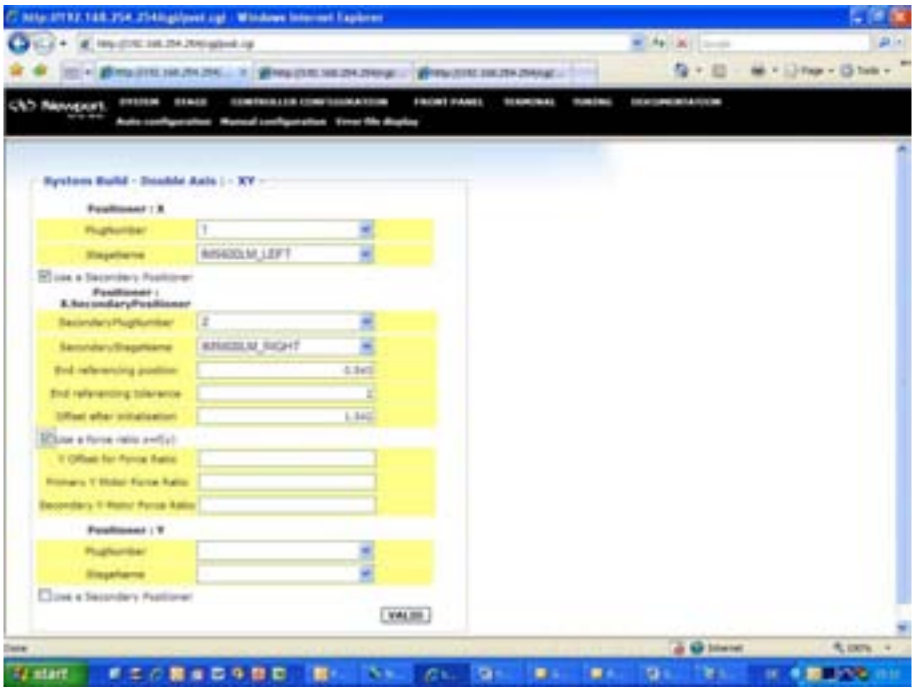
For XY gantries, where the two X-axes are driven by linear motors (means driven in acceleration mode), it is also possible to apply a variable force ratio for the two X-axes positioners. This variable force ratio accounts for the different forces required by the primary and the secondary X-axes positioner depending on the position of the Y axis. When correctly set it ensures a torque-free acceleration and deceleration on the x-axis, see picture below for illustration.



For applying a variable load ratio for an XY gantry, check the box "Use a force ratio", during the group definition. See example below. There are three parameters to input:

Y Offset for force ratio
Primary Y Motor Force Ratio
Secondary Y Motor Force Ratio

A correct definition for these three parameters is not simple. In case of interest in this function, please call Newport.



System Build - Double Axis : - XY -

Positioner : X

PlugNumber	1
StageName	IMS600LM_LEFT

[✓] Use a Secondary Positioner

Positioner : X.SecondaryPositioner

SecondaryPlugNumber	2
SecondaryStageName	IMS600LM_RIGHT
End referencing position	[illegible]
End referencing tolerance	2
Offset after initialization	[illegible]

[✓] Use a force ratio x=f(y)

Y Offset for Force Ratio	
Primary Y Motor Force Ratio	
Secondary Y Motor Force Ratio	

Positioner : Y

PlugNumber	
StageName	

[ ] Use a Secondary Positioner

This is an example of a system.ini file with one XY gantry:

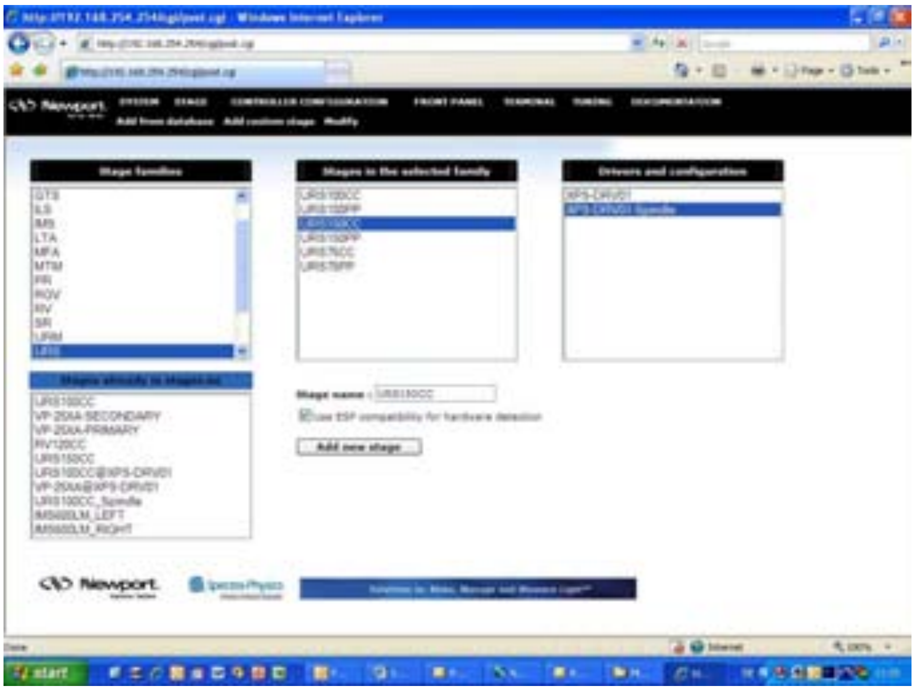
```
[GROUPS]
SingleAxisInUse =
SpindleInUse =
XYInUse = MyXYGantry
XYZInUse =
MultipleAxesInUse =

[MyXYGantry]
PositionerInUse = X, Y
InitializationAndHomeSearchSequence = YThenX
XMappingFileName =
YMappingFileName =

;--- Gantry Force Ratio parameters
YOffsetForForceRatio = 0
PrimaryYForceRatio = 0
SecondaryYForceRatio = 0

[MyXYGantry.X]
PlugNumber = 1
StageName = IMS600LM

;---- Secondary positioner (X2)
SecondaryPlugNumber = 4
SecondaryStageName = IMS600LM
SecondaryPositionerGantryEndReferencingPosition = 10.2243
SecondaryPositionerGantryEndReferencingTolerance = 0.1
SecondaryPositionerGantryOffsetAfterInitialization = 7.47

[MyXYGantry.Y]
PlugNumber = 3
StageName = IMS400LM
```

4.9 STAGE – Add from Data Base

With the help of this tool, a stage from the Newport stage data base can be added to the personal stage data base, called stages.ini. In the lower left corner you can review the name of the stages that are already in this stage data base. To add a new stage, do the following:

1. Select a family name from the list. Double click.
2. Select the part number corresponding to your hardware. Double click.
3. 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 with a periodic position reset at 360°, means 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 7.3.
4. Once the stage name appears, you can modify it, if you like. The default name is the Newport part number, but in some cases it makes sense using a different name. That ways, for instance, it is possible adding the same set of parameters several times in the stage data base under different stage names. Later, you can modifying certain parameters, like travel ranges or PID settings, to optimize the stage for different applications.
5. The box "Use ESP Compatibility for Hardware detection" is checked by default. If your stage has an ESP chip inside (see blue ESP compatible sticker on the stage) this box shall remain checked. Otherwise, with vacuum compatible stages or with old Newport stages, uncheck this box.
6. Click on "Add new stage" to add the stage.



Stage families

GTS
ILS
IMS
LTA
MFA
MTM
PR
RGV
RV
SR
URM
URS

Stages in the selected family

URS100CC
URS100PP
URS150CC
URS150PP
URS75CC
URS75PP

Drivers and configuration

XPS-DRV01
XPS-DRV01 Spindle

Stages already in stages.ini

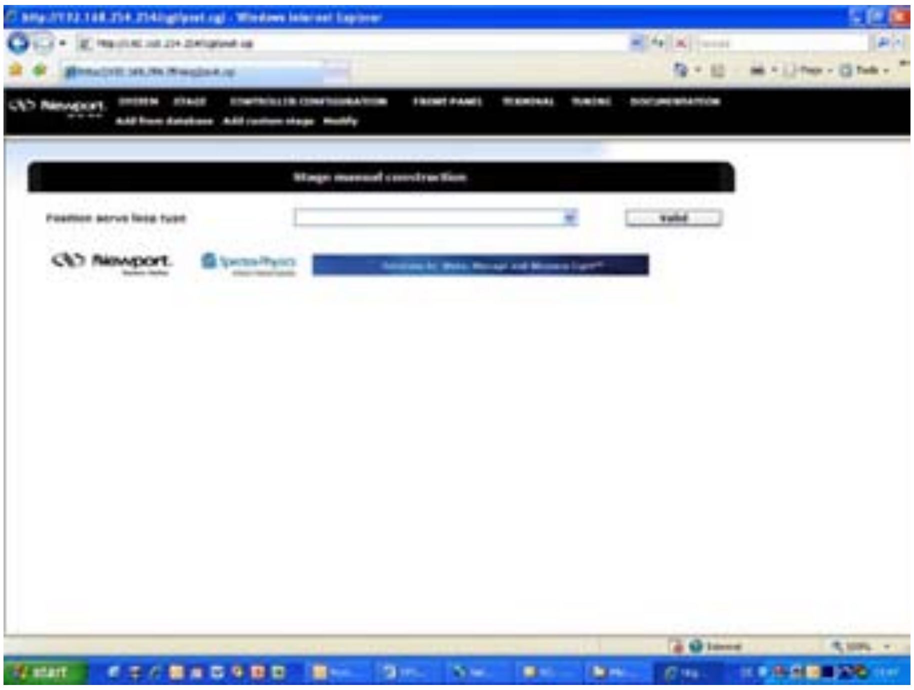
URS100CC
VP-25XA-SECONDARY
VP-25XA-PRIMARY
RV120CC
URS150CC
URS100CC@XPS-DRV01
VP-25XA@XPS-DRV01
URS100CC_Spindle
IMS600LM_LEFT
IMS600LM_RIGHT

Stage name : URS150CC

[✓] Use ESP compatibility for hardware detection

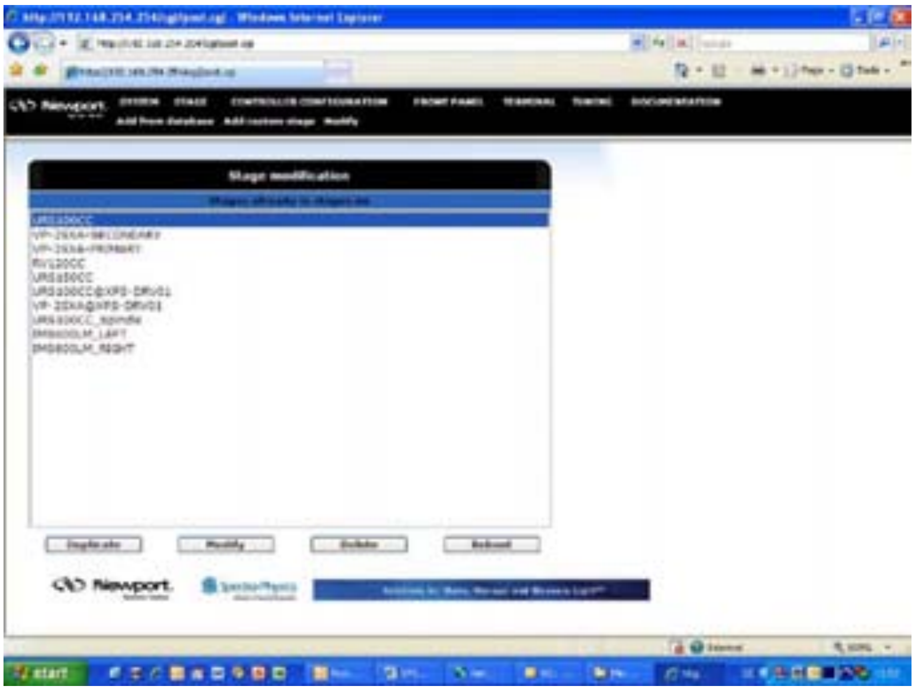
4.10 STAGE – Add Custom Stage

This tool supports the configuration of the XPS controller to a stage or to a positioning device that is not made by Newport. For detailed information about this tool, please refer to the document ConfigurationWizard.pdf provided under the main tag **DOCUMENTATION**.



4.11 STAGE – Modify

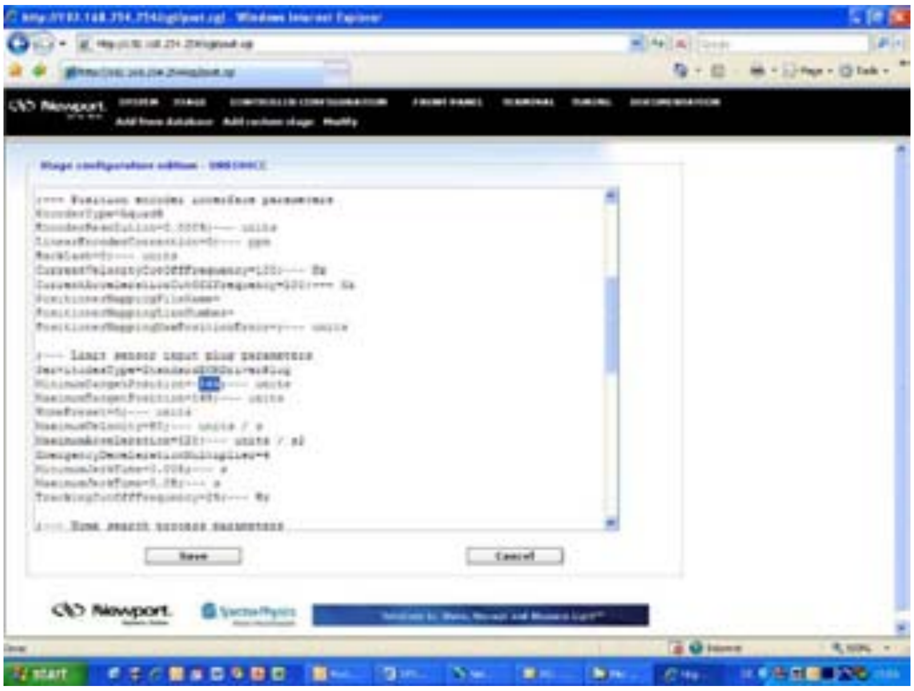
This tool allows you to review and modify all parameters of all stages included in the stages.ini. Only experienced users should modify these parameters. For the exact meaning of the different parameters, please refer to the document ConfigurationWizard.pdf, accessible from the main tag **DOCUMENTATION.**



URS100CC
VP-25XA-SECONDARY
VP-25XA-PRIMARY
RV120CC
URS150CC
URS100CC@XPS-DRV01
VP-25XA@XPS-DRV01
URS100CC_Spindle
IMS600LM_LEFT
IMS600LM_RIGHT

To modify parameters of a stage, do the following:

1. Select a stage from the list. Click on Modify.
2. Scroll down to the section that contains the parameters that you want to modify. Parameters, that require quite common changes, are the minimum and the maximum target position 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.



```
[illegible]
```

3. When done, click "Save" to apply the new values, else click Cancel.
4. To take the new values into account, you need to reboot the controller.

The same tool allows also 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.

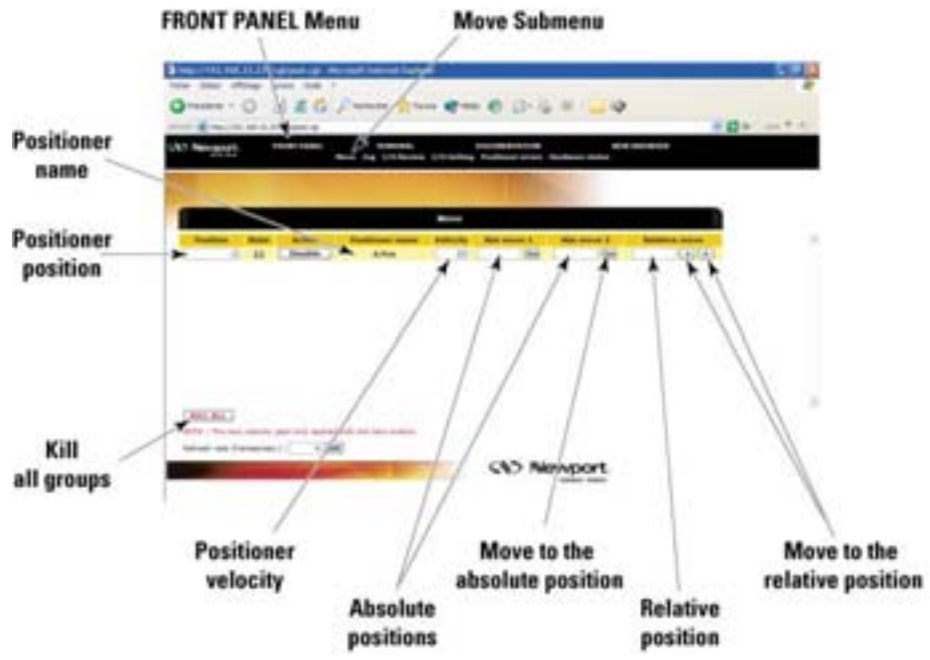
4.12 FRONT PANEL – Move

The move page provides access to basic group functions like initialize, home, or motor disable, and allows executing relative and absolute moves.

The move page provides also 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

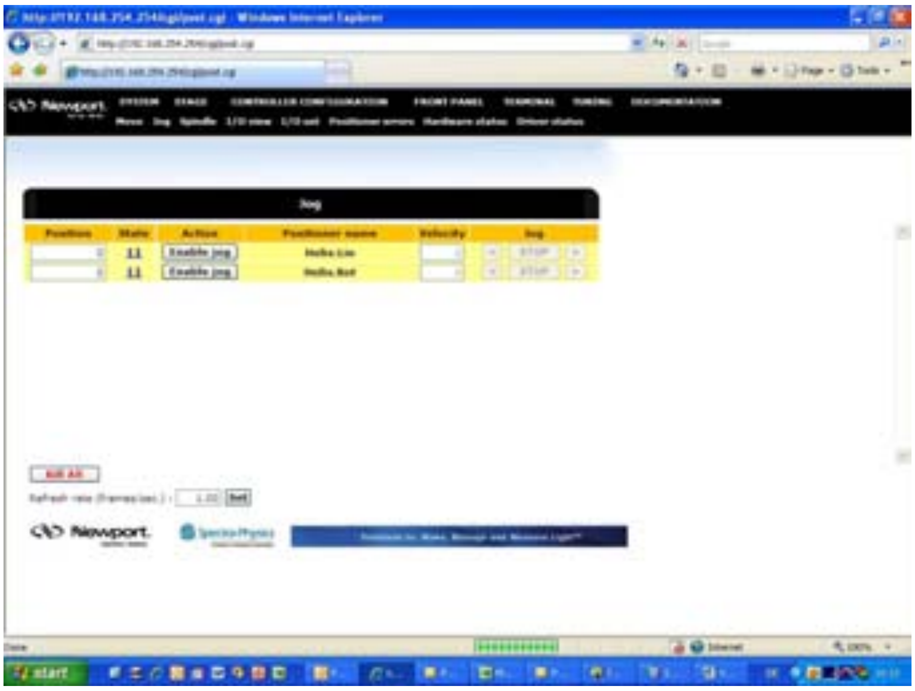
A spindle group can do relative moves and absolute moves. So it can be used in the MOVE page.



4.13 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.

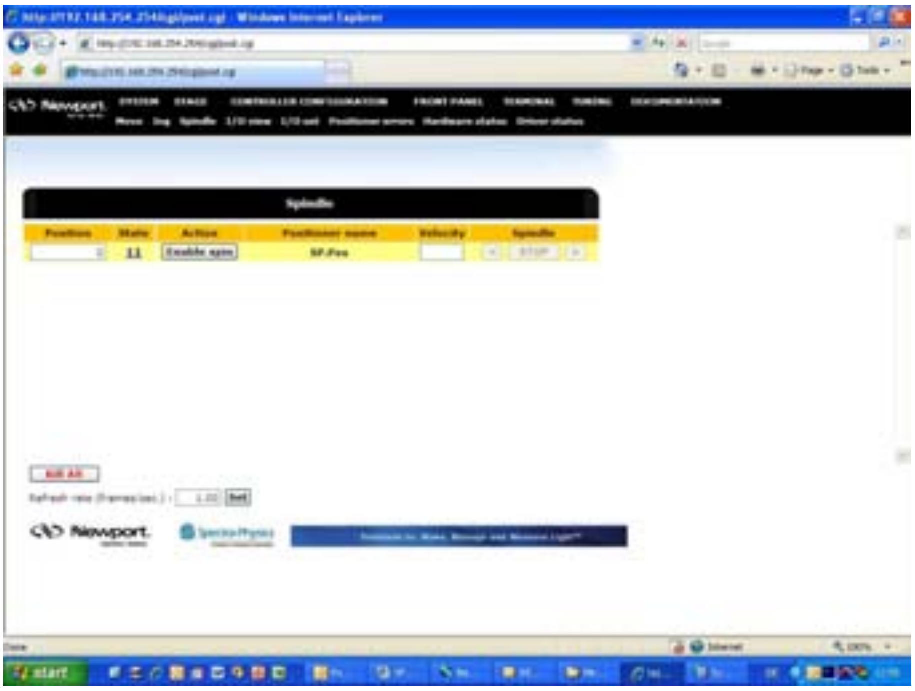


Jog

Position	State	Action	Positioner name	Velocity	Jog
	11	Enable jog	Helix.Lin		STOP
	11	Enable jog	Helix.Rot		STOP

4.14 FRONT PANEL – Spindle

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

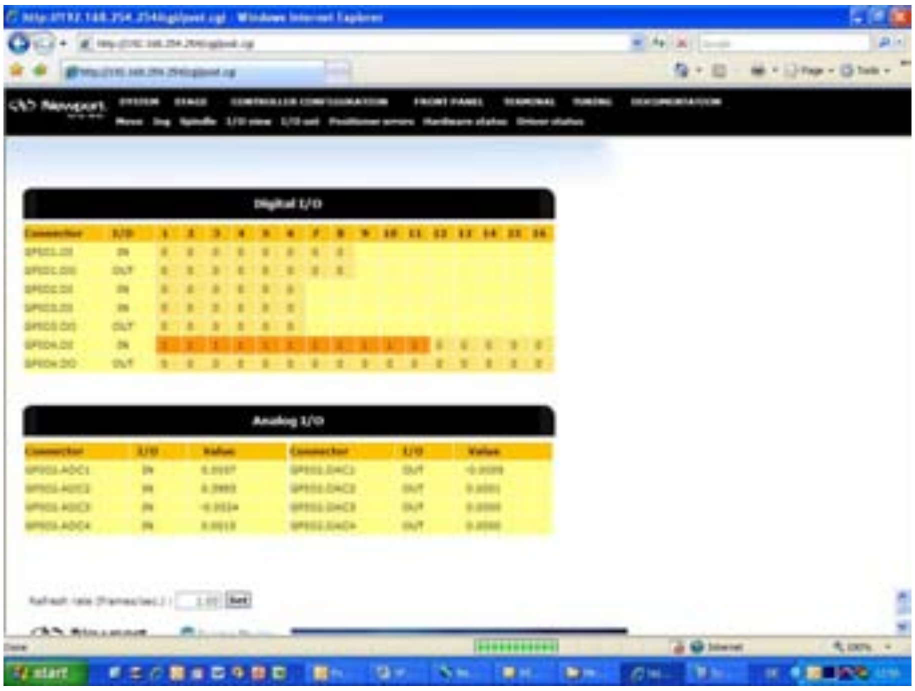


Spindle

Position	State	Action	Positioner name	Velocity	Spindle
	11	Enable spin	SP.Pos		STOP

4.15 FRONT PANEL – I/O View

The I/O Review page provides review of all analog and all digital I/O’s of the controller. To set the outputs, use the page I/O Set.



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	0	0	0	0	0	0	0	0								
GPIO1.DO	OUT	0	0	0	0	0	0	0	0								
GPIO2.DI	IN	0	0	0	0	0	0										
GPIO3.DI	IN	0	0	0	0	0	0										
GPIO3.DO	OUT	0	0	0	0	0	0										
GPIO4.DI	IN	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
GPIO4.DO	OUT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

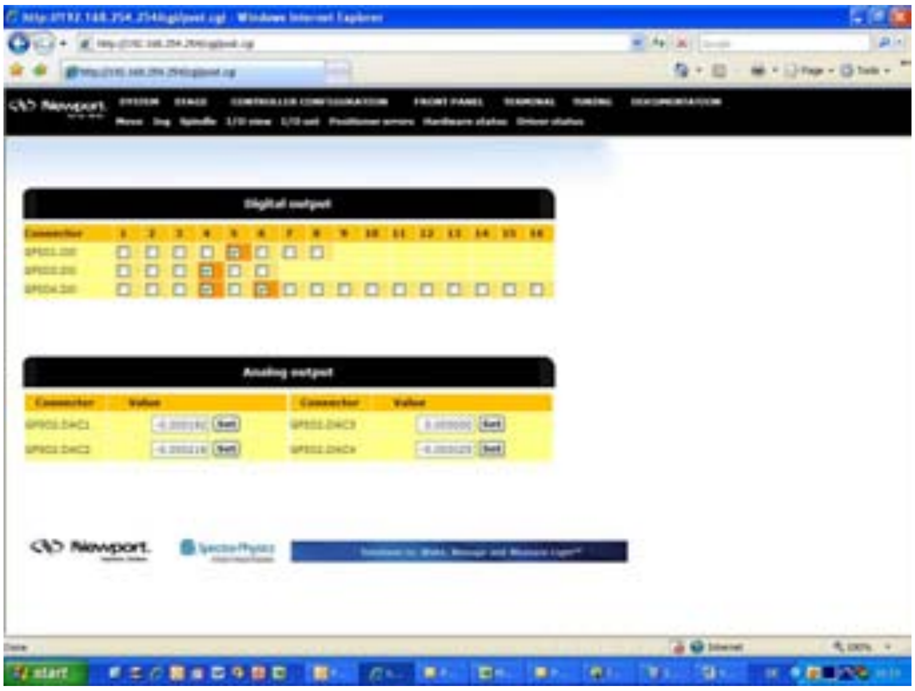
Analog I/O

Connector	I/O	Value	Connector	I/O	Value
GPIO2.ADC1	IN	0.0037	GPIO2.DAC1	OUT	-0.0009
GPIO2.ADC2	IN	0.3993	GPIO2.DAC2	OUT	0.0001
GPIO2.ADC3	IN	-0.0024	GPIO2.DAC3	OUT	0.0000
GPIO2.ADC4	IN	0.0019	GPIO2.DAC4	OUT	0.0000

Refresh rate (frames/sec.) : 1.00 Set

4.16 FRONT PANEL – I/O Set

The I/O set page provides access to set the analog and digital outputs of the controller.



Digital output

Connector	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
GPIO1.DO	☐	☐	☐	☐	☑	☐	☐	☐								
GPIO3.DO	☐	☐	☐	☑	☐	☐										
GPIO4.DO	☐	☐	☐	☑	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Analog output

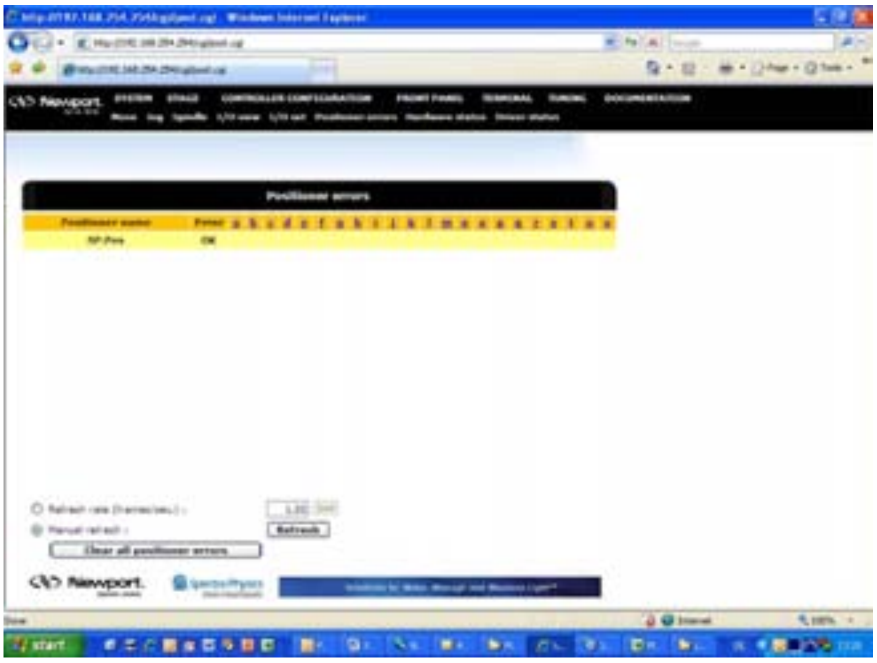
Connector	Value	Connector	Value
GPIO2.DAC1	-0.000192 Set	GPIO2.DAC3	0.000000 Set
GPIO2.DAC2	-0.000216 Set	GPIO2.DAC4	-0.000025 Set

4.17 FRONT PANEL – Positioner Errors

The positioner errors page is an important page for trouble-shooting. When encountering any problems during the use of the system, you find here valuable information about the errors related to the positioners.

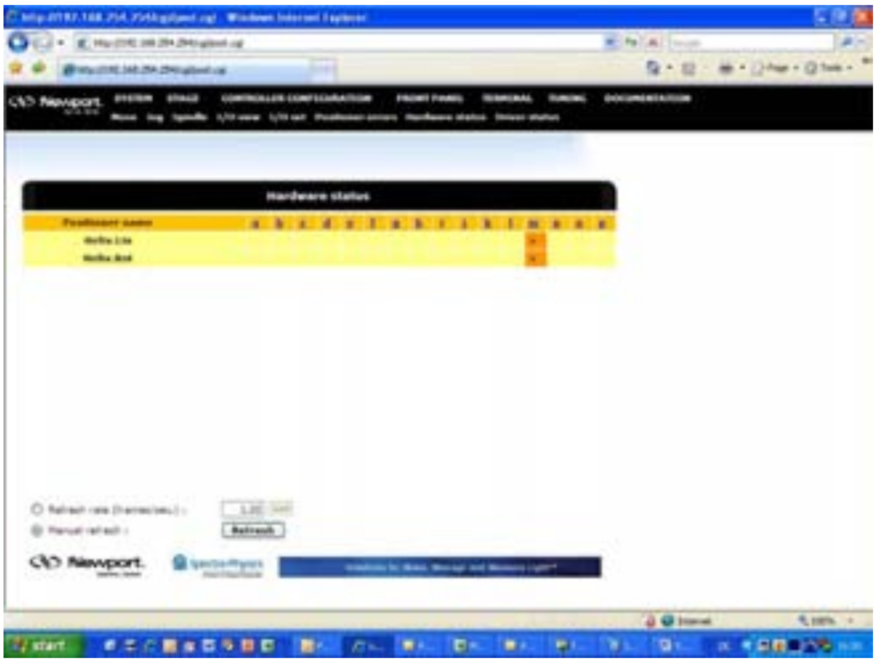
Note that all positioner errors encountered since the last “Clear all positioner errors” are displayed, even if some of the errors may not be present anymore. The button “Refresh” refreshes the error page. This means, new errors are displayed, but old errors that do not exist anymore, are not removed.

To clear the errors, use the button “Clear all positioner errors”.



4.18 FRONT PANEL – Hardware Status

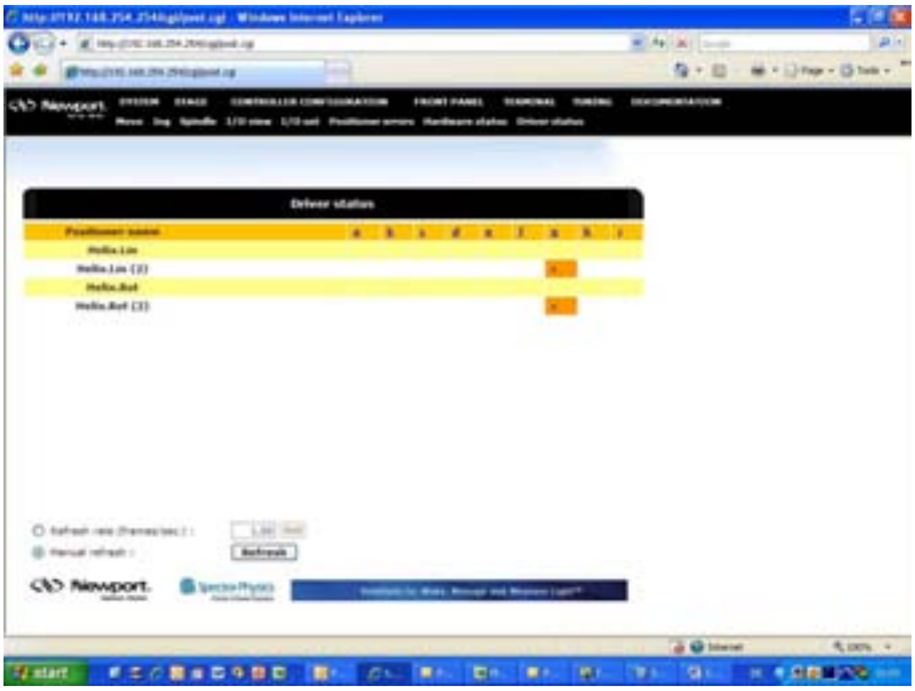
The Hardware status page is another important page for trouble-shooting. You find more valuable information related to your hardware here. Not all information is related to an error.



4.19 FRONT PANEL – Driver Status

The Driver status page is another important page for trouble-shooting. Here you find valuable information related to the status of the driver. Not all information is related to an error.

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

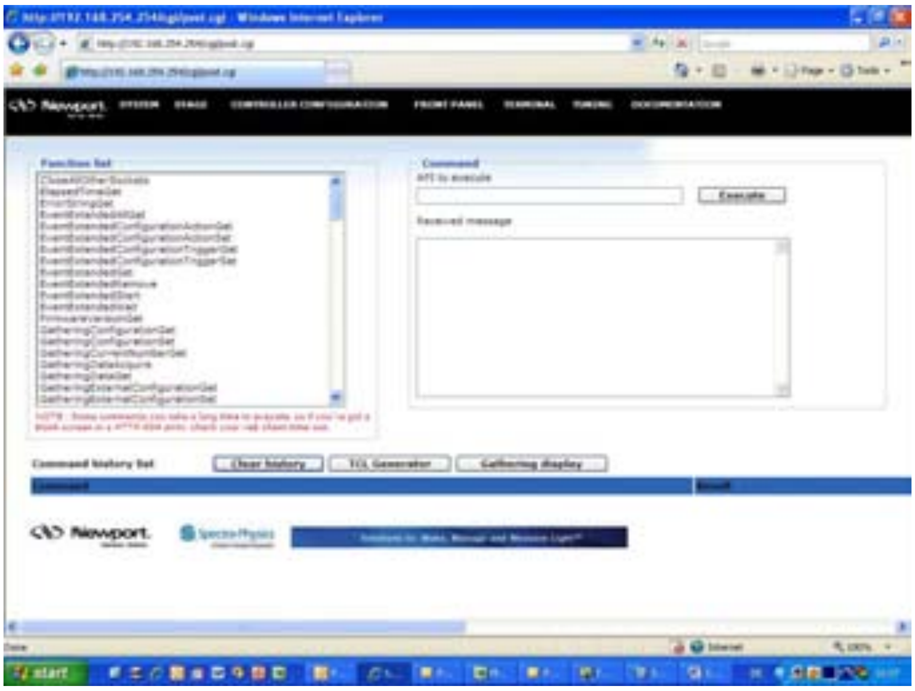


Driver status

Positioner name	a	b	c	d	e	f	g	h	i
Helix.Lin									
Helix.Lin (2)							x		
Helix.Rot									
Helix.Rot (2)							x		

4.20 TERMINAL

The terminal tool allows executing all functions of the XPS controller. It provides also a convenient method for generating executable TCL scripts. For more details about TCL scripts, see chapter 16.1.

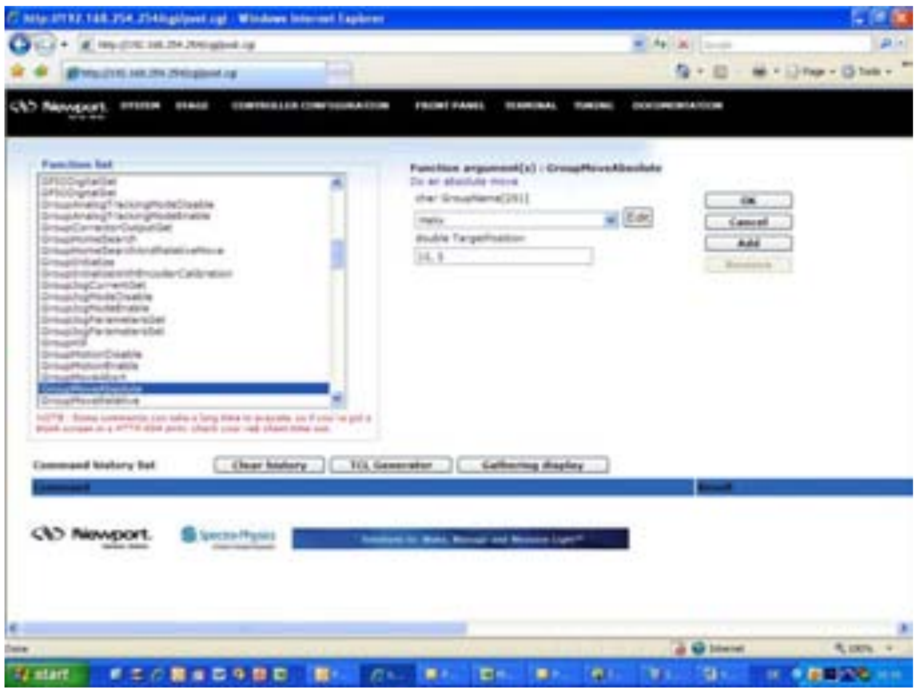


CloseAllOtherSockets
ElapsedTimeGet
ErrorStringGet
EventExtendedAllGet
EventExtendedConfigurationActionGet
EventExtendedConfigurationActionSet
EventExtendedConfigurationTriggerGet
EventExtendedConfigurationTriggerSet
EventExtendedGet
EventExtendedRemove
EventExtendedStart
EventExtendedWait
FirmwareVersionGet
GatheringConfigurationGet
GatheringConfigurationSet
GatheringCurrentNumberGet
GatheringDataAcquire
GatheringDataGet
GatheringExternalConfigurationGet
GatheringExternalConfigurationSet

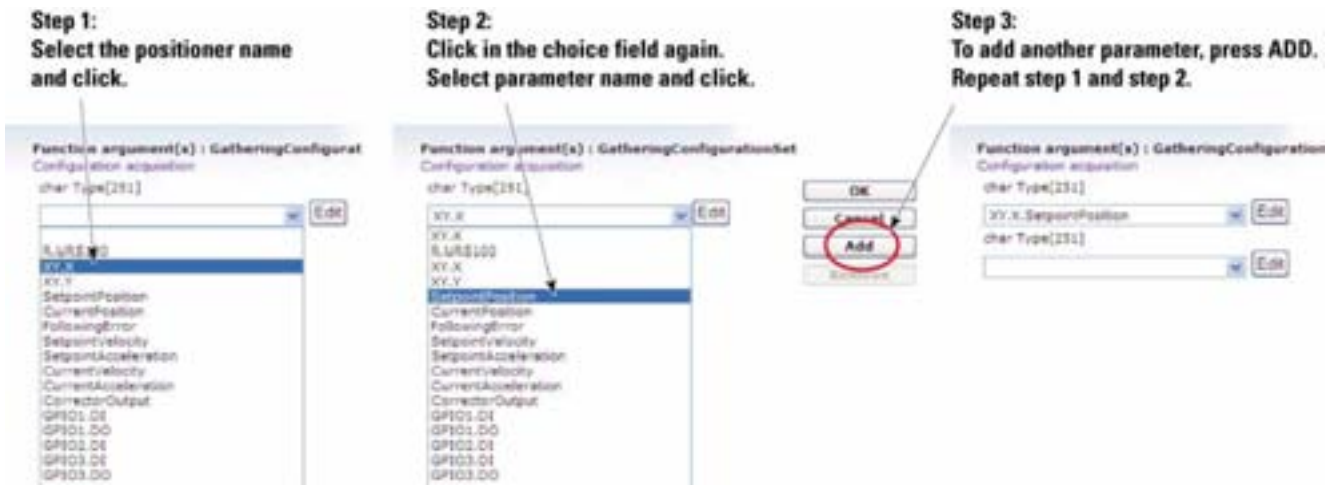
To execute a function from the Terminal, do the following:

1. Select a function and double-click to validate the selection.
2. Define the arguments for the function.

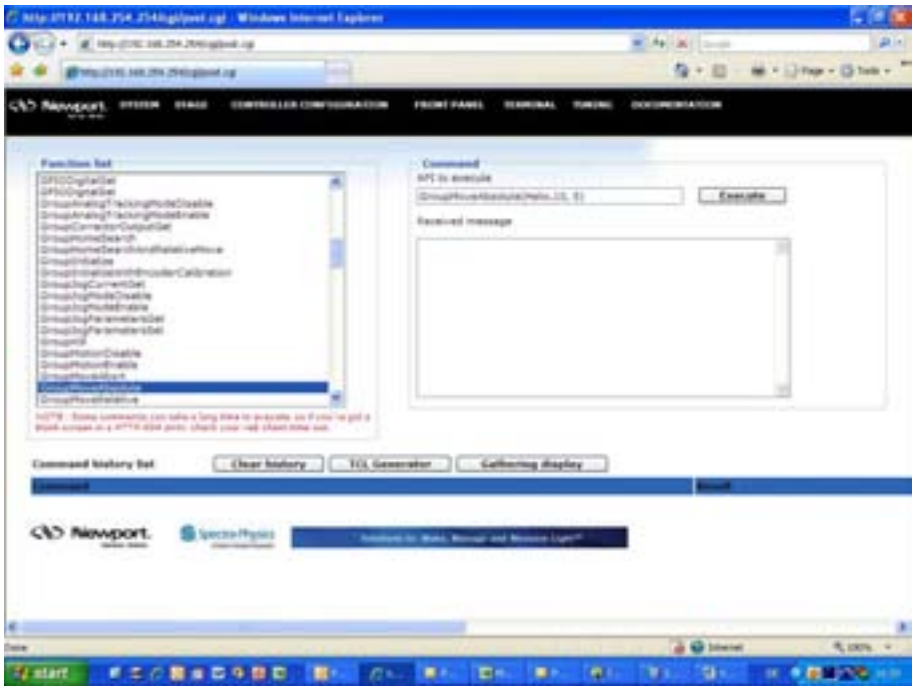
   For functions with dynamic arguments “ADD” and “REMOVE” buttons are available. Alternative, you can use a “,” as separator between different arguments.



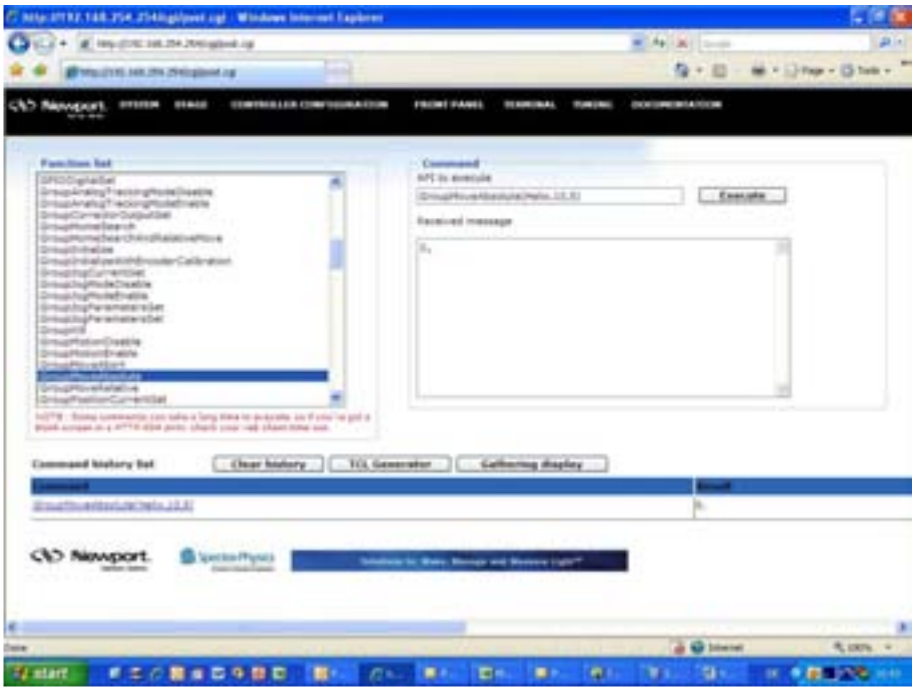
   For some arguments like ExtendedEventName, ExtendedActionName or GatheringType, the argument name is not directly accessible. In these cases, define the first part of the argument name, then click in the choice field again and define the second part of the argument name. See example below for defining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



3. When all arguments are defined, click "OK". Now you can review the final syntax of the function and can make final text changes, if you want. When done, click "Execute".



4. When the function is executed, you'll see the controller's response in the Received message window. A returned 0 means that the function has been executed successfully. In all other cases, you'll receive an error code. Use the function ErrorStringGet() to get more information about the error.



The functions are listed in alphabetic order. Only those functions are listed that are available with 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.

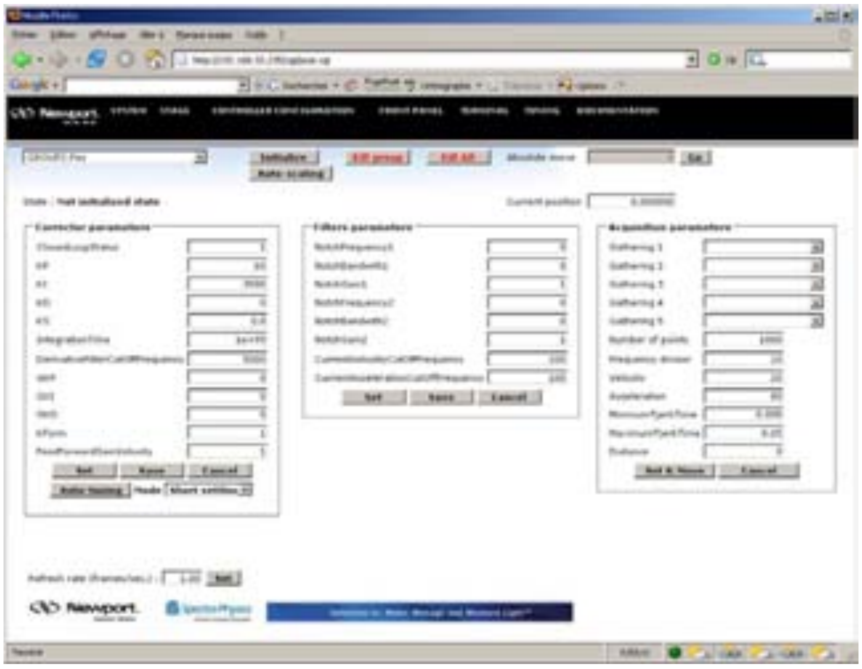
4.21 TUNING – Auto-Scaling

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

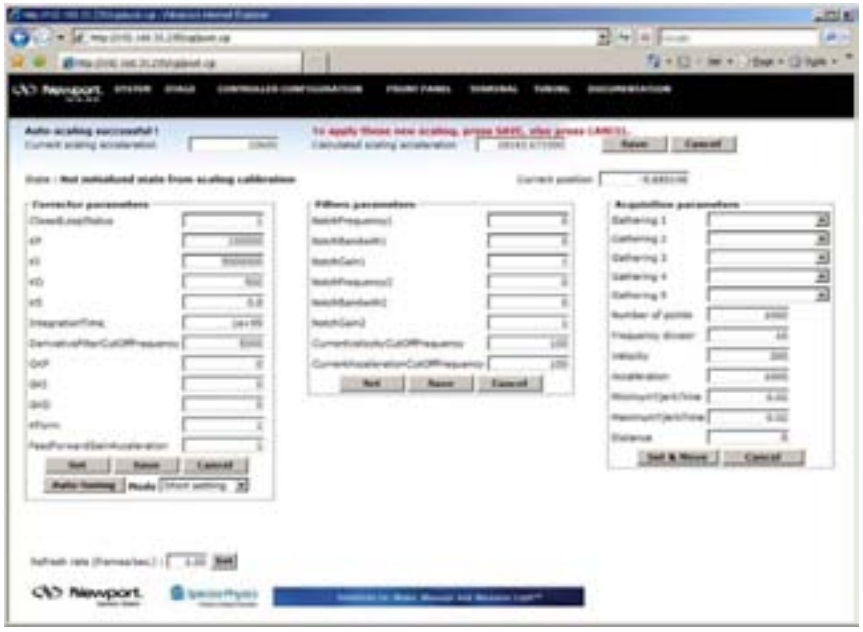
Repeat Auto-scaling with any major change of the payload on the positioner. With no major change of the payload, there is no need to redo the auto-scaling.

To perform an auto-scaling, do the following:

1. Select the main tag TUNING. Select a positioner name. The following screen appears:



2. Click “Kill group”, then click “Auto-scaling”. The stage vibrates for a couple of seconds. Then, the following message appears:



Auto-scaling successful !

To apply these new scaling, press SAVE, else press CANCEL.

Current scaling acceleration

Calculated scaling acceleration

State : Not initialized state from scaling calibration

3. To save the recommended values, click “Save”. To apply these new values, you need to reboot the controller. Your positioner should now work properly.

Repeat auto-scaling with any major change of the payload of the positioner.

NOTE

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

4.22 TUNING – Auto-Tuning

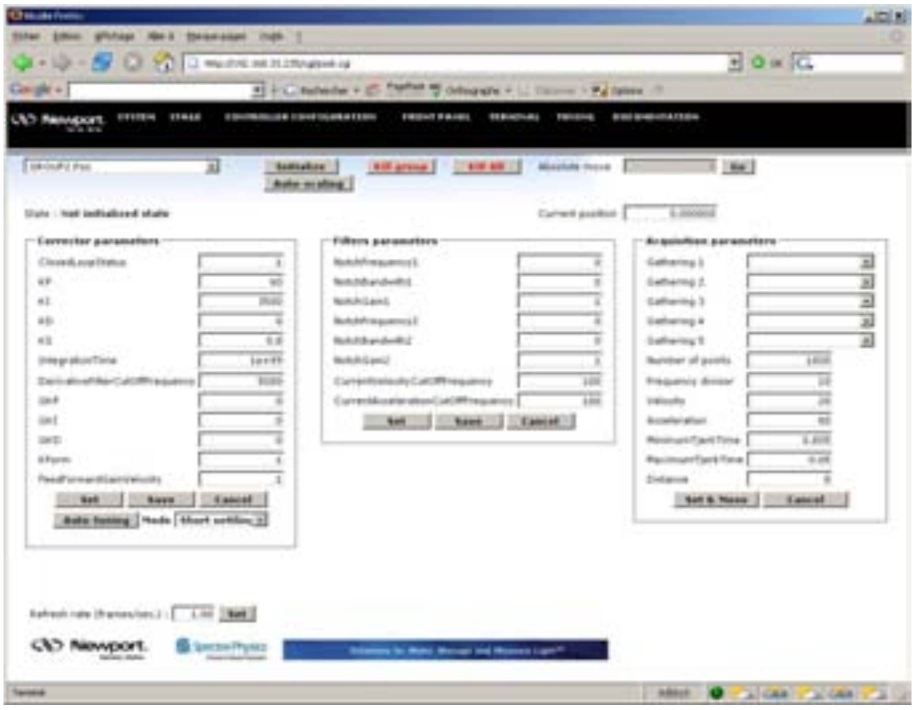
NOTE

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 consistent high quality results for the vast majority of applications. Use the tuning tool with Newport positioners only when not fully satisfied with the dynamic behavior of your positioners.

The following provides a brief description of the TUNING tool:

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



State : Not initialized state

Current position 0.000000

Corrector parameters

ClosedLoopStatus	1
KP	60
KI	2500
KD	0
KS	0.8
IntegrationTime	1e+99
DerivativeFilterCutOffFrequency	5000
GKP	0
GKI	0
GKD	0
KForm	1
FeedForwardGainVelocity	1

Mode Short settling

Filters parameters

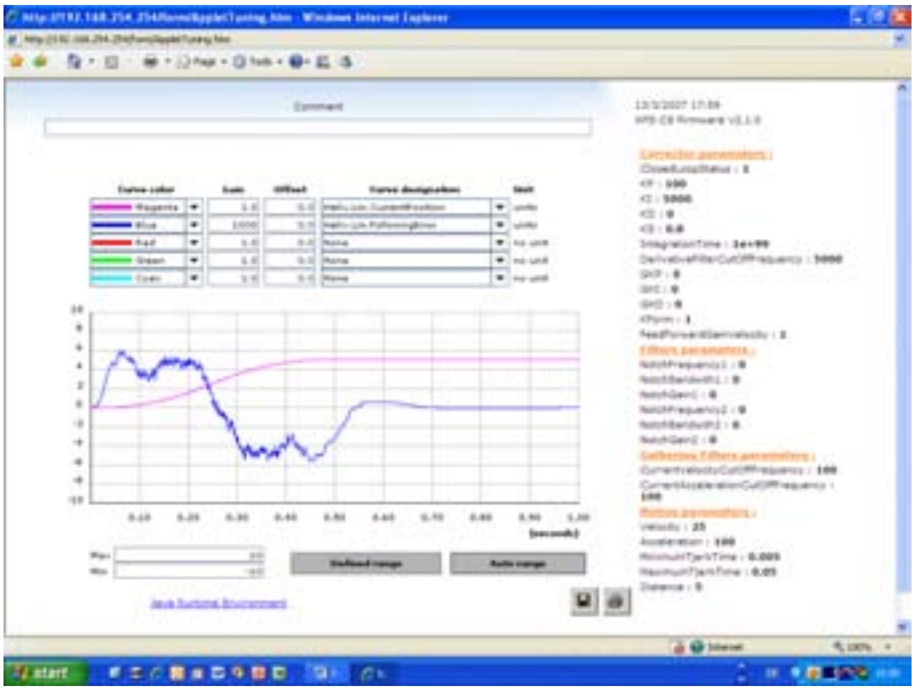
NotchFrequency1	0
NotchBandwith1	0
NotchGain1	1
NotchFrequency2	0
NotchBandwith2	0
NotchGain2	1
CurrentVelocityCutOffFrequency	100
CurrentAccelerationCutOffFrequency	100

Acquisition parameters

Gathering 1	
Gathering 2	
Gathering 3	
Gathering 4	
Gathering 5	
Number of points	1000
Frequency divisor	10
Velocity	20
Acceleration	80
MinimumTjerkTime	0.005
MaximumTjerkTime	0.05
Distance	0

Refresh rate (frames/sec.) : 1.00

2. Perform a gathering with your current parameter settings.
   To do so, Initialize and home your positioner, then move to the desired start position. Define the gathering data: For a stage tuning, Newport recommends gathering only the following error and the current position. Define a motion distance. 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, or every 100 μs. With most positioners, it is sufficient to set a value of 10, means one data set every 1 ms. Define the number of points in relation to the distance, the frequency divisor, the velocity and the acceleration. Define the velocity, acceleration and jerk time. When done, click "Set & Move".
3. The gathering results are displayed in a Java applet window. To see the results, you have to install Java™ Runtime Environment Standard Edition on your computer. The XPS provides a direct link to download Java™ Runtime when not installed on your computer.



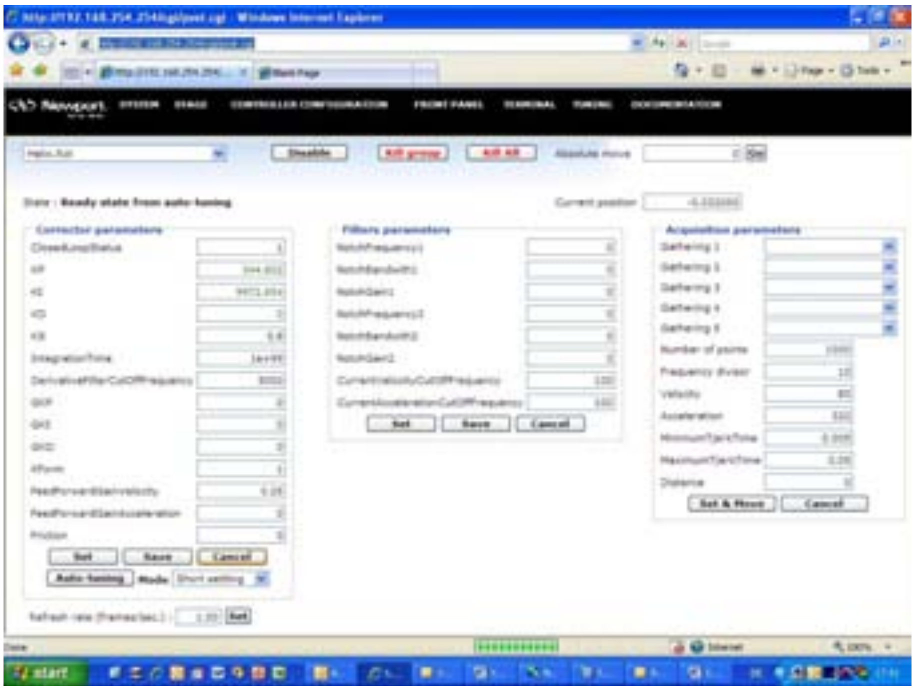
Curve color	Gain	Offset	Curve designation	Unit
Magenta	1.0	0.0	Hello.Lin.CurrentPosition	units
Blue	1000	0.0	Hello.Lin.FollowingError	units
Red	1.0	0.0	None	no unit
Green	1.0	0.0	None	no unit
Cyan	1.0	0.0	None	no unit

13/3/2007 17:59
XPS-C8 Firmware V2.1.0

Corrector parameters :
ClosedLoopStatus : 1
KP : 100
KI : 5000
KD : 0
KS : 0.8
IntegrationTime : 1e+99
DerivativeFilterCutOffFrequency : 5000
GKP : 0
GKI : 0
GKD : 0
KForm : 1
FeedForwardGainVelocity : 1
Filters parameters :
NotchFrequency1 : 0
NotchBandwidth1 : 0
NotchGain1 : 0
NotchFrequency2 : 0
NotchBandwidth2 : 0
NotchGain2 : 0
Gathering Filters parameters :
CurrentVelocityCutOffFrequency : 100
CurrentAccelerationCutOffFrequency : 100
Motion parameters :
Velocity : 25
Acceleration : 100
MinimumTjerkTime : 0.005
MaximumTjerkTime : 0.05
Distance : 5

4. When satisfied with the results, there is no need to tune the stage. When not satisfied, go back to the tuning page and move back to the start position.
5. Next to the button auto-tuning you have a choice field for the Auto-tuning parameters. 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 oscillations. Choose "High robustness" to improve the robustness of your system and to avoid oscillations during or after a motion. A "High robustness" tuning, for instance, can avoid oscillation of rotation stage with high payload inertias. When done with your selection, press Auto-tuning.

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



7. Press "Set" to apply the new parameters. "Set" only changes the temporary working parameters. You can still recover the old parameters by rebooting the system.
8. To test the behavior of your system with the new parameters, redo the same data gathering and compare the results. You can now also make manual changes to the settings. For further instructions and recheck the behavior.
9. To permanently save the settings to the stages.ini, press "Save". "Save" overwrites the current settings in your stages.ini. Press "Save" only when fully satisfied with the results. For recovery, Newport recommends taking always 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.

5.0 FTP (File Transfer Protocol) Connection

FTP is the protocol for exchanging files over the Internet. It 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 review the information inside the XPS controller, to download documentation, to transfer configuration files (to modify them directly), to transfer TCL scripts, etc…

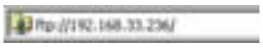
To connect to the FTP server:

Start the XPS controller and wait until the boot sequence completes.

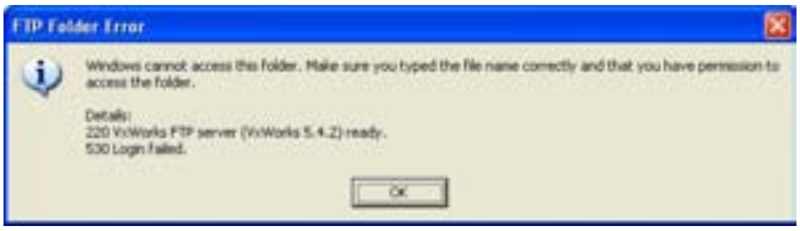
Open an Internet browser window.

Connect to the FTP server with the IP address of the controller:

Example


ftp://192.168.33.236/

It is normal that the following error message appears:

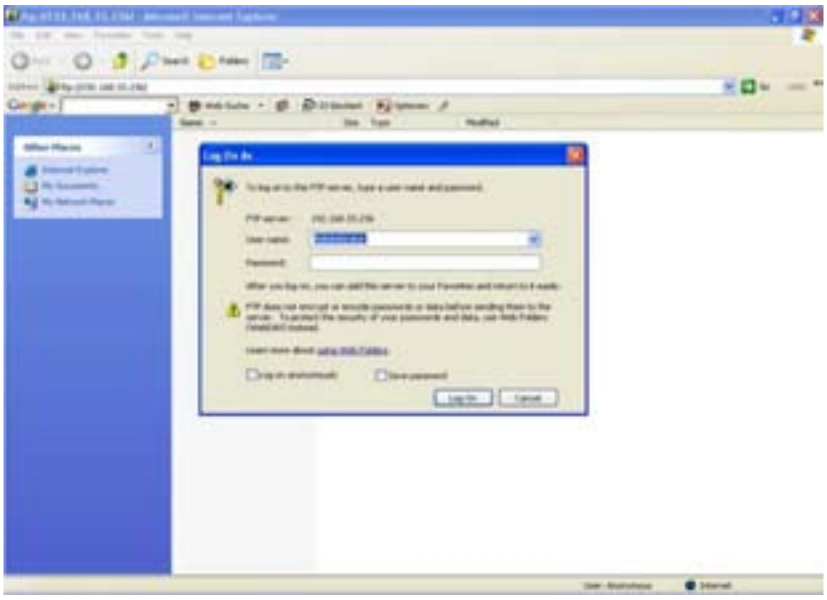

FTP Folder Error

Windows cannot access this folder. Make sure you typed the file name correctly and that you have permission to access the folder.

Details:
220 VxWorks FTP server (VxWorks 5.4.2) ready.
530 Login failed.

Press OK.

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


Log On As

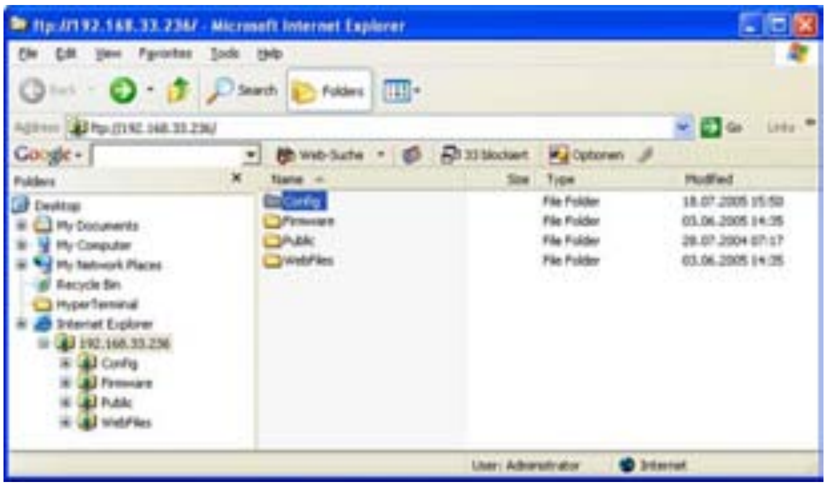
To log on to this FTP server, type a user name and password.

FTP server: 192.168.33.236
User name:
Password:

[illegible]

[illegible]

Specify the XPS user name and XPS 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 with Windows Explorer.


ftp://192.168.33.236/ - Microsoft Internet Explorer

Name	Size	Type	Modified
Config		File Folder	18.07.2005 15:50
Firmware		File Folder	03.06.2005 14:35
Public		File Folder	28.07.2004 07:17
WebFiles		File Folder	03.06.2005 14:35

User: Administrator   Internet

6.0 Maintenance and Service

6.1 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.

6.2 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).

Instrument serial number (on rear panel) or original order number.

Description of the problem.

If the instrument 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 as shown at the end of this User's Manual and include it with your shipment.

6.3 Troubleshooting

For troubleshooting, the user can query different error and status information from the controller. The XPS controller can provide 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 belonging 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 belonging 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 use to retrieve the Group Status:

GroupStatusGet: Returns the group status code.

GroupStatusStringGet: Returns the description belonging 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 the Newport web utilities.

6.4 Updating the Firmware Version of Your XPS Controller

Users can regularly update the controller with new firmware releases. To review the current available versions, refer to the FTP server:

ftp://download.newport.com/MotionControl/Current/MotionControllers/XPS/Updates/

There are separate folders for the different versions, each folder contains the Firmware pack, the RC IHM and the Stage data base.

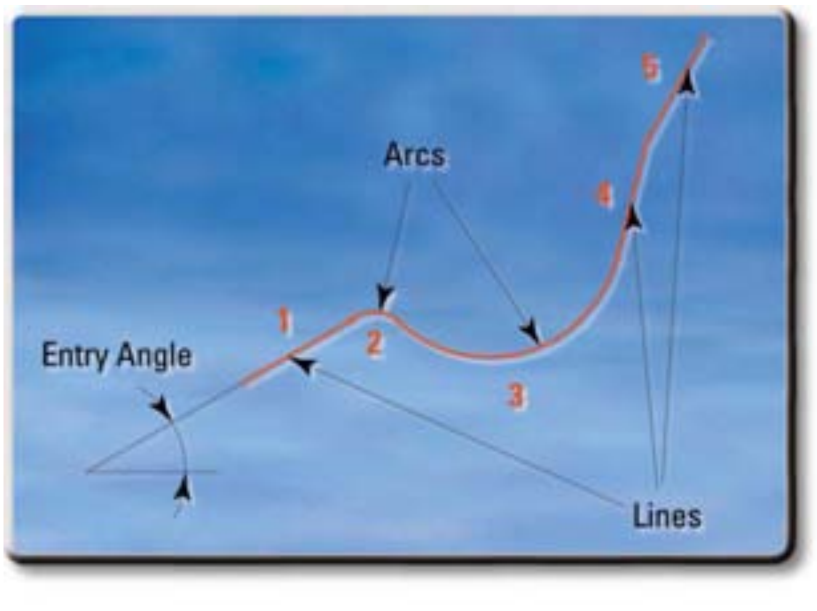
To install a new version, follow these instructions:

1. Double click on the XPS-C8_Install.exe file (in FirmwarePack) and follow the instructions without rebooting the controller.

2. Double click on the StageDBInstall.exe file (in StageDataBase), follow the instructions and reboot the controller.

3. Uninstall both installers. To do so, click on Windows Start menu → Programs → Newport → XPS Tools → Firmware Pack FTP Updater → Uninstall Firmware Pack FTP Updater, then click on "Accept", and "OK".

A history file for the firmware and the stage database is added to the web documentation.



Motion Tutorial

7.0 XPS Architecture

7.1 Introduction

The architecture of the XPS firmware is based upon an object-oriented approach. Objects are key to understanding object-oriented technology. 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:

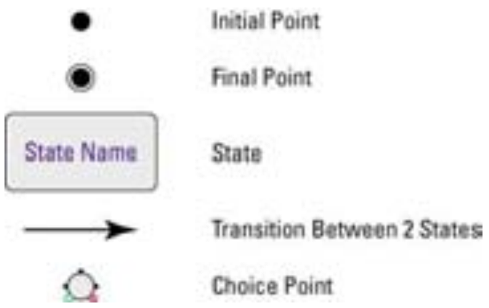
Modularity: The source code for an object can be written and maintained independently of the source code for other objects. Also, an object can be easily passed around in the system.

Information hiding: 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.

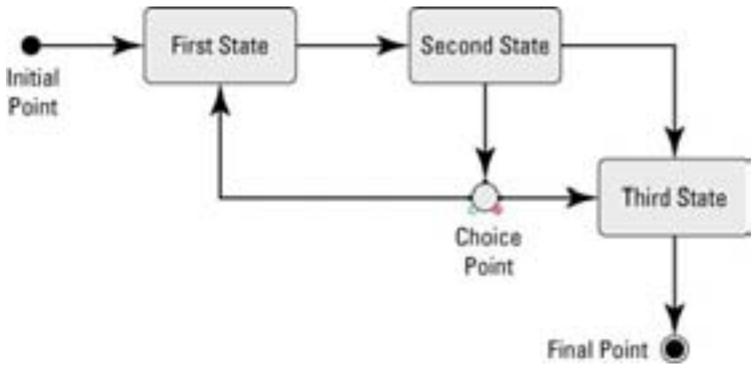
All objects have a life cycle and state diagrams are used to show the life cycle of the objects. The transition from one state to another will be initiated by the reception of a message from another object. Like all other diagrams, state diagrams can be nested in different layers to keep them simple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

7.2 State Diagrams

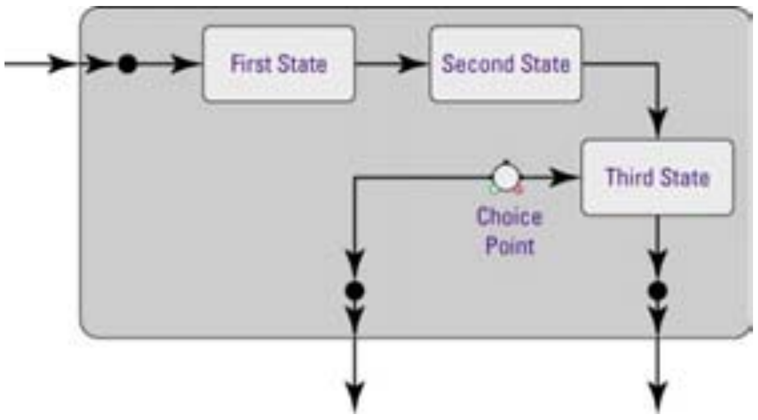
State diagrams are a way to describe the behavior of each group or object. They represent each steady state of a group and every transition between states in an exhaustive way. State diagrams contain the following components:



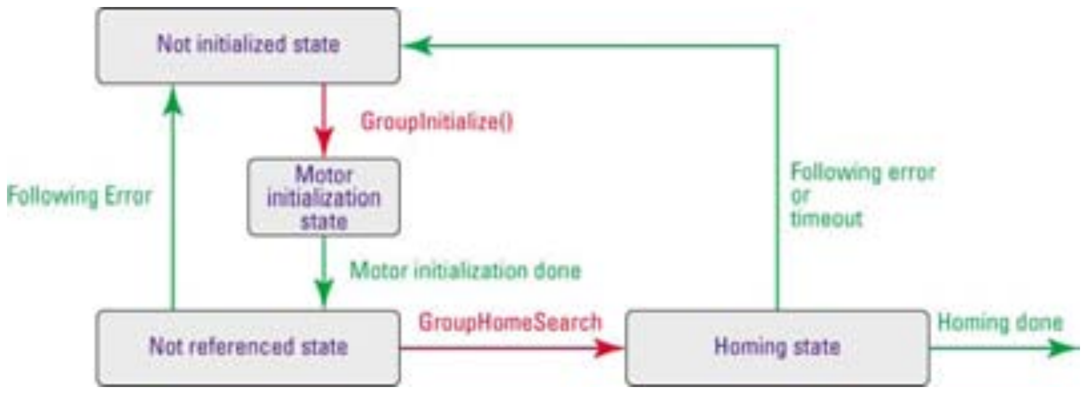
Here is an example of a simple stage diagram:



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.

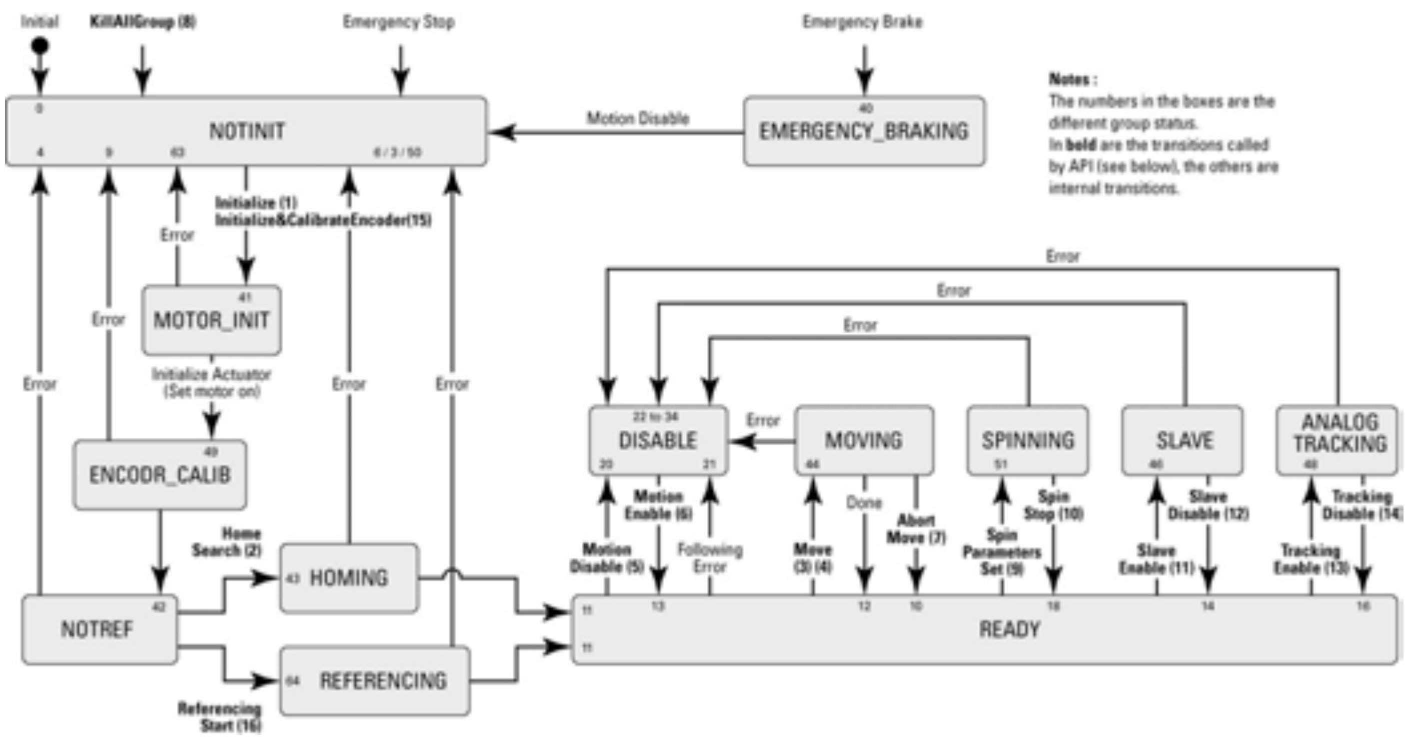
All positioners of a group are bundled together for security handling. Security handling on different groups is treated independently. Following is a list of the different faults and consequences that can happen in the XPS controller:

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 are associated to the state and to the event that has generated the transition (if any). The function **GroupStatusStringGet (StateNumber)** returns the state description corresponding to the state number.



Notes :
The numbers in the boxes are the different group status.
In **bold** are the transitions called by API (see below), the others are internal transitions.

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.

7.3 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, 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, you can 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. You can also use this same command 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 to be learned by the user 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 gets 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), then only the XY table will be disabled while the auto-focusing tool (a vertical stage that is defined as a separate SingleAxis group) can continue to function.

Each of the four available motion groups has specific features:

7.3.1 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.

7.3.2 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.

7.3.3 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.

7.3.4 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.

7.3.5 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, an XYZ group cannot be slaved to another group. However, any positioner of a MultipleAxes group can be a master to a slaved SingleAxis group.

For further details on the different types of trajectories (Line-Arc, Spline, PVT) and how to define these trajectories, please refer to chapter 9.0: “Trajectories“.

For simple point-to-point motion of individual positioners, the most commonly used group is the SingleAxis group.

7.4 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 natural 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: TargetPostion, 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 controller calculates a set of SetpointPositions to determine where the positioner should be at each time.

When the positioner is controlled by a digital servo loop with a PID corrector, a 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 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 natural 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.

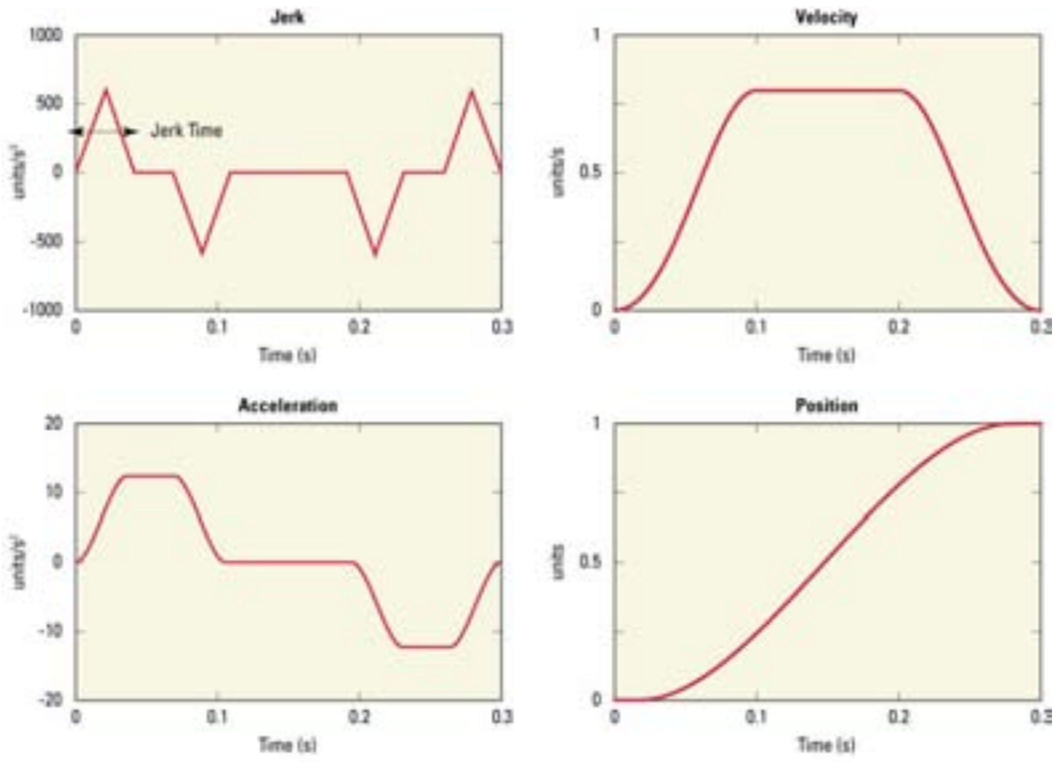
The TargetPosition, SetpointPosition, CurrentPositon and FollowingError can be queried from the controller using the appropriate function calls.

8.0 Motion

8.1 Motion Profiles

When talking about motion commands, we refer to certain strings sent to a motion controller that will initiate a certain action, usually 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 specifics worth mentioning on the motion profiler used by 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 1 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/s²
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 17: 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 1, 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 allows for a much smoother motion. The jerk time defines the time 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 systems behavior to different motion step sizes.

NOTE

Because of the jerk controlled acceleration time, 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 the positioner. When using the XPS controller with Newport stages these settings are automatically done during the configuration of the system.

The velocity, acceleration and jerk time parameters can be modified by the function **PositionerSGammaParametersSet().**

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/s2 and minimum and maximum jerk time to 0.02 seconds. The set velocity and acceleration have to 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 be set back to the values set in the stages.ini file.

In actual use, the XPS places a priority on the displacement position values over the velocity value. To reach the exact demanded position, it may be possible that the speed of the positioner varies 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.

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 can 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 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 follow:

Following error = SetpointPosition - CurrentPosition

The functions to query the SetpointPosition and the CurrentPosition are:

GroupPositionCurrentGet() and GroupPositionSetpointGet()

8.2 Home Search

Home search is a specific motion process. Its goal is to find a reference point along the travel accurately and repeatedly. 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 protect 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 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 an XPS predefined motion process that moves a stage to a unique reference position. 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 (these are the ones supported by XPS). 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, the controller must find a reference position that is unique to the entire travel, called a home switch or origin switch.

An important requirement is that this switch must have the same accuracy 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 accuracy.

If, on the other hand, a rotary encoder is used, homing becomes more complicated. To have the same accuracy, 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 accurate or repeatable enough. The solution is to use both in a dedicated search algorithm as follows.

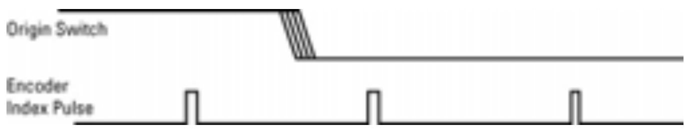


Figure 18: Home (Origin) Switch and Encoder Index Pulse.

A Home switch (Figure 18) 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 looking to the origin switch level, the controller knows already on which side of the transition the positioner is and which direction to move.

The task of the home search process is to identify 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 19).

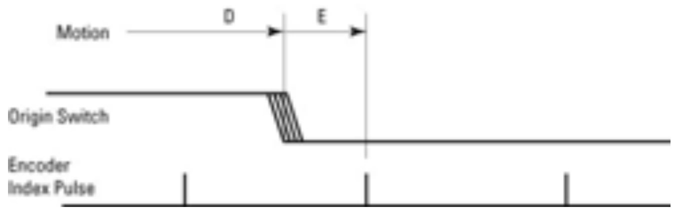


Figure 19: Slow-Speed Origin Switch Search.

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

The homing process described 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 20.

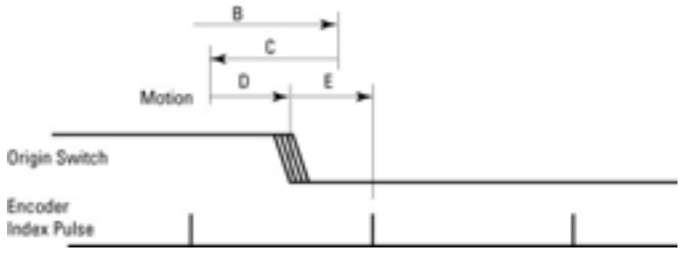


Figure 20: High/Low-Speed Home (Origin) Switch Search.

Motion segment B is performed at high speed, with 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 velocity (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 21.

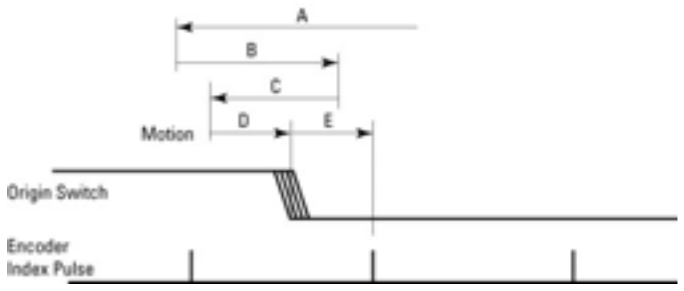


Figure 21: Home (Origin) Search from Opposite Direction.

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

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 accurate and repeatable reference position.

There are 8 different home search processes available in the XPS controller:

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 of Newport standard stages as described above.

MechanicalZeroHomeSearch is used with positioners that have a hardware home switch but no zero index from the encoder.

IndexHomeSearch is used with positioners that have a home index, but no home switch signal. In this process, the positioner always moves initially in the same direction to find the index. When a limit switch is detected, the direction of motion reverses until the index is found. Note: For users with CIE03 boards, if a limit is detected before the index, there will be an emergency brake and the group will go in NOT_INITIALIZED status.

CurrentPositionAsHome is used when the positioner has no home switch or index. This process will keep the positioner's home at its current location.

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. The positioner homes to a position that is different from the MechanicalZeroAndIndexHomeSearch location.

MinusEndOfRunHomeSearch uses the positioner's minus end-of-run limit for homing.

PlusEndOfRunHomeSearch uses the positioner's plus end-of-run limit for homing. Note: This home search works only with the CIE05 board or later versions.

The home search process is set up in the stages.ini file. When using the XPS controller with Newport 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)

8.3 Referencing State

The XPS 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 to 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 offers the referencing state as an alternative to the predefined XPS home search processes.

NOTE

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

The referencing state is a parallel state to the homing state, see state diagram page 76, Figure 22. 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)** is available to 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 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 parameter 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 in three categories: Moves that stop on a sensor event, moves of certain displacement, and position counter reset actions.

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	v	v			v
LatchOnHighToLowTransition	v	v			v
LatchOnIndex			v		v
LatchOnIndexAfterSensorHighToLowTransition	v	v			v
SetPosition			v	v	
SetPositionToHomePreset			v		
MoveToPreviouslyLatchedPosition			v		v
MoveRelative			v	v	

8.3.1 Move on sensor events

The “move on sensor events” start 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 either **MechanicalZero** or **MinusEndOfRun**. With **LatchOnIndex** and **LatchOnIndexAfterSensorHighToLow**, the latching happens on the index signal. With **LatchOnIndexAfterSensorHighToLow** the latching happens on the first index after a high to low transition on the defined sensor (**MechanicalZero** or **MinusEndOfRun**). Because of the dedicated hardware circuitries used for the position latch, there is essentially no latency between sensor transition detection and position acquisition.

In all cases, the motion stops after the latch. However, this means that the motion doesn’t rest on the sensor transition, but at some small 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 securities are still enabled until the **MinusEndOfRun** sensor is specified with a **GroupReferencingActionExecute()** function. When specified, the limit switch securities get disabled and will only get re-enabled with the function **GroupReferencingStop()**.

The Parameter is a signed velocity (floating point). This means that the direction of motion is specified by the sign of the parameter.

8.3.2 Moves of Certain Displacements

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

MoveRelative

The action **MoveRelative** allows commanding 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 ones 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 8.3.1: “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 ones defined by a **PositionerSGammaParametersSet**.

8.3.3 Position Counter Resets

The "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 referencing state.

8.3.4 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 state diagram below:

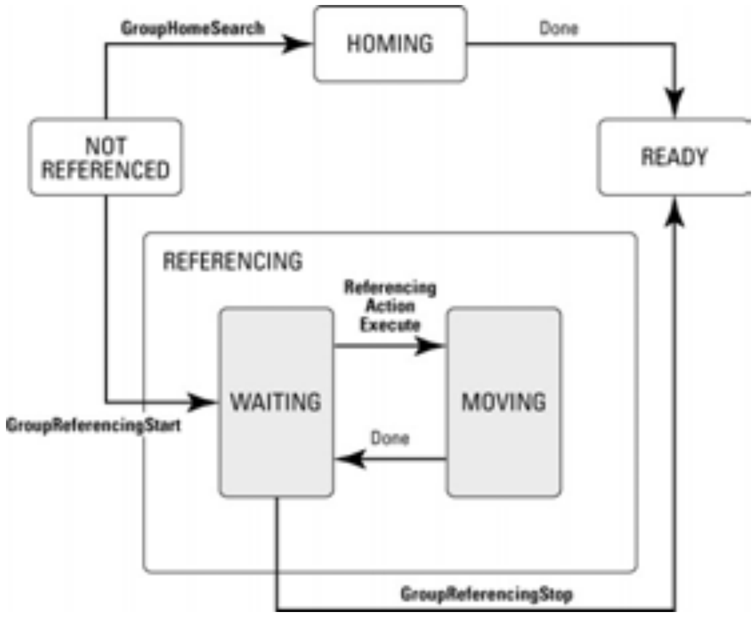


Figure 22: State Diagram.

8.3.5 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)
```

8.4 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 the 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 being updated at 10 kHz and the profile generator at 2.5 kHz, providing highly accurate closed loop positioning. Between profiler and corrector there is a time-based linear interpolation to accommodate for 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 0 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 an integer of an encoder steps without accumulating errors. See section 8.4 for further details.

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. It means at any time, any positioner of that group has executed the same part of its trajectory, while the "slowest" positioner defines the speed and acceleration of the other positioners. 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 consist of one XY group called ScanTable and one SingleAxis group called FocusStage. ScanTable has two positioners in use, 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 8.1). 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 next following moves. When wanting to change the velocity or acceleration of a positioner during the motion, use the Jogging mode (see section 8.5)

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)** gets accepted when the motion has been also 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)**. If you want to stop this motion, you need to 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, need to be managed by separate threads (see also section 16.4 for details).

8.5 Motion Done

The XPS controller supports two methods to define when a motion is completed (MotionDone): The theoretical MotionDone and the VelocityAndPositionWindow MotionDone. The preferred method for MotionDone used is set up in the stages.ini file. In theory, MotionDone is when motion is completed as defined by the profiler. 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 VelocityAndPositonWindow 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:

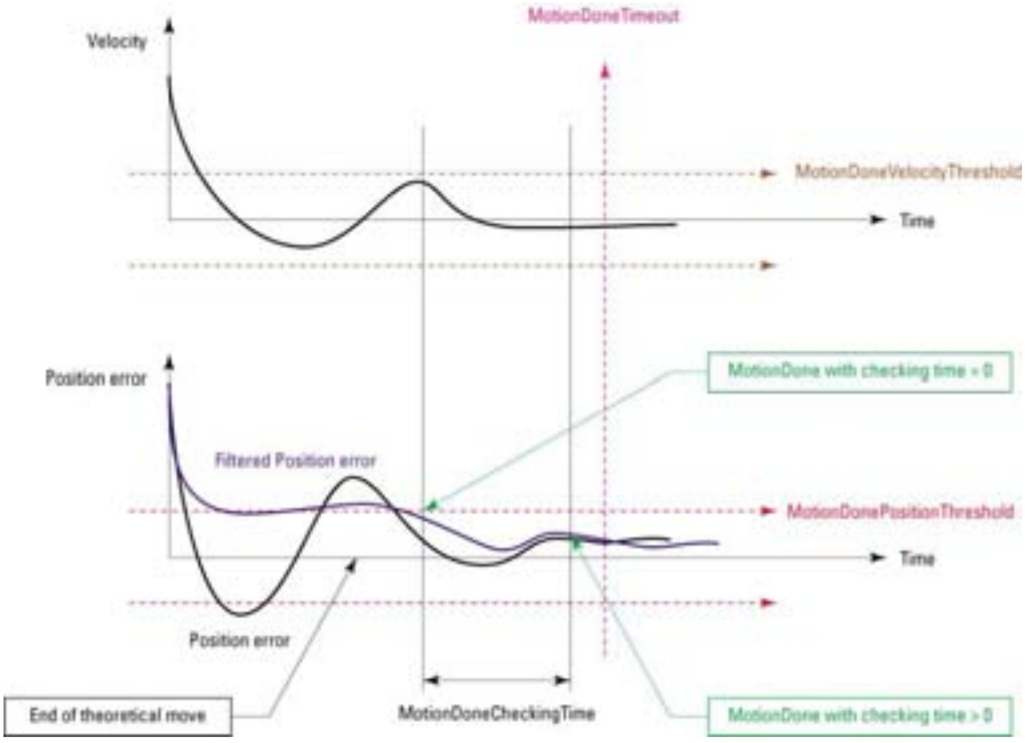


Figure 23: Motion Done.

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 chapter 5 "FTP 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 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, 0.00 and 0.5.

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

This function replaces the parameters by the new entered values.

8.6 JOG

Jog is an indeterminate motion defined by velocity and acceleration parameters. 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 applicable to positioners and to motion groups. 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 absolute 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 a 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 taken into account only every 0.4 ms. In Jog mode the profiler uses a trapezoidal motion profile (see also section 8.1 for further details on motion profiles).

8.7 Master Slave

In master slave mode, any motion axis can be electronically geared to another motion axes, including 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 (means all 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 than example 1, all motions, including the motion done during the home search routine, are done 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 you might not be able to use the full capabilities of the master or the slave positioners.

8.8 Analog Tracking

Analog tracking allows the control of 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.

8.8.1 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 is 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 motion, 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)

8.8.2 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 24.

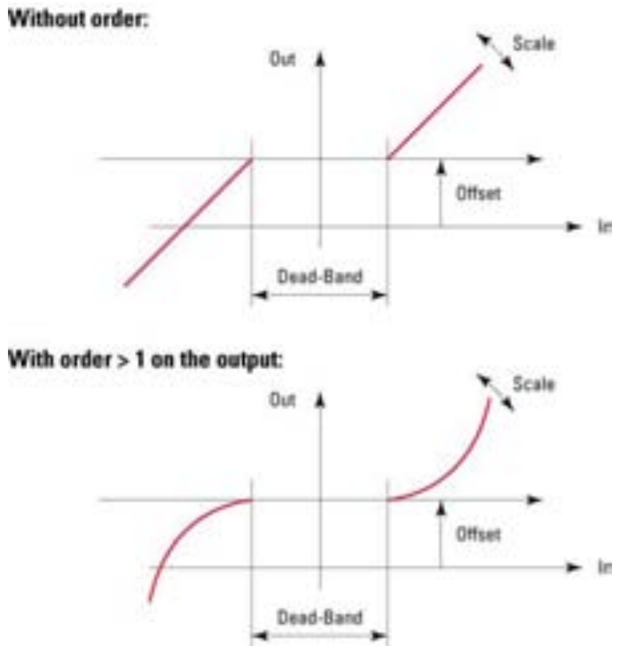


Figure 24: The Relationship Among Offset, Scale, Dead Band & Order.

The tracking velocity calculates as follow:

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)

9.0 Trajectories

The XPS controller supports 3 different types of trajectories:

The line-arc trajectory is a trajectory defined by a number 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 a constant speed (speed being the scalar of the trajectory velocity) throughout the entire path, not including 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 major requirements of a spline is 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 for 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 allows for the most complex trajectories and is only available with MultipleAxes groups. In a PVT trajectory, each trajectory element is defined by the end position 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.

9.1 Line-Arc Trajectories

9.1.1 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 major 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 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.

9.1.2 Trajectory Conventions

When defining and executing a line-arc trajectory, a number of rules must be followed:

The motion group must be a 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. Though, N-times (N defined by user) non-stop executions of the same trajectory without stopping is possible. As the trajectory is stored in a file, the trajectory maximum size (maximum elements number) is unlimited for practical purposes.

Two types of 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 some velocity discontinuities from one segment to next (which means that the line-arc trajectory continuity property is R0). An excessive velocity discontinuity at joints can damage the mechanics, 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.

9.1.3 Geometric Conventions

The coordinate system 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 stages.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, represented as floating point numbers. Angle origin and signs follow the trigonometric convention: positive angles are measured counter-clockwise.

9.1.4 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.

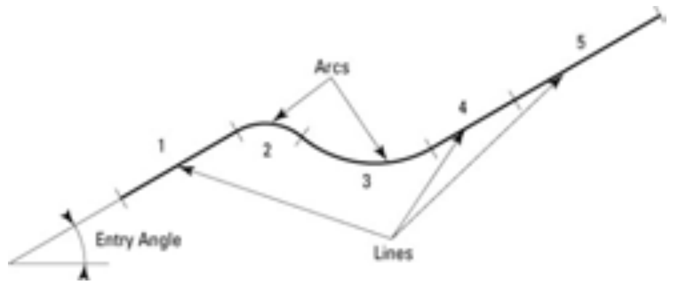


Figure 25: Line-arc trajectory example.

Figure 25 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 angle discontinuity. The angle 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 angle difference smaller than the allowed angle of discontinuity, as shown in the Figure 26.

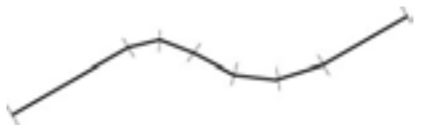


Figure 26: Contouring with linear lines only.

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 of the following error. The larger the angle of discontinuity, the larger the shock and the following error will be. Special consideration must be given to both these effects when increasing the maximum discontinuity angle from its default value.

9.1.5 Define Lines

A line element is defined by specifying the (X_i, Y_i) ending point.

The element 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 starting point (0, 0).

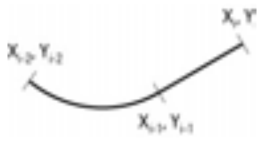


Figure 27: Line element to (X_i, Y_i) position coordinates.

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.

9.1.6 Define Arcs

An arc is defined by specifying the radius R and the sweep angle A (Figure 28).

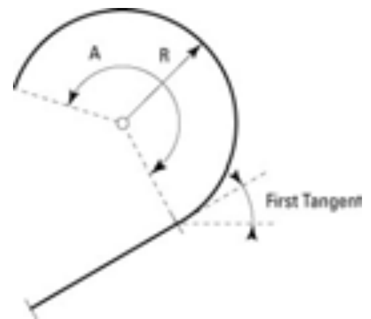


Figure 28: An arc defined with radius and angle.

Both radius and sweep angles are expressed in double precision floating point numbers. The sweep angle can range from 1 E^{-14} to 1.797 E^{308} allowing a definition of arcs from a fraction of a degree to a practically infinite number of overlapping circles.

9.1.7 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":	Define 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":	Define 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.

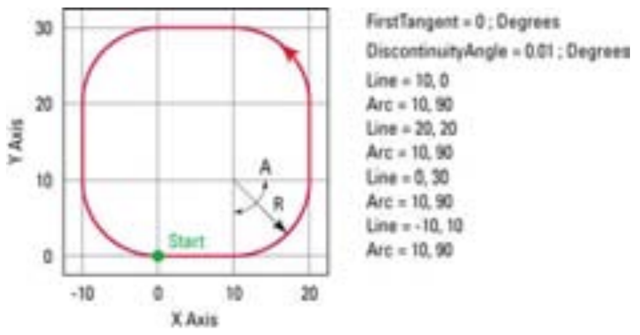
An element can be a "Line" or an "Arc":

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).

9.1.8 Trajectory File Examples

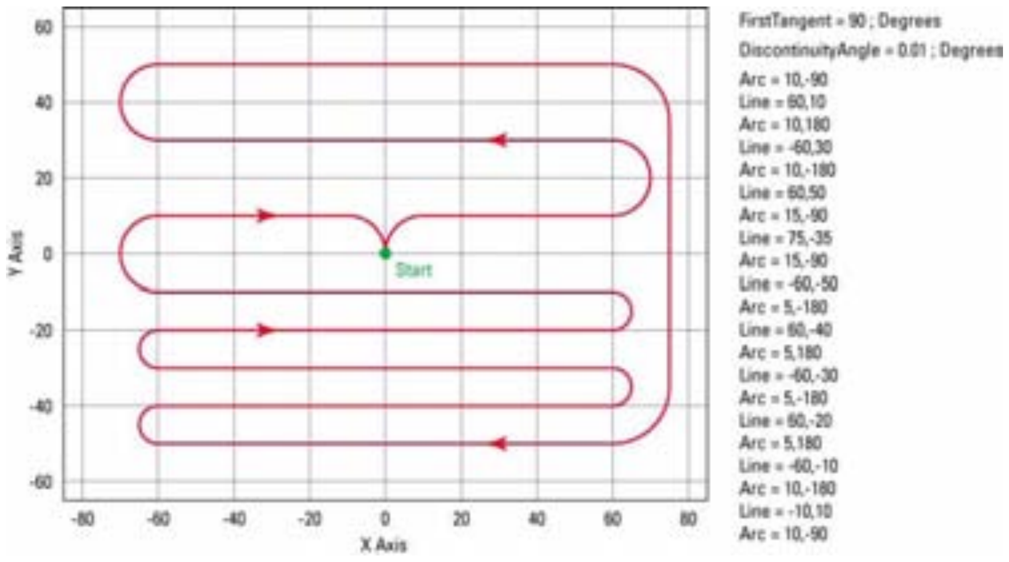
The following is an example of a trajectory file that represents a rectangle with rounded corners and with the end point equal to the starting point:



```
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 29: 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 30: Graphical display of the second line-arc trajectory data file example.

9.1.9 Trajectory Verification and Execution

To verify and execute a line-arc trajectory, there are four functions:

XYLineArcVerification(): Verifies a line-arc trajectory data file.

XYLineArcVerificationResultGet(): Returns the last trajectory verification results, actuator by actuator. This function works only after a XYLineArcVerification().

XYLineArcExecution(): Executes a trajectory.

XYLineArcParametersGet(): Gets the trajectory's current execution parameters. This function works only during the execution of the trajectory.

The **function XYLineArcVerification()** can be executed at any time and is independent from the trajectory execution. This function performs the following:

Checks the trajectory file for data 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 helps define the parameters for the 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 a 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 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 Square Root $\{Summ (V_{max\_actuator})^2\}$. However, this value does not take into account that the positioners' acceleration 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 positioner parameters. At a trajectory, acceleration of 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 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 the 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 interference. In case of an error during the 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 go 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 the trajectory velocity and the Min{All $A_{max\_actuator}$} for the 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 at 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 in execution.

9.1.10 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 requirement for the XYGroup.XPositioner, the positive or right travel requirement 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.

9.2 Splines

9.2.1 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 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 equal by default.

9.2.2 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. Though, N-times (N defined by user) non-stop executing a trajectory is possible. As the trajectory is stored in a file, the trajectory 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 presenting parameter varying from 0 (corresponding to P_{i-1}) to 1 (corresponding to P_i).

Spline 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 spline trajectory continuity property is R^1.

9.2.3 Geometric Conventions

The coordinate system is a 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, represented as floating point numbers. Angle origin and sign follow the trigonometric convention: positive angles are measured counter-clockwise.

9.2.4 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 only act as control points. Within the class of precise interpolation we have:

Global polynomial interpolation: One polynomial presents the whole trajectory. Examples are Lagrange polynomial or Newton polynomial.

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, we have the cubic spline interpolation.

The interpolation methods are also classified by the continuity criterion Ck. An interpolating curve has the continuity C^k if it and its derivatives up to k-degree 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 pi is calculated based on the previous pi-1 and the next point p_{i+1} on the spline. In case of the spline curve tension τ = 1/2 (normal case), the **Catmull-Rom** spline is described by the following equation:

$$S(u) = \left(u^3\ u^2\ u\ 1\right) \Diamond \frac{1}{2} \Diamond \begin{array}{cccc} 1 & 3 & 3 & 1 \\ 2 & 5 & 4 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{array} \Diamond \begin{array}{c} p_{i\ 1} \\ p_i \\ p_{i+1} \\ p_{i+2} \end{array}$$

Here, pi 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 C1 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.

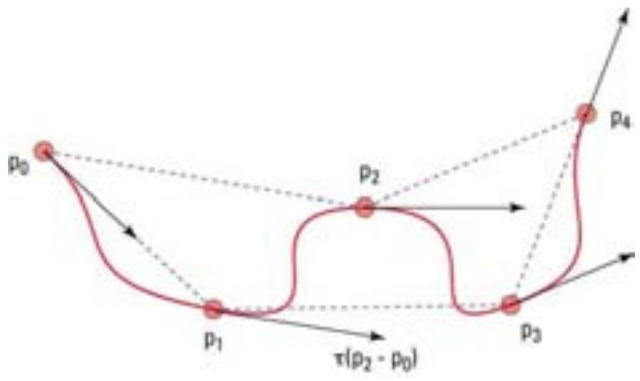


Figure 31: A Catmull-Rom spline.

9.2.5 Trajectory Elements Arc Length Calculation

Spline contouring at constant speed requires a high precision calculation of the segment's arc length. The segment's arc length can be expressed as follow:

$$L(u_0,u_1) = {}^{u_1}_{u_0}\sqrt{\frac{d}{du}Sx(u)^{\ 2} + \frac{d}{du}Sy(u)^{\ 2} + \frac{d}{du}Sz(u)^{\ 2}}\ du$$

Here, $u_0 = 0$ is the segment starting point and $u1 = 1$ 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}.

9.2.6 Trajectory File Description

The spline trajectory is described in a file that is in the ..\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 line format of the file is:

X-Position, Y-Position, Z-Position

The separator between the X-, Y-, and Z-Position is the comma.

As mentioned before, the first and last lines of the file are only needed for the first and the last segment spline interpolation. 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 furthermore shifted by this value.

Example

The spline trajectory file has the following format:

$$\begin{array}{ccc} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{array}$$

At the moment of the execution of the trajectory, 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:

$$\begin{array}{ccc} x_{c+x_0-x_1} & y_{c+y_0-y_1} & z_{c+z_0-z_1} \\ x_c & y_c & z_c \\ x_{c+x_2-x_1} & y_{c+y_2-y_1} & z_{c+z_2-z_1} \\ x_{c+x_3-x_1} & y_{c+y_3-y_1} & z_{c+z_3-z_1} \\ x_{c+x_4-x_1} & y_{c+y_4-y_1} & z_{c+z_4-z_1} \end{array}$$

9.2.7 Trajectory File Example

This trajectory represents a spiral from (0, 20, 0) starting point to (0, -20, 24) ending point. As described before, the trajectories 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 start 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			

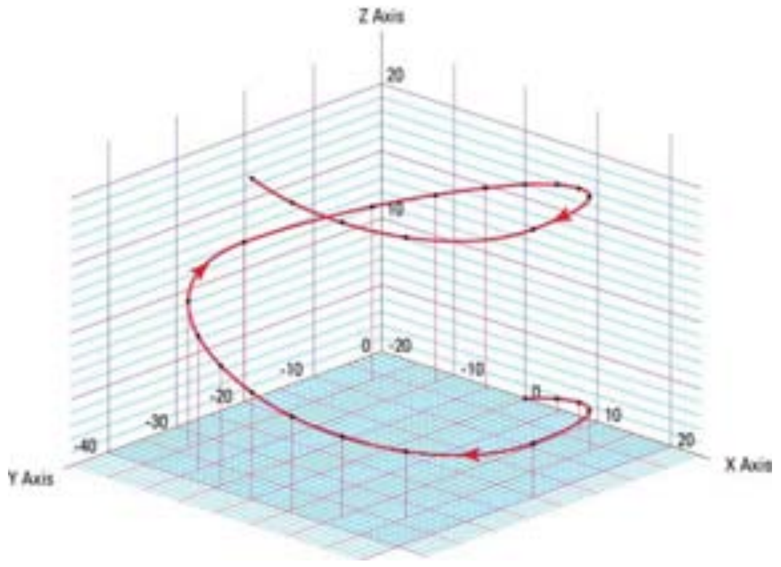


Figure 32: Executing the above trajectory data file with the Catmull-Rom spline algorithm.

9.2.8 Spline Trajectory Verification and Execution

To verify and execute a spline trajectory, there are four functions:

XYZSplineVerification(): Verifies a spline trajectory data file.

XYZSplineVerificationResultGet(): Returns the last trajectory verification results, actuator by actuator. This function works only after a XYZSplineVerification().

XYZSplineExecution(): Executes a trajectory.

XYZSplineParametersGet(): Returns the trajectory current execution parameters. This function works only during the execution 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 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 helps define the parameters for the 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 a XYZSplineVerification() 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 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 Square Root {Summ(Vmax_actuator)2}. However, this value does not take into account that also 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 the 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 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 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 without verifying the trajectory and without looking to the maximum possible values. In case of an error during the 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 almost surely executable default values which are the Min{All $V_{max\_actuator}$} for the trajectory velocity and the
Min{All $A_{max\_actuator}$} for the 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 in execution.

9.2.9 Examples of the Use of the Functions

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 or left travel requirement for the XYZGroup.XPositioner, the positive or right travel requirement 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 in execution (Spline1.trj), the trajectory velocity (10), the trajectory acceleration (100) and the number of the current executed trajectory element.

9.3 PVT Trajectories

9.3.1 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 8). These are available with MultipleAxes groups. A PVT trajectory is generated with continuous movements of the MultipleAxes 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 this segment. In the PVT data file, each element is presented by a line:

```
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.

9.3.2 Trajectory Conventions

When defining and 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 under ..\public\trajectories. Once a trajectory is started, it executes in background allowing other groups to work independently and simultaneously.

Each trajectory must have a beginning and an end. Endless (infinite) trajectories are not allowed. Though, N-times (N defined by user) non-stop execution of a trajectory is possible. As the trajectory is stored in a file, the trajectory 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 ti-1 with a velocity v_{i-1}) and positions P_i (at time ti with a velocity v_i). There is no direct link between the trajectories of the different positioners of the MultipleAxes 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.

9.3.3 Geometric Conventions

The coordinate system can be any system convention, it does not need to be an orthogonal system.

A PVT trajectory can be defined on any MultipleAxes group. There is no limit to the number of positioners belonging to that MultipleAxes group. It is also possible to define a PVT trajectory on a MultipleAxes group that contains only one positioner.

9.3.4 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:

$$\text{Jerk} = \frac{6 \cdot \left[DT \cdot (V_{in} + V_{out}) \quad 2 \cdot DX\right]}{DT^3}$$

Initial acceleration:

$$G_{in} = \frac{2 \cdot \left[3 \cdot DX - DT \cdot (2 \cdot V_{in} + V_{out})\right]}{DT^2}$$

Final acceleration:

$$G_{out} = \frac{2 \cdot \left[DT \cdot (V_{in} + 2 \cdot V_{out}) \quad 3 \cdot DX\right]}{DT^2}$$

Profile equation

Acceleration:

$$\text{Acc}(t) = G_{in} + \text{Jerk} \cdot t$$

Velocity:

$$\text{Vel}(t) = V_{in} + G_{in} \cdot t + \frac{\text{Jerk} \cdot t^2}{2}$$

Position:

$$\text{Pos}(t) = V_{in} \cdot t + \frac{G_{in} \cdot t^2}{2} + \frac{\text{Jerk} \cdot t^3}{6}$$

Here:

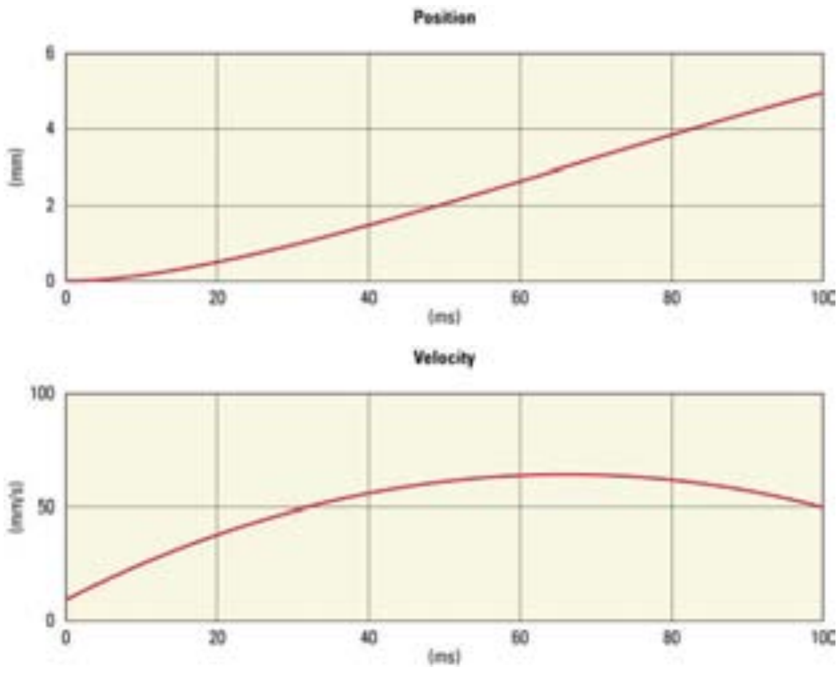
DT	is the segment duration in seconds
DX	is the displacement during DT
V_{in}	is the output velocity of the previous segment (which is equal to the input velocity of the current segment)
V_{out}	is the output velocity of the current segment.
t	is the time in seconds starting at 0 (entry of the current element) and ending at DT (end of the segment)

9.3.5 Influence of the Element Output Velocity to the Trajectory

The contour of each PVT trajectory element is not only influenced by the displacement, but also by the input and output velocities. As the user decides on these velocities, attention must be placed to 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.

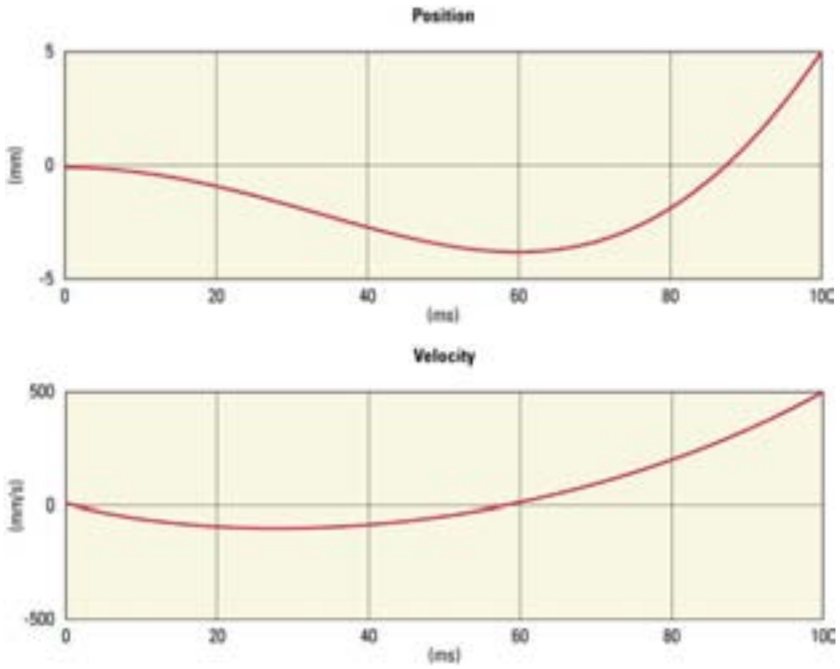


Figure 33: PVT trajectory element in execution: the comparison.

A PVT trajectory must have three parameters: position, velocity and time. With 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 requirements 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.

9.3.6 Trajectory File Description

The PVT trajectory is described in a file that is in the ..\public\trajectories folder of the XPS controller. 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 belonging to the MultipleAxes 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.

So the line format is as follow:

Data #1:	Element duration (seconds).
Data #2:	1st positioners displacement (units).
Data #3:	1st positioners output velocity (units/s).
Data #4:	2nd positioners displacement (units).
Data #5:	2nd positioners output velocity (units/s).

(And so on…)

NOTE

The first positioner is always defined first in the system.ini of the MultipleAxes group (see ActuatorInUse), the second positioner is always defined as second, and so on…

9.3.7 Trajectory File Example

Following is an example of a PVT trajectory defined on a MultipleAxes group that contains two positioners. The tabs are added for better readability:

```
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.

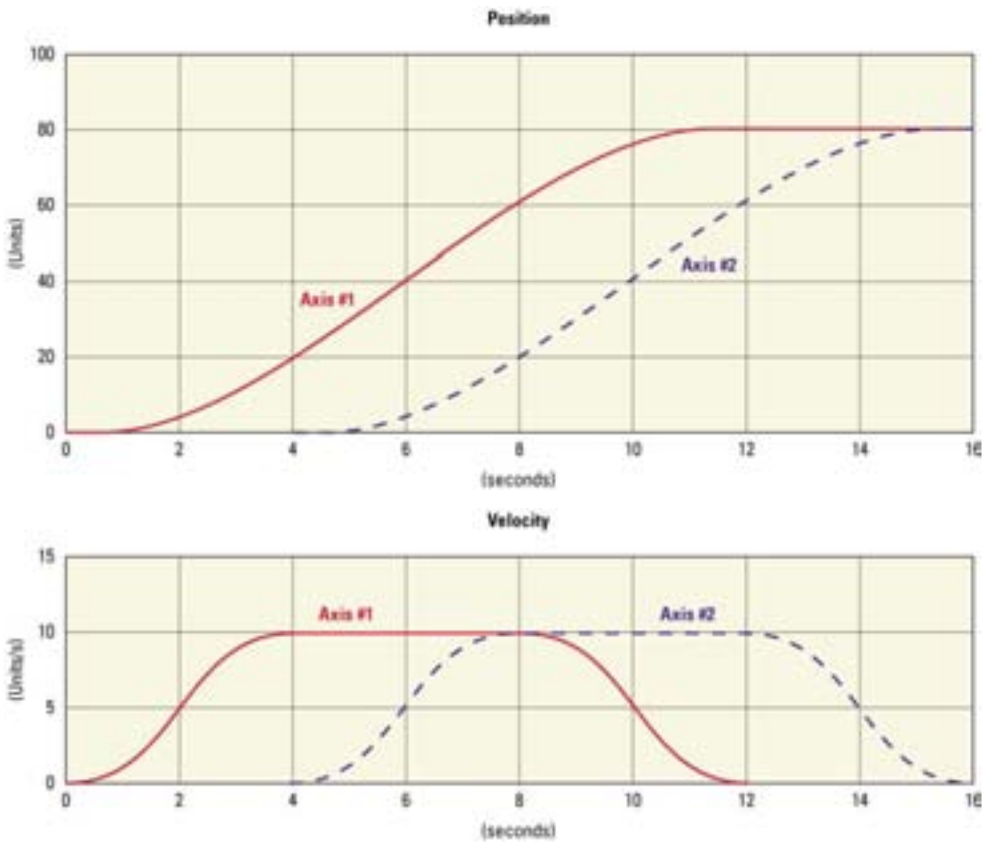


Figure 34: Executing trajectory data file with the PVT algorithm.

9.3.8 PVT Trajectory Verification and Execution

To verify and execute a PVT trajectory, there are four functions:

MultipleAxesPVTVerification(): Verifies a PVT trajectory data file.

MultipleAxesPVTVerificationResultGet(): Returns the last trajectory verification results, actuator by actuator. This function works only after a MultipleAxesPVTVerification().

MultipleAxesPVTExecution(): Executes a trajectory.

MultipleAxesPVTParametersGet(): Returns the trajectory current execution parameters. This function works only during the execution of the trajectory.

The function **MultipleAxesPVTVerification()** can be executed at any moment and is independent from the trajectory execution. This function is doing the following:

Checks the trajectory file for data coherence.

Simulates the trajectory to determine the positioner's requirements, which are the travel requirements in negative and positive direction and the maximum reached speed and acceleration for each positioner. This function helps determine whether the trajectory is executable.

If all is OK, it returns an "OK" (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 direction, the maximum reached speed and the maximum reached acceleration.

To execute a PVT trajectory, send the function **MultipleAxesPVTExecution()** while specifying the file name and the execution number. This function does not verify the trajectory 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 the execution, f because of bad data or because of a following error, the motion group will make an emergency stop and will go the disabled state.

Finally, the function MultipleAxesPVTParametersGet() returns the trajectory name and the number of the trajectory element that is currently in execution. This function returns an error if the trajectory is not in execution.

9.3.9 Examples of the Use of the functions

MultipleAxesPVTVerification (NGroup, PVT1.trj)

This function returns a 0 if the trajectory is executable.

MultipleAxesPVTVerificationResultGet (NGroup.1Positioner, *Name, *NegTravel, *PosTravel, *MaxSpeed, *MaxAcceleration)

This function returns the name of the trajectory checked with the last sent function MultipleAxesPVTVerification to the motion group NGroup (PVT1.trj) and the trajectory limits for the positioner NGroup.1Positioner. These trajectory limits are: the negative or left travel requirement, the positive or right travel requirement, the maximum reached speed and the maximum reached acceleration. Make sure that these trajectory limits (negative and positive travel requirements, speed and acceleration) are below the soft limits of the stages defined in the stages.ini file (section Travel: MinimumTargetPosition, MaximumTargetPosition and section Profiler: MaximumVelocity, MaximumAcceleration).

MultipleAxesPVTExecution (NGroup, PVT1.trj, 5)

Executes the trajectory PVT1.trj five (5) times.

MultipleAxesPVTParametersGet (NGroup, *FileName, *ElementNumber)

Returns the trajectory file name in execution (PVT1.trj) and the number of the current executed trajectory element.

10.0 Compensation

The XPS controller features different compensation methods that improve the performance of a motion system:

Backlash compensation: The setting of a backlash compensation improves the bi-directional repeatability and bi-directional accuracy of a motion device that has mechanical play. Backlash compensation is available with all positioners, but it is not compatible with all motion modes. When the backlash compensation is activated, the XPS controller adds a user-defined BacklashValue to the TargetPosition to calculate a new target position whenever reversing the direction of motion. This internally used new target position is then the basis for the calculations of the motion profiler. No modification of the actual target is done.

Linear error compensation: The linear error compensation helps to 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 (angles between feedback device and direction of motion). Linear error compensation is available with all positioners. It's value is defined in the stages.ini. When set different than zero, the encoder positions are compensated by this value. Linear error compensation can be used in parallel to any other compensation. For this reason, care must be taken to the effects when using linear error compensation in addition to other compensation methods.

Positioner mapping: In contrast to the linear error compensation, positioner mapping allows to compensate also for nonlinear error sources. Positioner mapping is done by sending a compensation table to the XPS controller and doing 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. For this reason, care must be taken to the effects when using linear error compensation in addition to other compensations.

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. For this reason, care must be taken to the effects when using for instance XY mapping and positioner mapping at the same time.

XYZ mapping: XYZ mapping is only available with XYZ groups. It compensates for all errors of a 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. For this reason, care must be taken to the effects when using for instance XYZ mapping and positioner mapping at the same time.

TargetPosition, SetpointPosition & CurrentPosition are accessible via function and Gathering (Data Collection).

SetpointVelocity, SetpointAcceleration & FollowingError are accessible via Gathering (Data Collection).

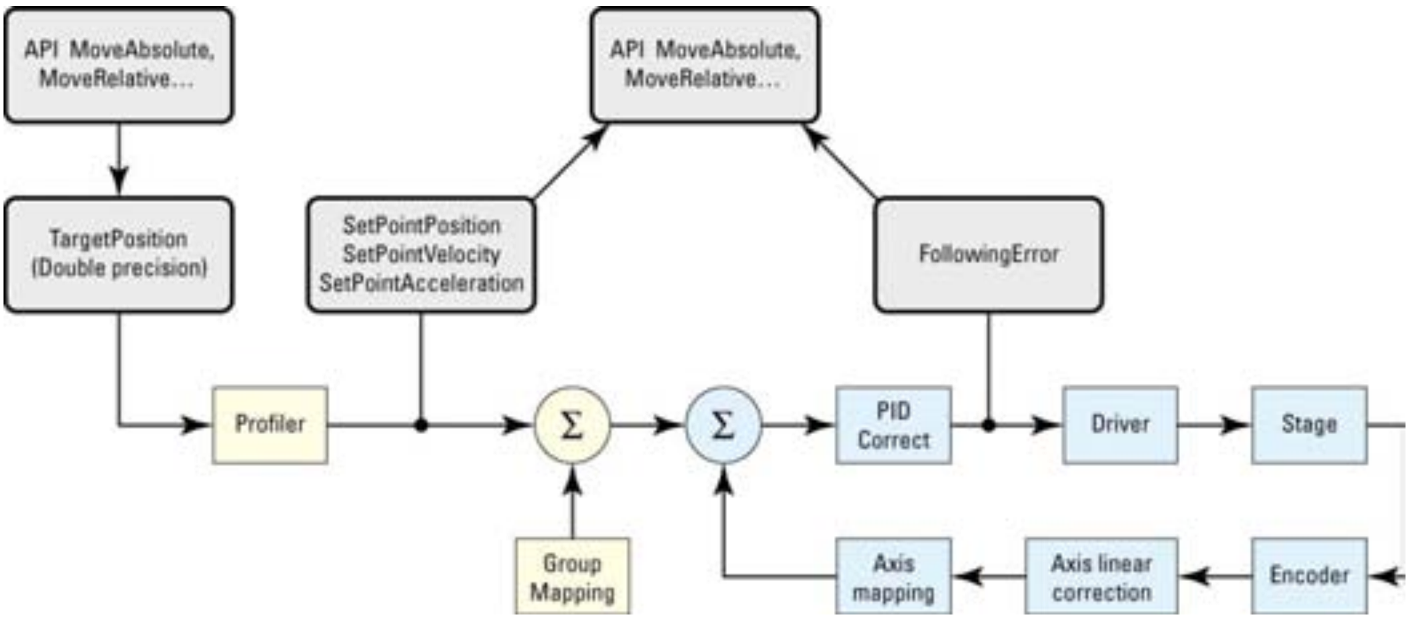

API MoveAbsolute, MoveRelative…
TargetPosition (Double precision)
SetPointPosition SetPointVelocity SetPointAcceleration
API MoveAbsolute, MoveRelative…
FollowingError
Profiler
Σ
Σ
PID Correct
Driver
Stage
Group Mapping
Axis mapping
Axis linear correction
Encoder

Figure 35: Definition of different positions for one actuator.

10.1 Backlash Compensation

Backlash compensation is available with all positioners, but works only under certain conditions:

The "HomeSearchSequenceType" in the stages.ini must be different than "CurrentPositionAsHome".

Backlash compensation is not compatible with positioner mapping. So for positioners with backlash compensation it is not allowed to have any 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 enabled backlash compensation.

After the above has been taken into consideration, a number of steps need to be done to enable the 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 the backlash compensation. To do so, you need to send the function **PositionerBacklashEnable()** while the motion group, to which this positioner belongs, must be disabled. To disable the backlash compensation (for instance to execute some jog motion or to use analog tracking), use the function **PositionerBacklashDisable().** The value for the backlash compensation can be changed at any time by 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 for the backlash and the backlash status ("enabled" or "disabled").

For set backlash to remain set after power down, the stages.ini file must be modified with the value desired.

Example

In the stages.ini file, set a value different from 0 in section Backlash:

```
;--- Backlash
Backlash = 5                    ; units
```

This example shows the sequence of functions to enable the 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)

10.2 Linear Error Correction

Linear error correction is available with all positioners and works in parallel to any other compensation. To use the linear error correction you need to set a value for "LinearErrorCorrection" in the stages.ini. When set, the corrected positions are calculated the following way:

Corrected position = HomePreset +
(EncoderPosition – HomePreset) x (1 + LinearEncoderCorrection/106)

The LinearEncoderCorrection is specified in ppm (parts per million). The correction is applied relatively 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 for the HomePreset.

Example

In the stages.ini file, set a value different from 0 in section Encoder, parameter LinearEncoderCorrection:

```
;--- Encoder
EncoderType =AquadB
EncoderResolution = 0.001       ; unit
LinearEncoderCorrection =5      ; ppm
```

10.3 Positioner Mapping

Positioner mapping allows correcting for any nonlinear errors of a positioner. Positioner mapping is available with all positioners and can be used in addition to other compensations, except for the backlash compensation.

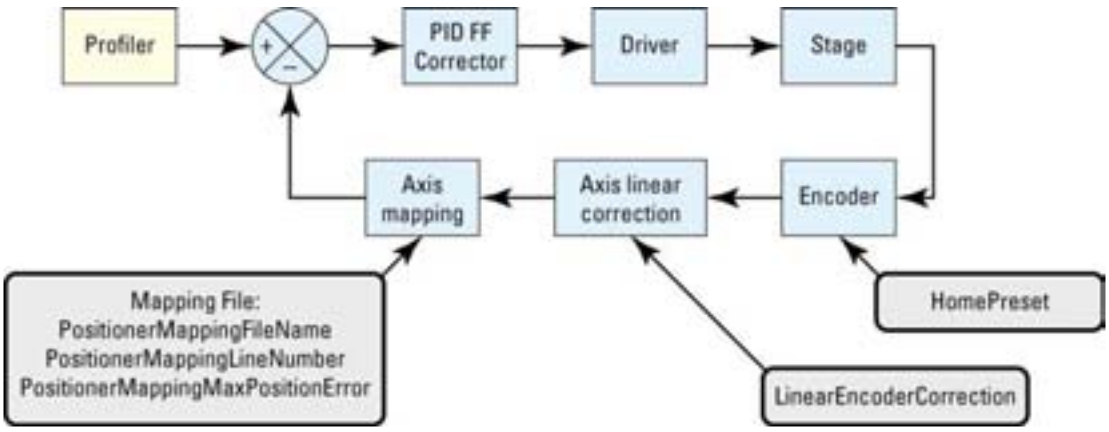


Figure 36: Positioner Mapping.

HomePreset: Encoder position value at the home position.

LinearEncoderCorrection: Value in ppm. Correction is given by

CorrectedPosition = HomePreset + (EncoderPosition – HomePreset)*(1+LinearCorrection/106).

Mapping file: Declaration of the mapping in the stages.ini file (part Positioner mapping).

The positioner mapping data gets 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 for 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 the positioner mapping, the mapping file must be in the ..\admin\config directory of the XPS controller and the following settings must be done in the stages.ini:

PositionerMappingFileName: Name of the mapping file.

PositionerMappingLineNumber: Number of lines of that file.

PositionerMappingMaxPositionError: Maximum absolute error in that file (any value larger than the actual largest value in the file will be accepted as well).

The PositionerMappingLineNumber and the

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 =-2                ; unit
HomePreset =0                            ; unit
MaximumTargetPosition =3                 ; unit
```

NOTE

These travel limits must be equal to or within the positioner limit positions of the mapping file (here +3 and -3).

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 the 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, the XPS controller is performing 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 functionality implemented within the controller to correct the errors of accuracy. Once activated, mapping is transparent for the user. The function GroupPositionCurrentGet doesn't return 0.24965 (0.25 - 0.00140*0.25/1) but 0.25.

10.4 XY Mapping

XY mapping is only available with XY groups. It compensation 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 to the cross-effects of using XY mapping and other compensations at the same time.

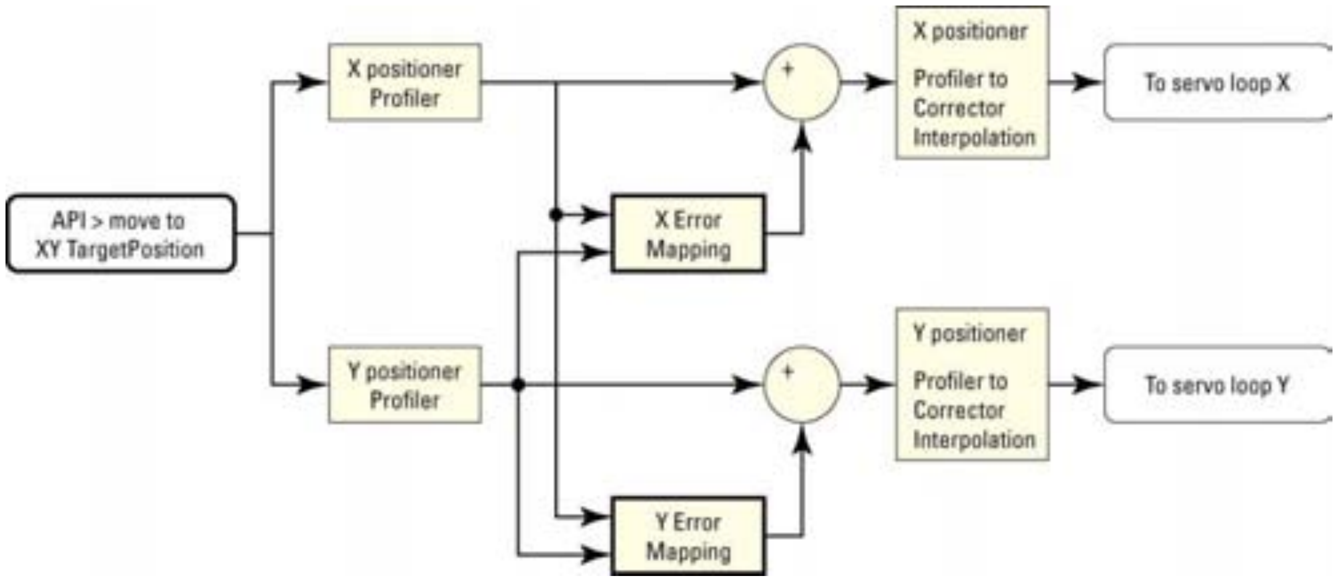
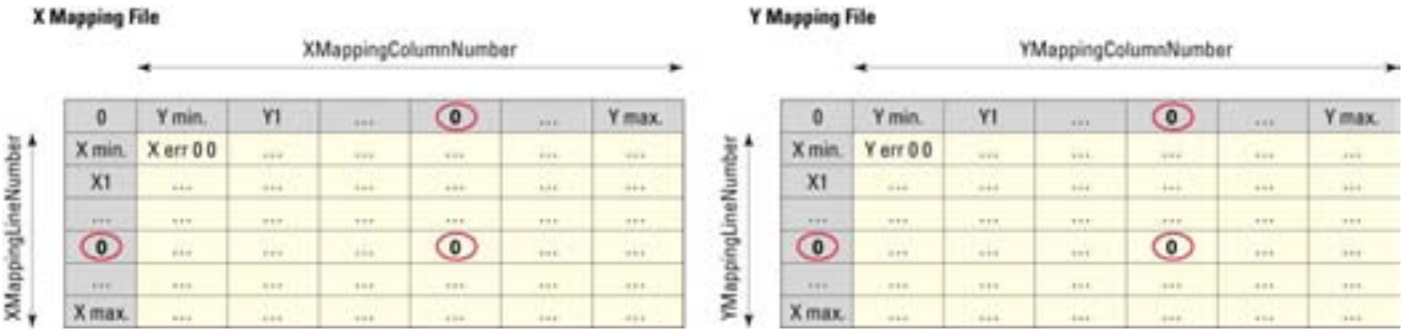

API > move to XY TargetPosition
X positioner Profiler
Y positioner Profiler
X Error Mapping
Y Error Mapping
+
+
X positioner Profiler to Corrector Interpolation
Y positioner Profiler to Corrector Interpolation
To servo loop X
To servo loop Y

Figure 37: XY Mapping

XY mapping is defined by 2 compensation tables, each for X and Y, in a text file format. In each of these files, the first column specifies the X positions, X being the first positioner of the XY group, and the first row, the Y positions. Each cell represents the error for that X,Y position. The first entry in that file must be 0 (zero). 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 tXY mapping has the advantage of being independent of the value for the HomePreset.

The following shows the structure of such mapping files:



X Mapping File

XMappingColumnNumber

XMappingLineNumber

0	Y min.	Y1	...	0	...	Y max.
X min.	X err 0 0	...	...	...	...	...
X1	...	...	...	...	...	...
...	...	...	...	...	...	...
0	...	...	...	0	...	...
...	...	...	...	...	...	...
X max.	...	...	...	...	...	...

Y Mapping File

YMappingColumnNumber

YMappingLineNumber

0	Y min.	Y1	...	0	...	Y max.
X min.	Y err 0 0	...	...	...	...	...
X1	...	...	...	...	...	...
...	...	...	...	...	...	...
0	...	...	...	0	...	...
...	...	...	...	...	...	...
X max.	...	...	...	...	...	...

Figure 38: XY Mapping Files.

NOTE

Error in X = Y = 0 must be 0. This value in the file corresponds to the HomePreset positions 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 done 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 (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, here +3 and –3, respectively.

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, part 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 follow: 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, the XPS controller is doing 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 functionality implemented within the XPS controller to correct the errors of accuracy. When mapping is activated, it is transparent for 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 XYZ Mapping

XYZ mapping is only available with XYZ groups. It compensation for all errors of an XYZ group at any position of that XYZ group. XYZ mapping can be used in addition to other compensations, including positioner mapping. Care must be taken to the cross-effects when using XYZ mapping and other compensations at the same time.

XYZ mapping is defined by 3 compensation files (compensation for X, Y and Z) in a 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 position; the first row of the first table contains the Y position; and the first cell of each table contains the Z position. The separator between the different data in each row is a tab. For better readability, inserting an empty line between successive tables is recommended, but is 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 referential for the XYZ mapping has the advantage of being independent of the value for the HomePreset.

Figure 39 shows the structure for the three mapping files for X, Y, and Z correction:

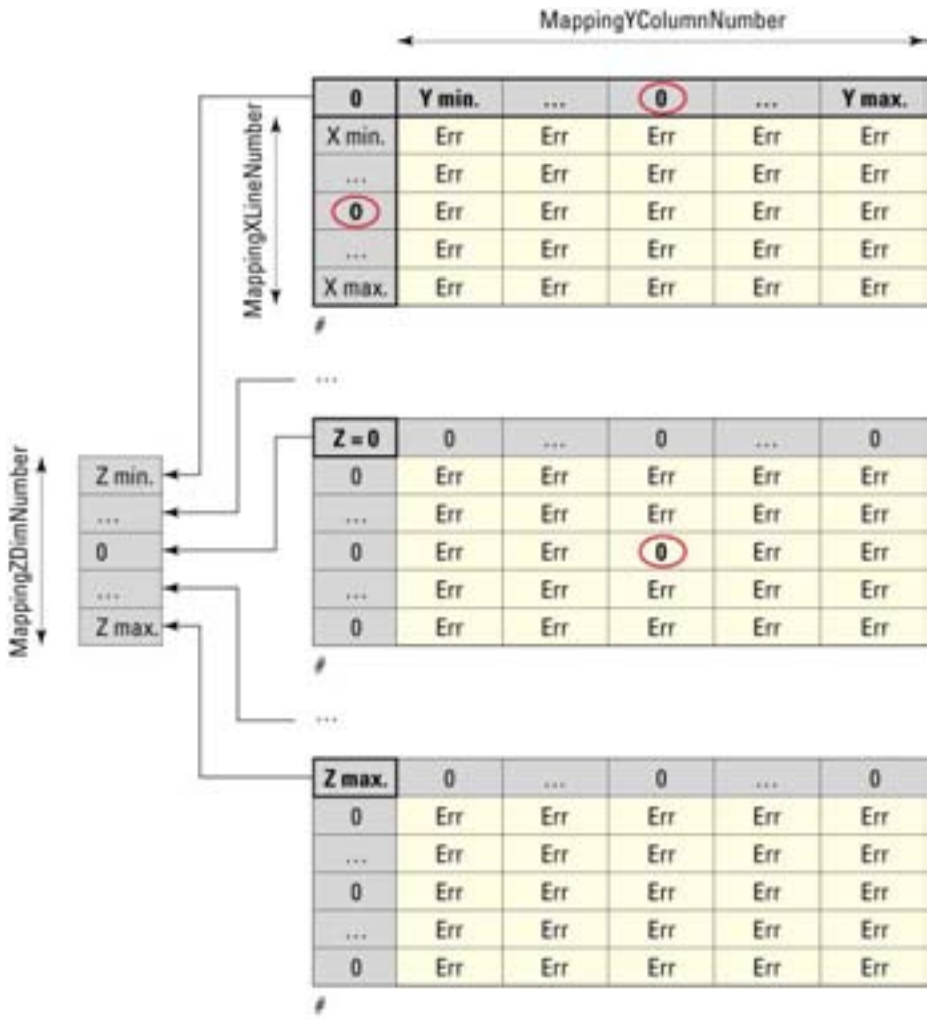


Figure 39: XYZ Mapping Files.

NOTE

Error in 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 matrix.

To activate the XYZ mapping, the mapping files must be in the ..\admin\config directory of the XPS controller and the following settings must be done in the system.ini:

XMappingFileName: Name of the mapping file.

XMappingXLineNumber: Total number of lines of each table including header.

XMappingYColumnNumber: Total number of columns.

XMappingZDimNumber: Number of successive tables.

XMappingMaxPositionError: Maximum absolute error in that file (Any value larger than the actual largest value in that file will be accepted as well).

YMappingFileName: Name of the mapping file.

YMappingXLineNumber: Total number of lines of each table including header.

YMappingYColumnNumber: Total number of columns.

YMappingZDimNumber: Number of successive tables.

YMappingMaxPositionError: Maximum absolute error in that file (Any value larger than the actual largest value in that file will be accepted as well).

ZMappingFileName: Name of the mapping file.

ZMappingXLineNumber: Total number of lines of each table including header.

ZMappingYColumnNumber: Total number of columns.

ZMappingZDimNumber: Number of successive tables.

ZMappingMaxPositionError: 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,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 shows examples for the X, Y, and Z mapping files for an XYZ mapping:

Matrix X: 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 Y: 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 Z: 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 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 (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 (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 (here +3 and –3).

In the system.ini file:

```
;--- Mapping XYZ
XMappingFileName = XYZMapping_X.txt
XMappingXLineNumber = 7
XMappingYColumnNumber = 7
XMappingZDimNumber = 3
XMappingMaxPositionError = 0.00787
YMappingFileName = XYZMapping_Y.txt
YMappingXLineNumber = 7
YMappingYColumnNumber = 7
YMappingZDimNumber = 3
YMappingMaxPositionError = 0.00534
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, part 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 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 follow: 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 data, the XPS controller is doing 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 functionality implemented inside the XPS controller to correct the errors of accuracy. But when mapping is activated, it is transparent for 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 3.

11.0 Event Triggers

Important Note to Users of XPS Firmware prior to Revision 1.6.0

With XPS firmware 1.6.0 Newport, has significantly expanded the use of the event triggers. However this extended use also requires changes to the event structure and the introduction of new functions that now all start with EventExtended. The old functions like EventAdd() etc. are still part of the new firmware, but no longer documented.

The XPS event triggers work similar to IF/THEN statements in programming. "If" the **event** occurs, "then" an **action** is triggered. Programmer's 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 at the same event. Furthermore, it is possible to link several events to an event configuration. In this case, all events have to happen at the same time to trigger the action(s). It is comparable to a logic AND between the different events.

Some events are singular 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 on programming event triggers, the user must be careful and consider possible unwanted effects. Some events might have 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 get illustrated in section 11.3 by some examples.

To trigger an action with an event, first the event and the associated action need to get configured using the functions **EventExtendedConfigurationTriggerSet()** and **EventExtendedConfigurationActionSet()**. Then, the event trigger needs to get activated using the function **EventExtendedStart()**. When activated, the XPS controller checks for the event 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 time of 100 μs or equal to the profiler cycle time of 400 μs. For events with duration, it means also that the same action is triggered each servo cycle, means every 100 μs, or each profiler cycle, means every 400 μs, as long as the event is happening.

Event triggers (and their associated action) get automatically removed after the event configuration has happened at least once and is no longer true anymore. The only exception is if the event configuration contains any of the permanent events "Always" or "Timer". In that 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 gets 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, means every 100 μs. For events that are defined as “Immediate”, an action is triggered once immediately (means during the very next servo cycle). For the events “Timer” an action is triggered immediately and every nth servo cycle. Here, “n” corresponds to the “FrequencyTicks” defined with the function **TimerSet()**. There are five different timers available that get 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, beside MotionDone) refer to the motion profiler of the XPS controller. The motion profiler runs at a frequency of 2.5 kHz, or every 400 μs. Thus, events triggered by the motion profiler have a resolution of 400 μs. Consequently, events with duration, such as MotionState, will trigger an action every 400 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 (.).

Actor				**Category**					**Event Name**	**Parameter**			
Group	**Positioner**	**GPIO**	**TimerX**	**SGamma**	**Jog**	**XYLineArc**	**Spline**	**PVT**		**1**	**2**	**3**	**4**
									Immediate				
									Always				
			v						Timer				
	v			v	v				MotionStart				
	v			v	v				MotionStop				
	v			v	v				MotionState				
	v			v	v				ConstantVelocityStart				
	v			v					ConstantVelocityEnd				
	v			v	v				ConstantVelocityState				
	v			v					ConstantAccelerationStart				
	v			v					ConstantAccelerationEnd				
	v			v					ConstantAccelerationState				
	v			v					ConstantDecelerationStart				
	v			v					ConstantDecelerationEnd				
	v			v					ConstantDecelerationState				
v						v	v	v	TrajectoryStart				
v						v	v	v	TrajectoryEnd				
v						v	v	v	TrajectoryState				
v						v	v	v	ElementNumberStart	Element #			
v						v	v	v	ElementNumberState	Element #			
	v								MotionDone				
v						v		v	TrajectoryPulse				
v						v		v	TrajectoryPulseOutputState				
		v							DILowHigh	Bit index			
		v							DIHighLow	Bit index			
		v							DIToggled	Bit index			
		v							ADCHighLimit	Value			
		v							ADCLowLimit	Value			
	v								PositionerError	Mask			
	v								PositionerHardwareStatus	Mask			

An event is entirely composed by:

[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 most parameters have no specific meaning. For the parameters with no meaning, it is still required to have a 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 is started, the event gets 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 started on the positioner MyGroup.MyPositioner. The addition of the event always has here just the effect that the event gets not removed anymore after the next motion has been started (see difference to first example above).

The exact meaning of the different events and event parameters is as follow:

Always:	Triggers an action ALWAYS, means 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, means once during the very next servo cycle: Event parameter 1 to 4 = 0 by default. **NOTE:** This event is EPHEMERAL.
Timer:	Triggers an action every nth servo cycle, where n gets 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 the motion done is reached. Event parameter 1 to 4 = 0 by default. For the exact definition of MotionDone, please refer to section 8.5.
ConstantVelocityStart:	Triggers an action when the constant velocity is reached. Event parameter 1 to 4 = 0 by default.
ConstantVelocityEnd:	Triggers an action when the constant velocity is finished. Event parameter 1 to 4 = 0 by default.
ConstantVelocityState:	Triggers an action during the constant velocity state. Event parameter 1 to 4 = 0 by default.

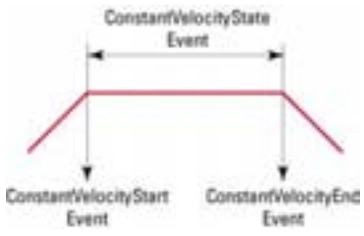


Figure 40: Constant Velocity Event.

ConstantAccelerationStart:	Triggers an action when the constant acceleration is reached. Event parameter 1 to 4 = 0 by default.
ConstantAccelerationEnd:	Triggers an action when the constant acceleration is finished. Event parameter 1 to 4 = 0 by default.
ConstantAccelerationState:	Triggers an action during the constant acceleration state. Event parameter 1 to 4 = 0 by default.

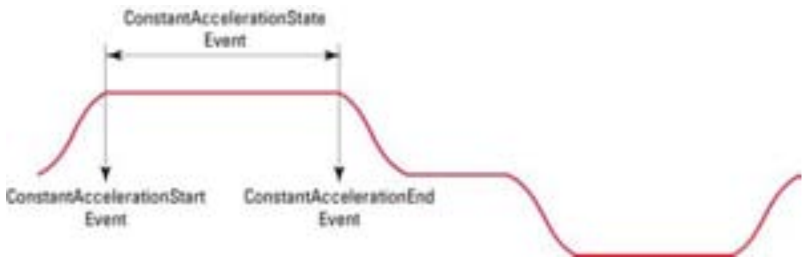


Figure 41: Constant Acceleration Event.

The same definition applies to ConstantDecelerationStart, ConstantDecelerationEnd and ConstantDecelerationState.

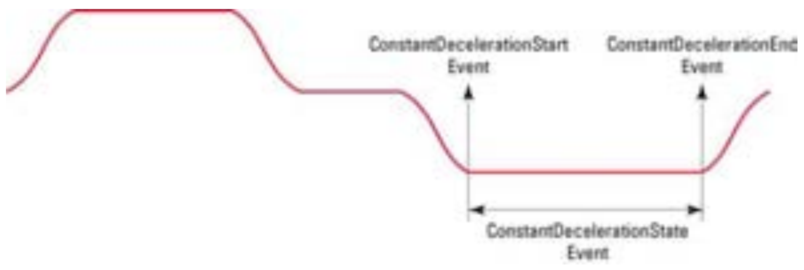

ConstantDecelerationStart Event
ConstantDecelerationEnd Event
ConstantDecelerationState Event

Figure 42: Constant Deceleration Event.

MotionStart:	Triggers an action when the motion starts. Event parameter 1 to 4 = 0 by default.
MotionEnd:	Trigger an action when the motion is ended. Event parameter 1 to 4 = 0 by default. Note, MotionEnd refers to the end of the theoretic motion which is not the same as MotionDone depending on the definition (see also section 8.5).
MotionState:	Triggers an action during the motion. Event parameter 1 to 4 = 0 by default.

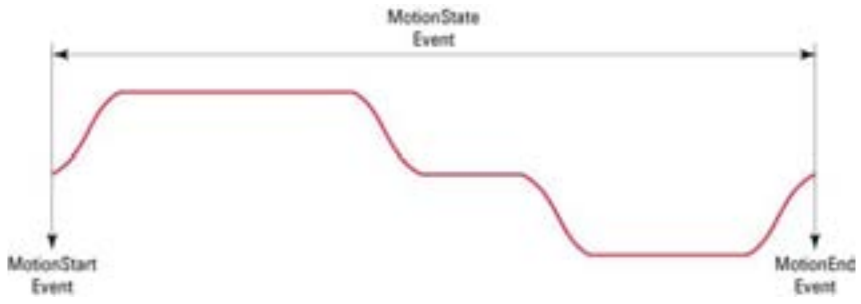


Figure 43: Motion Event.

There are also several trajectory events that can be defined:

TrajectoryStart:	Triggers an action when the trajectory is started. Event parameter 1 to 4 = 0 by default.
TrajectoryEnd:	Triggers an action when the trajectory is stopped. Event parameter 1 to 4 = 0 by default.
TrajectoryState:	Triggers an action during the trajectory state. Event parameter 1 to 4 = 0 by default.

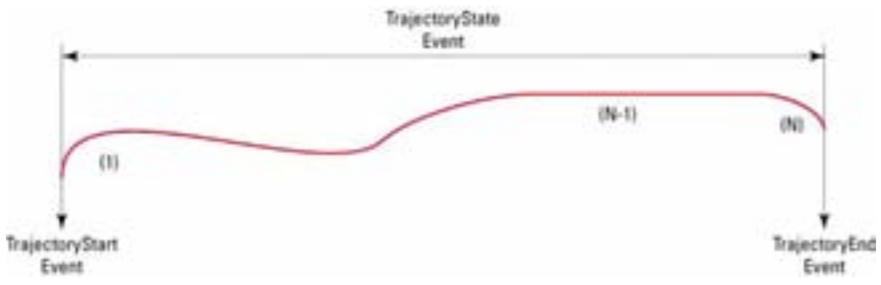


Figure 44: Trajectory Event.

ElementNumberStart:	Triggers an action when the trajectory element number is 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.

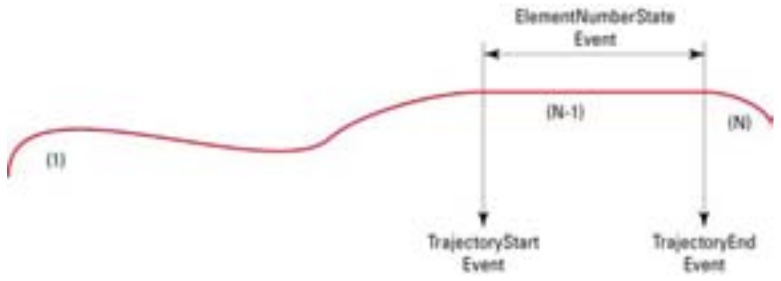


Figure 45: Element Number Event.

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.

Note, that any event trigger gets automatically removed when it has happened at least once and not anymore on the next servo or profiler cycle. Hence, in order to trigger an action at every TrajectoryPulse it is required to link that event to the event Always (see also section 11.4: "Examples").

TrajectoryPulseOutputState: Triggers an action during the trajectory pulse output state, means between the start and the end definitions of the trajectory output pulses (see sections 13.4 and 13.5: "Triggers on Trajectories" for details). All event parameters are 0 by default.

DILowHigh: Triggers an action when the digital input bit switches from low state to 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 high state to 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.

PositionerError: Triggers an action when the current positioner error applied with the error mask results in a value different than zero. The first event parameter specifies the error mask in a decimal format. The other event parameters are 0 by default.

Code (Hexa)	Bit #	Decimal	Positioner error description
0			No error
0x00000001	0	1	General inhibition detected
0x00000002	1	2	Fatal following error detected
0x00000004	2	4	Home search time out
0x00000008	3	8	Motion done time out
0x00000010	4	16	Requested position exceed travel limits in trajectory or slave mode
0x00000020	5	32	Requested velocity exceed maximum value in trajectory or slave mode
0x00000040	6	64	Requested acceleration exceed max value in trajectory or slave mode
0x00000100	8	256	Minus end of course activated
0x00000200	9	512	Plus end of course activated
0x00000400	10	1024	Minus end of run glitch
0x00000800	11	2048	Plus end of run glitch
0x00001000	12	4096	Encoder quadrature error
0x00002000	13	8192	Encoder frequency and coherence error
0x00010000	16	65536	Hard interpolator encoder error
0x00020000	17	131072	Hard interpolator encoder quadrature error
0x00100000	20	1048576	First driver in fault
0x00200000	21	2097152	Second driver in fault

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 different than zero. The first event parameter specifies the status mask in a decimal format. The other event parameters are 0 by default.

Code (Hexa)	Bit #	Decimal	Hardware status description
0x00000001	0	1	General inhibition detected
0x00000004	2	4	ZM high level
0x00000100	8	256	Minus end of run activated
0x00000200	9	512	Plus end of run activated
0x00000400	10	1024	Minus end of run glitch
0x00000800	11	2048	Plus end of run glitch
0x00001000	12	4096	Encoder quadrature error
0x00002000	13	8192	Encoder frequency or coherence error
0x00010000	16	65536	Hard interpolator encoder error
0x00020000	17	131072	Hard interpolator encoder quadrature error
0x00100000	20	1048576	First driver in fault
0x00200000	21	2097152	Second driver in fault
0x00400000	22	4194304	First driver powered on
0x00800000	23	8388608	Second driver powered on

Example

EventExtendedConfigurationTriggerSet (MyGroup.MyPositioner.PositionerHardwareStatus, 768, 0, 0, 0)

This event happens when the positioner MyGroup.MyPositioner has either a plus end of run or a minus end of run detected.

11.2 Actions

There are several actions that can be triggered by the events listed above. 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 functions **EventExtendedConfigurationActionSet()**, similar to how it is done with events.

Actor				**Action Name**	**Parameter**			
Group	**Positioner**	**GPIO**	**TimerX**		**1**	**2**	**3**	**4**
		v		DOToggle	Mask			
		v		DOPulse	Mask			
		v		DOSet	Mask	Value		
		v		DACSet.CurrentPosition	Positioner name	Gain	Offset	
		v		DACSet.CurrentVelocity	Positioner name	Gain	Offset	
		v		DACSet.SetpointPosition	Positioner name	Gain	Offset	
		v		DACSet.SetpointVelocity	Positioner name	Gain	Offset	
		v		DACSet.SetpointAcceleration	Positioner name	Gain	Offset	
				ExecuteTCLScript	TCL file name	Task name	Arguments	
				KillTCLScript	Task name			
				GatheringOneData				
				GatheringRun	Nb of points	Divisor		
				GatheringRunAppend				
				GatheringStop				
				ExternalGatheringRun	Nb of points	Divisor		
v				MoveAbort				

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 the constant velocity state and to set it back 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 back to its original value at the event ConstantVelocityEnd. The same effect can NOT be achieved with the sole use of the event ConstantVelocityState.

An action is entirely composed by:

[Actor].Action Name, Parameter1, Parameter2, Parameter3, Parameter4.

Not all action names have a preceding actor, but all actions have four parameters. Even though not all actions have a specific meaning for all four parameters, it is still required to have a zero (0) as 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 mask 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 start the execution of the TCL script "Example.tcl". The task name is 1 and the TCL script has no arguments (a zero for the third parameter means no arguments).

EventExtendedConfigurationActionSet (GatheringRun, 1000, 10, 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 10th servo cycle, means one data point every 10/10000 s = 1 ms.

It is also possible to trigger several actions at the same event. To do so, just define another action in the SAME function. Several actions are separated by a comma (,).

<u>**Example:**</u>

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 action and action parameters is as follow:

DOToggle: This action is used to reverse the value of one or many bits on 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 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, the bit #3 will be toggled.
Action Parameter #2 to #4	These parameters must be 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 each cycle of the motion profiler (or every 400 μs) 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 must be 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 the 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. So 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 must be 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 the offset are used to calibrate the output. This action makes most sense with events that have some duration (always, MotionState, ElementNumberState, etc.) as the analog output will be updated each servo cycle or each profiler cycle as long as the event lasts. When used with events that have no duration (like MotionStart or MotionEnd), the analog output gets only updated once and this value is hold until the next change.

Action Parameter #1 – Positioner Name	This parameter defines the name of the positioner which 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, then the output voltage is 10 V.
Action Parameter #3 – Offset	The offset value is used to correct for any voltage that may be present on the Analog output.

Analog output = Position value * gain + offset

Action parameter #4	This parameter must be 0 by default.

DACSet.CurrentVelocity and **DACSet.SetpointVelocity:** This action sets a voltage on the Analog output in relation 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 some duration (Always, MotionState, ElementNumberState, etc.) as the analog output will be updated each servo cycle or each profiler cycle as long as the event lasts. When used with events that have no duration (like MotionStart or MotionEnd), the analog output gets only updated once and this value is hold until its next change.

Action Parameter #1 – Positioner Name	This parameter defines the name of the positioner which 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, then the output voltage is 10 V.
Action Parameter #3 – Offset	The offset value is used to correct for any voltage that may be present on 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 some duration (Always, MotionState, ElementNumberState, etc.) as the analog output will be updated each servo cycle or each profiler cycle as long as the event lasts. When used with events that have no duration (like MotionStart or MotionEnd), the analog output gets only updated once and holds this value until its next change.

Action Parameter #1 – Positioner Name	This parameter defines the name of the positioner which SetpointAcceleration is used to output on 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/s2 then the output voltage will be 10 V.
Action Parameter #3 – Offset	The offset value is used to correct for any voltage that may be present on the Analog output.

Analog output = SetpointAcceleration value * gain + offset

Action parameter #4	This parameter must be 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 on 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 different or even the same TCL scripts can run simultaneously, the TCL Task Name is used to track individual TCL programs. For example, the TCL Task Name allows stopping a particular program without stopping all other TCL programs that run simultaneously.
Action Parameter #3 – TCL Arguments List	The Argument list is used to run the TCL scripts with input parameters. For the argument parameter, any input can be given (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 must be 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 different or even the same TCL scripts can run simultaneously, the TCL Task Name is used to track individual TCL programs.
Action parameter #2 to #4	These parameters must be 0 by default.

GatheringOneData: This action acquires one data as defined by the function GatheringConfigurationSet. Different than 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, you first need to launch the function GatheringReset, which deletes the current gathering file from memory.

Action parameter #1 to #4	These parameters must be 0 by default.

GatheringRun: This action starts an internal 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 with 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 the frequency for the gathering in relation to the servo frequency of the system (10 kHz). This parameter has to be an integer and greater or equal to 1. For instance, if the parameter is set to 10, then the gathering will take place every 10th servo cycle or at a rate of 1 kHz (10 kHz/10) or at every 1 msec.
Action Parameter #3 and #4	These parameters must be 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 must be 0 by default.

GatheringStop: This action halts a data gathering previously launched by the action GatheringStart. Use the action GatheringRunAppend to continue the data gathering. Please note, that the action GatheringStop does not automatically store the gathered data from the buffer to the flash disk of the controller. For doing so, please use the function GatheringStopAndSave. For more details about data gathering, please refer to chapter 12: “Data Gathering”.

Action parameter #1 to #4	These parameters must be 0 by default.

ExternalGatheringRun: This action starts an external gathering. It requires that an external 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 with 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 trigger input signal at which the gathering will take place. This parameter has to be an integer and greater 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 must be 0 by default.

For further details on data gathering see chapter 12: “Data Gathering”.

MoveAbort: This action allows to stop (abort) a motion on an event. It is similar to sending a MoveAbort() function on the event. After breaking, the group is in the READY state.

Action Parameter #1 to #4	These parameters must be 0 by default.

11.3 Functions

The following functions are related to the event triggers:

EventExtendedConfigurationTriggerSet (): This function configures one or several events. In 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 get 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 each servo cycle (or 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 of 100 μs or equal to the profiler cycle time of 400 μs for motion related events. For events with duration, it means also that the same action is triggered each servo cycle, means every 100 μs, or each profiler cycle, which means every 400 μs as long as the event is happening.

Event triggers (and their associated action) get automatically removed after the event configuration has happened at least once and is no longer true anymore. The only exception is if the event configuration contains any of the permanent events “Always” or “Trigger”. In that 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 to the defined event identifier.

EventExtendedGet (): This function returns the event configuration and the action configuration associated to 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.

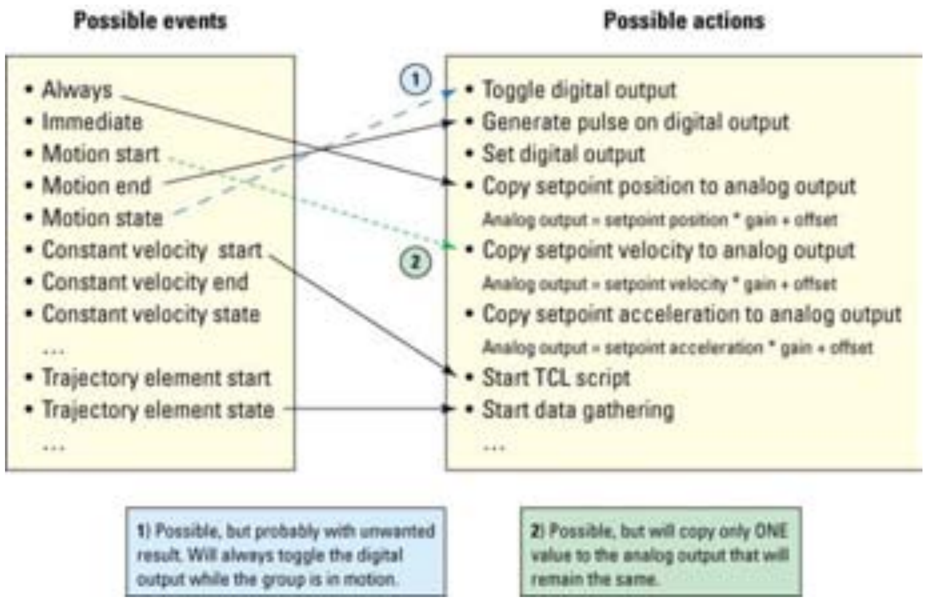

Possible events
• Always
• Immediate
• Motion start
• Motion end
• Motion state
• Constant velocity start
• Constant velocity end
• Constant velocity state
...
• Trajectory element start
• Trajectory element state
...
Possible actions
1
2
• Toggle digital output
• Generate pulse on digital output
• Set 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) 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 46: Possible Events.

<u>Examples</u>

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 is 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 than in the previous example, here the concatenate with the event "Always" avoids that the event trigger gets 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.

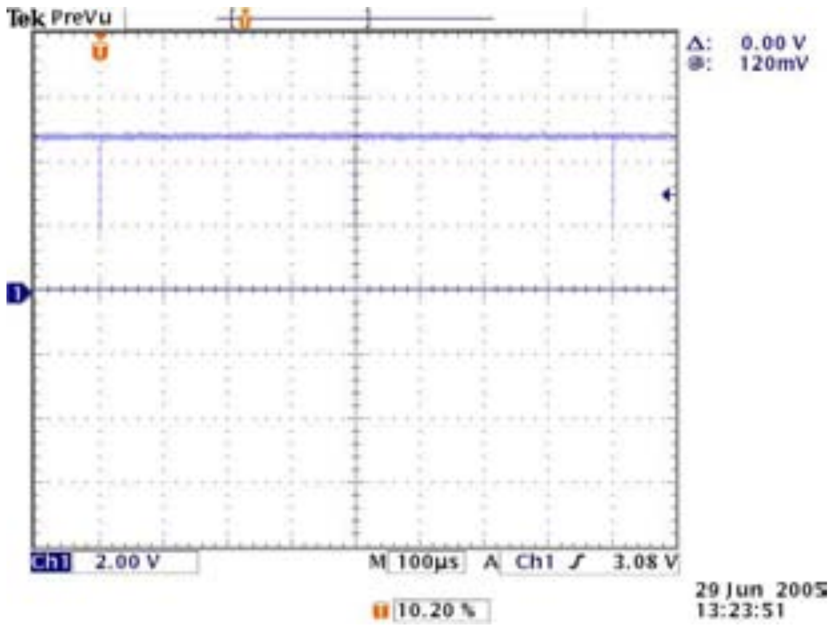
4. **EventExtendedConfigurationTriggerSet (G1.P1.SGamma.ConstantVelocityState, 0, 0, 0, 0)**

   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 on the digital output on connector number 1 every cycle of the motion profiler (Note: 255 = 11111111). The cycle time of the motion profiler is 400 μs, so pulses are generated every 400 μs (see picture below).



5. **EventExtendedConfigurationTriggerSet (Always, 0, 0, 0, 0)**

   EventExtendedConfigurationActionSet (GPIO2.DAC1.DACSet.SetpointPosition, 0.1, -10, 0, 0

   GPIO2.DAC2.DACSet.SetpointVelocity, 0.5, 0, 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 when the stage is at the position 0 unit, a voltage of -10 V will be outputted. When the stage is at the position 10 units, a voltage of -9V will be outputted.
   Here, the event "Always" makes that these values will get updated every servo cycle, means every 0.1 ms. If instead of the event "Always" the event "Immediate" will be used, only the most recent values will be outputted and kept. If instead of the event "Always" a motion related event such as MotionState will be used, the update will only happen every profiler cycle, or every 0.4 ms.

6. **TimerSet(Timer1,10000)**

   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 10 000th servo cycle, or at one second. Hence, in this example, every second all bits on 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, first the generation of an output pulse every one second between the 2nd and the 20th element on the next PVT trajectory executed on the group G1 is defined (function MultipleAxisPVTPulseOutputSet). Then, the 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, the concatenate of 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 gets automatically removed when once happened and not happening anymore on 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)

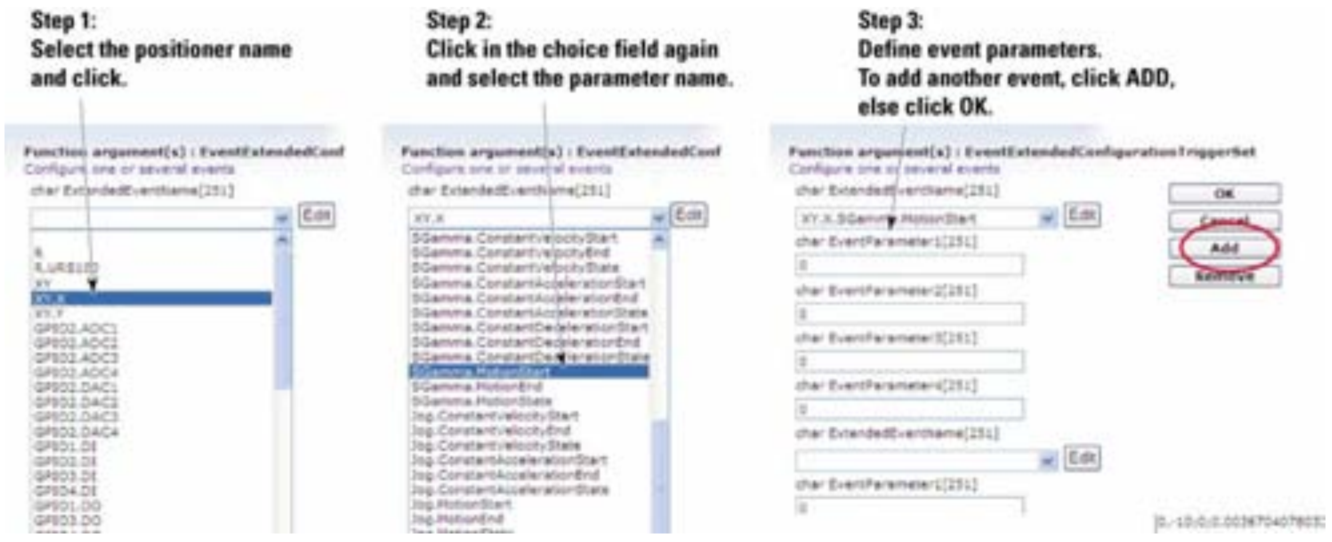
   GatheringStopAndSave()

   In this example, an internal data gathering of 20 data points every 0.1 second (every 1000th servo cycle) is launched with the start of the next motion of the positioner G1.P1. The type of data that gets 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, you need to send the function GatheringStopAndSave(). The GatheringRun deletes the current data file in the 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 that you store your valuable data files under different name after the 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 button. Select the next parameter as described above.



Step 1:
Select the positioner name and click.

Function argument(s) : EventExtendedConf
Configure one or several events
char ExtendedEventName[251]

S
S.URS100
XY
XY.X
XY.Y
GPIO2.ADC1
GPIO2.ADC2
GPIO2.ADC3
GPIO2.ADC4
GPIO2.DAC1
GPIO2.DAC2
GPIO2.DAC3
GPIO2.DAC4
GPIO1.DI
GPIO2.DI
GPIO3.DI
GPIO4.DI
GPIO1.DO
GPIO3.DO

Step 2:
Click in the choice field again and select the parameter name.

Function argument(s) : EventExtendedConf
Configure one or several events
char ExtendedEventName[251]
XY.X

SGamma.ConstantVelocityStart
SGamma.ConstantVelocityEnd
SGamma.ConstantVelocityState
SGamma.ConstantAccelerationStart
SGamma.ConstantAccelerationEnd
SGamma.ConstantAccelerationState
SGamma.ConstantDecelerationStart
SGamma.ConstantDecelerationEnd
SGamma.ConstantDecelerationState
SGamma.MotionStart
SGamma.MotionEnd
SGamma.MotionState
Jog.ConstantVelocityStart
Jog.ConstantVelocityEnd
Jog.ConstantVelocityState
Jog.ConstantAccelerationStart
Jog.ConstantAccelerationEnd
Jog.ConstantAccelerationState
Jog.MotionStart
Jog.MotionEnd

Step 3:
Define event parameters.
To add another event, click ADD, else click OK.

Function argument(s) : EventExtendedConfigurationTriggerSet
Configure one or several events
char ExtendedEventName[251]
XY.X.SGamma.MotionStart
char EventParameter1[251]
0
char EventParameter2[251]
0
char EventParameter3[251]
0
char EventParameter4[251]
0
char ExtendedEventName[251]

char EventParameter1[251]
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 every nth servo cycle.
2. Event based (internal) data gathering. With this method one data set is gathered on 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 every nth external trigger input (see also chapter 13.0: “Output Triggers“).

With method 1, 2, and 3 we are also referring to an internal or servo cycle synchronous data gathering. With the trigger based data gathering we are also referring to an external data gathering as the event that triggers the data gathering, the receive 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 common file called gathering.dat. The trigger based (external) data gathering stores the data in 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).

During the 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 utilities, 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..\Admin\Public folder of the XPS controller under the name Gathering.dat (with function GatheringStopAndSave() for an internal gathering) or GatheringExternal.dat (with function GatheringExternalStopAndSave() for an external gathering).

CAUTION

The functions GatheringStopAndSave() and GatheringExternalStopAndSave() overwrite any older files with the same name in the ..\Admin\Public folder. After a data gathering, it is required to rename or better, to relocate valid data files using an ftp link to the XPS controller (see also chapter 5: “FTP connection”).

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 = 0.0001 s), the second line contains the names of the data type(s) and the other lines contain the acquired data. A prototype of a sample file is shown below.

Gathering.dat

```
SamplePeriod         0                    0
GatheringTypeA  GatheringTypeB  GatheringTypeC
ValueA1              ValueB1              ValueC1
ValueA2              ValueB2              ValueC2
…                    …                    …
ValueAN              ValueBN              ValueCN
```

12.1 Time Based (Internal) Data Gathering

The data for the time based gathering get latched by an internal interrupt related to the servo cycle of the system (10 kHz). The function GatheringConfigurationSet() defines which type of data 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

PositonerName.CurrentAcceleration

PositionerName.SetpointAcceleration

PositionerName.CorrectorOutput

GPIO (ADC, DAC, DI, DO) See Programmer's Guide 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 whereas the current values refer to the actual or real values of position, velocity and acceleration.

For gathering information from the secondary positioner of a gantry, append ".SecondaryPositioner" to the positioner name. Example:

PositionerName.SecondaryPositioner.FollowingError

For details about gantry configurations, see chapter 4.8.

It is possible to start the gathering either by function call or at an event. The following sequence of functions is used for a time based data gathering started by 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 which triggers the action, for instance a GroupMoveRelative().

When all data is gathered, use the function **Gathering StopAndSave()** to save the data from the buffer to the flash disk of the XPS controller.

Other functions associated with internal Gathering are:

GatheringConfigurationGet()

GatheringCurrentNumberGet()

GatheringDataGet()

GatheringDataMultipleLinesGet()

GatheringStop()

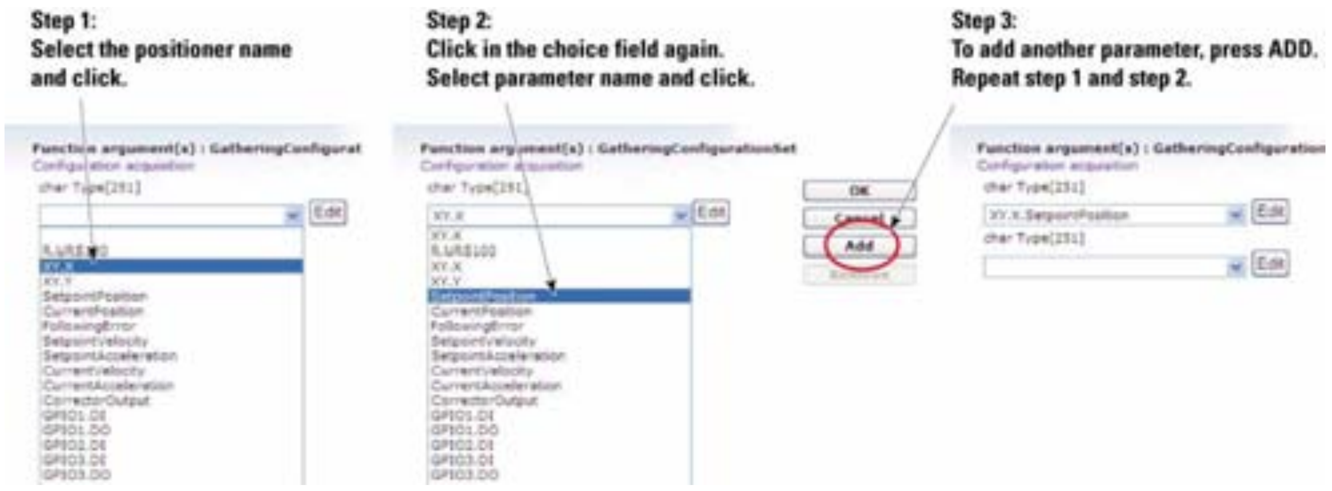
GatheringRunAppend()

See Programmer's Manual for details on 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 above.



Step 1:
Select the positioner name and click.

Step 2:
Click in the choice field again.
Select parameter name and click.

Step 3:
To add another parameter, press ADD.
Repeat step 1 and step 2.

<u>Example 1</u>

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,5000,10,0,0)

EventExtendedStart()

GroupMoveRelative(XY.X, 50)

GatheringStopAndSave()

In this example, a gathering is started when the positioner XY.X starts its next motion using the Sgamma profiler, for instance by the functions GroupMoveRelative() or GroupMoveAbsolute(). The types of data being collected are the Setpoint Position, Current Velocity and Setpoint Acceleration for the positioner XY.X. A total of 5000 data sets is collected, one data point every 10th servo cycles, or one data point every 10/10000 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 function call.

GroupInitialize(X)

GroupHomeSearch(X)

GatheringConfigurationSet(X.X.SetpointPosition, X.X.FollowingError)

GatheringRun (5000,10)

GroupMoveRelative (X, 10)

GatheringStop ()

GatheringStopAndSave ()

In this example, the gathering is started by function call. The SetpointPosition and FollowingError of the positioner XY.X are gathered at a rate of 1 kHz (every 10th servo cycle, 10 kHz servo cycle rate). The data gathering is stopped after the relative move is completed.

The 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()** has be to used to save the data to a file.

It is also possible to halt a 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 the gathering. For details, see chapter 11.0: “Event Triggers“.

Note

The function GatheringRun() starts always 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 the event based gathering are the same as for the time based and the 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.

For gathering information from the secondary positioner of a gantry, append ".SecondaryPositioner" to the positioner name. Example:

PositionerName.SecondaryPositioner.FollowingError

For details about gantry configurations, see chapter 4.8.

The following sequence of functions is used for an 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()

Please refer to the programmer's manual for details.

<u>Example 1</u>

GatheringReset()

Deletes gathering buffer.

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()

The data gathering starts when the value of the GPIO2.ADC1 exceeds 5 Volts. One set of data will be gathered each servo cycle or every 100 μs (as the event is checked each servo cycle). The data gathering automatically stops when the value of the GPIO2.ADC1 falls below 5V again, as the event is automatically removed then (see chapter 11.0: "Event Triggers" for details).

<u>Example 2</u>

TimerSet(Timer1, 10)

Sets the timer 1 to 10 servo ticks, means every 1 ms.

GatheringReset()

Deletes gathering buffer.

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 than the previous example, here the event ADCHighLimit is linked to the event Timer1. This has two effects. First, the event gets permanent as the event timer is permanent. Second, one set of data is gathered only every 10 ms (combination of events must be true). For details on the event definition, please see chapter 11.0: "Event Triggers".

As a result, in this example, one set of data is gathered every 10 ms whenever the value of the GPIO2.ADC1 exceeds 5 Volts.

Example 3

TimerSet(Timer1, 10)

Sets the timer 1 to 10 servo ticks, means every 1 ms.

GatheringReset()

Deletes gathering buffer.

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 10 ms. In contrast to the time based gathering, which allows programming of a similar function, here, the data gathering will automatically stop with the end of the trajectory. Also, it is not need to define the total number of data sets that will be gathered.

12.3 Function-Based (Internal) Data Gathering

The function based gathering provides a method to gather one set of data by a function. It uses the same file as the time based and the Event based data gathering, see chapters 13.1 and 13.4 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 chapter 12.1 and 12.2 for details.

Example

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 receive of an external trigger input (TRIG IN connector at the XPS, see section 24.0 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 a rate of 10 kHz and the XPS will store the most recent value. Hence, the acquired analog input data might be up to 100 µs old.

NOTE

There must be a minimum time of 100 µs 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 that can be collected:

PositionerName.ExternalLatchPosition and
PositionerName.SecondaryPositioner.ExternalLatchPosition
(for secondary positioners of gantries, see chapter 4.8 for details).

These positions refer to the uncorrected encoder position, means 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 1Vpp 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 GPIO.ADC4
(referring to the 4 analog input channels on the GPIO2)

The following sequence of functions is used for a trigger based data gathering:

GatheringExternalConfigurationSet()

EventExtendedConfigurationTriggerSet()

EventExtendedConfigurationActionSet()

EventExtendedStart()

Other functions associated with the event based gathering are:

GatheringConfigurationGet()

GatheringCurrentNumberGet()

GatheringExternalDataGet()

Please refer to the Programmer's Manual for details.

<u>**Example**</u>

GatheringExternalConfigurationSet(XY.X.ExternalLatchPosition, GPIO2.ADC1)

EventExtendedConfigurationTriggerSet(Immediate,0,0,0,0)

EventExtendedConfigurationActionSet(ExternalGatheringRun,100,2,0,0)

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 second trigger input. The 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 the internal data gathering.

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 per axis, see Appendix E, 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 then, 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 either provided 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 then, 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.

In addition and independent from the above, the XPS controller can output distance spaced pulses on line-arc trajectories and time spaced pulses on PVT trajectories. In these cases the distances/time intervals are checked on the servo cycle and a resolution of 100 μs is provided.

13.1 Distance Spaced Pulses (PCO – Position Compare Output)

In the 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 200 nsec by default and can be modified using the function **PositionerPositionComparePulseParametersSet (PositionerName, PCOPulseWidth, EncoderSettlingTime)**. Possible values for PCOPulseWidth are: 0.2 (default), 1, 2.5 and 10 (μs). Please note, that only the falling edge of the trigger pulse is precise and only this edge should be used for synchronization irrespactable from the PCOPulseWidth setting. Note also, that the duration of the pulse detected by your electronics may be longer depending on the time constant of your RC circuit. Successive trigger pulses should have a minimum time lag equivalent to the PCOPulseWidth time times two.

The second parameter, EncoderSettlingTime applies a filter to the encoder signals for the trigger pulse generation. Possible values are: 0.075 (default), 1, 4, 12 (μs). The setting of this EncoderSettlingTime should be done in relation to the application, in particular speed and encoder resolution, and the encoder/position noise. For most applications, the default value works fine. At very low speed, with high encoder resolution, and significant encoder/position noise, however, it may be possible that additional trigger pulses are generated where no trigger pulse should be generated from the application. In these cases, a higher value setting for the EncoderSettlingTime could avoid these unwanted extra pulses. The value for the EncoderSettlingTime, however, should not exceed the value for the Encoder resolution divided by the speed. Please note also, that the EncoderSettlingTime adds a nominal delay between the encoder transition and the trigger pulse.

Example

With XM stages and hardware interpolator set to 200 (see function PositionerHardInterpolatorFactorSet ()) the resolution of the trigger pulses is 20 nm (4 μm encoder scale pitch / 200). At continuous speed motion with 20 μm/s speed, the nominal time between successive encoder counts is 1 ms (20 nm / 20 μm/s). In a not optimum environment of the XM stages, it is possible, that the actual position detected by the trigger circuitry is not continuously increasing, but flickering around one encoder count (20 nm) from time to time. When using the default setting for the EncoderSettlingTime (0.075 μs) under these conditions, it is well likely possible that more than one trigger pulse is generated (since the stage, seen by the controller, is moving back and forth). A higher value setting for the EncoderSettlingTime could avoid these unwanted and unpredictable extra trigger pulses in this case.

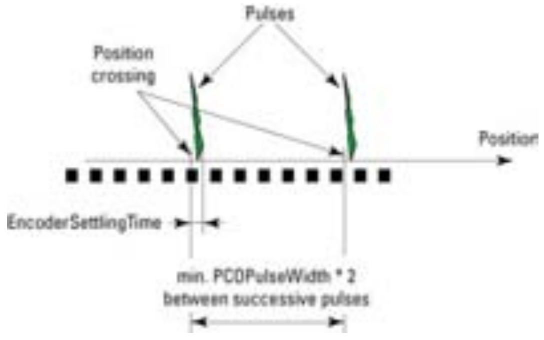


Figure 47: Position Compare Output.

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.

<u>**Example**</u>

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)

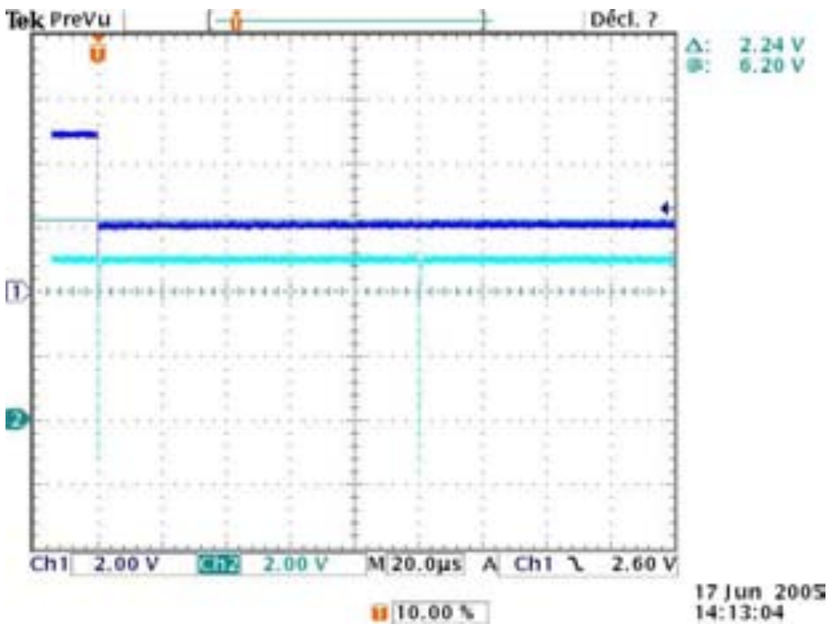
The group has to be in a READY state for the position compare to be enabled. Also, the PositionerPositionCompareSet() function must be completed before PositionerPositionCompareEnable() function. In this example, one trigger pulse is generated every 0.002 mm between the minimum position of 5 mm and the 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 E, PCO connector, for details.

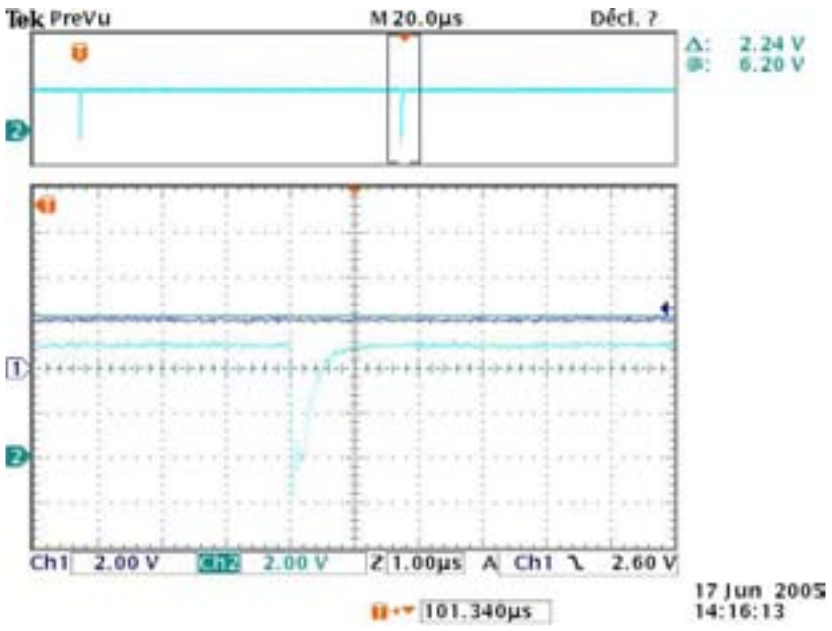
This table summarizes the results of the example above:

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, however there is a longer fall time that is related to the time constant of the RC circuit used. Please note that only the falling edge of the pulse is precise and should be used for synchronization purposes.

IMPORTANT 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 the interpolation factor defined by the function PositionerHardInterpolatorFactorSet.

AnalogInterpolated encoder

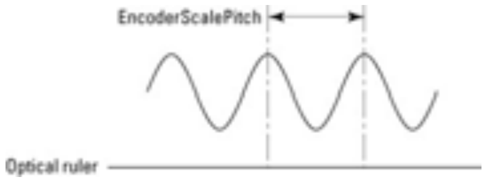


Figure 48: AnalogInterpolated Encoder.

$$\textbf{Trigger resolution} = \frac{\text{EncoderScalePitch}}{\text{PositionerHardInterpolatorFactor}}$$

Trigger pulses

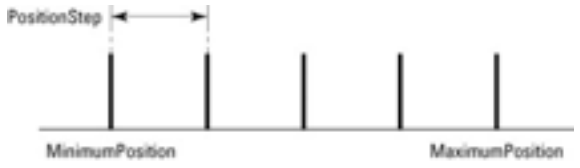


Figure 49: Trigger Pulses.

MinimumPosition, MaximumPosition, and PositionStep should be multiples of the Trigger resolution. If not, rounding to the nearest multiple value is made.

13.2 AquadB signals

In the AquadB signal 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 hardware interpolator by the function PositionerHardInterpolatorFactorSet ().

Example

XM stages feature an analog encoder with a signal period of 4 µm. With the setting PositionerHardInterpolatorFactorSet (200) the post-quadrature resolution of the AquadB signals is: 4 µm / 200 = 0.02 µm. In this case one full period of the AquadB signals equals 0.08 µm.

The following functions are used to configure AquadB signals:

PositionerPositionCompareAquadBWindowedSet

PositionerPositionCompareAquadBWindowedGet

PositionerPositionCompareEnable

PositionerPositionCompareAquadBAlwaysEnable

PositionerPositionCompareDisable

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 parameter:

Positioner name

Minimum Position

Maximum Position

To enable the AquadB signals, the function PositionerPositionCompareEnable() must be sent.

<u>**Example**</u>

GroupInitialize(MyStage)

GroupHomeSearch(MyStage)

PositionerPositionCompareAquadBWindowedSet(MyStage.X, 10, 20)

PositonerPositionCompareEnable(MyStage.X)

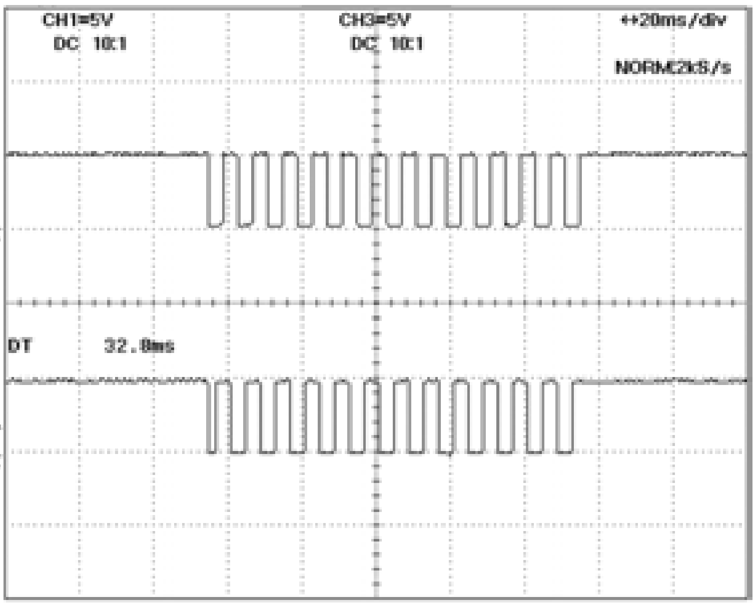
PositionerPositionCompareGet(MyStage, &MinimumPosition, &MaximumPosition, &EnableState)

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.

IMPORTANT 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 (see next section).

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().

13.3 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 50 ns jitter for the trigger pulses. The duration of the pulse is 200 nsec by default and can be modified using the function **PositionerPositionComparePulseParametersSet** (). Possible values for the PCOPulseWidth are: 0.2 (default), 1, 2.5 and 10 (μs). Please note, that only the falling edge of the trigger pulse is precise and only this edge should be used for synchronization irrespactable from the PCOPulseWidth setting. Note also, that the duration of the pulse detected by your electronics may be longer depending on the time constant of your RC circuit. Successive trigger pulses should have a minimum time lag equivalent to the PCOPulseWidth time times two.

The following functions are used to generate time spaced pulses:

PositionerTimeFlasherSet

PositionerTimeFlasherGet

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 0.0000004 seconds (0.4 μs) and less than or equal to 50 seconds. Furthermore, the time interval must be a multiple of 25 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)

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 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 E, PCO connectors for details.

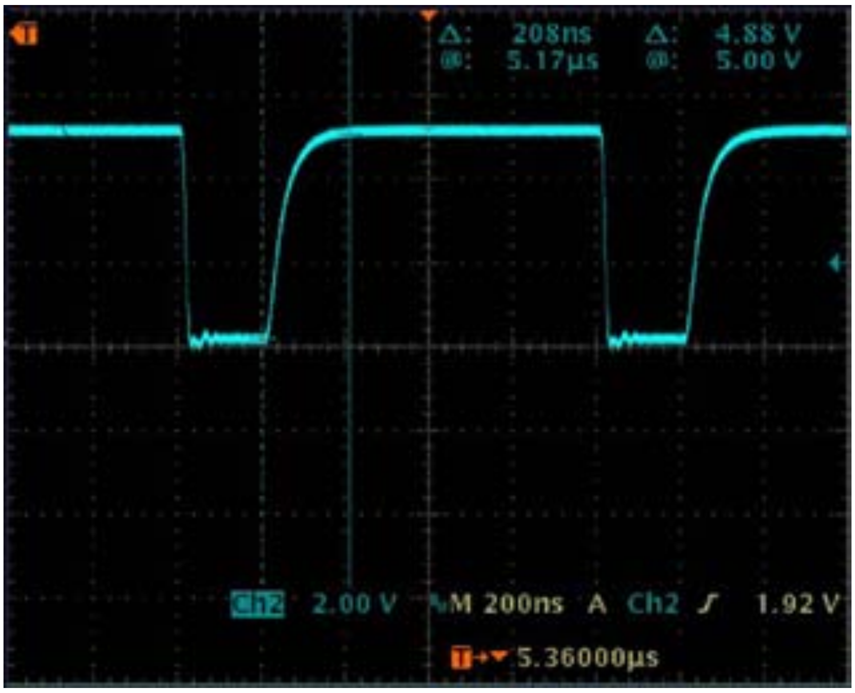


Figure 50: Temporal resolution of time spaced pulses in oscilloscope view.

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 20 nm (setting PositionerHardInterpolatorFactorSet(200)). If the goal is to get pulses at a nominal distance of 268 nm at a speed of 200 mm/s speed, this is not possible using the distance spaced pulse configuration. Either 260 nm or 280 nm are possible, but not 268 nm. With some minor adjustments to the target speed, however, this is well possible using the time spaced pulse configuration:

The target speed is 200 mm/s, the desired distance between successive pulses is 268 nm. So the nominal time interval between successive pulses is:
268 nm / 200 mm/s = 1.340 µs

Round this nominal value to the next possible time interval, means to the next integer multiple of 25 ns: 1.350 µs
Use this rounded time interval to calculate a corrected velocity:
268 nm / 1.350 µs = 198.51852 mm/s

GroupMoveAbsolute(MyStage.X, -50)
PositionerSGammaParametersSet(MyStage.X, 198.51852, 2500, 0.02, 0.02)
PositionerTimeFlasherSet(MyStage.X, -30, 30, 0.00000135)
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. As the stage moves at a speed of 198.51852 mm/s, the nominal distance between successive pulses is: 198.51852 mm/s * 1.35 µs = 268 nm.

13.4 Triggers on Line-Arc Trajectories

This capability allows output of 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 a rate of 10 kHz. This means that the resolution of the trajectory length is 0.0001 * trajectory velocity. For a trajectory velocity of 100 mm/s for instance, the resolution of the trajectory length is 10 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:

GPIO2, pin11, Window:	A constant 5 V signal is sent between the StartLength and the EndLength.
GPIO2, pin12, Pulse:	A 1 µs pulse with 5 V peak voltage is sent every PathLengthInterval.

For details about the XPS I/O connectors, see appendix, section 20.2.

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 section 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 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.

13.5 Triggers on PVT Trajectories

This capability allows output of pulses at constant time intervals on a PVT trajectory. The pulses are generated between a first and a last trajectory element (see 9.3, PVT Trajectories for details). The minimum possible time interval is 100 µs.

Two signals are provided:

GPIO2, pin11, Window:	A constant 5 V signal is sent between the beginning of the first and the end of the last trajectory element.
GPIO2, pin12, Pulse:	A 1 µs pulse with 5V peak voltage is sent for every time interval

For details about the XPS I/O connectors, see appendix, section 20.2.

To define the first element, the last element and the time interval, use the function **MultipleAxesGroupPVTPulseOutputSet()**.

<u>**Example 1**</u>

MultipleAxesGroupPVTPulseOutputSet (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(XY, 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 section 11.0: “Event Triggers“ for details). This allows the gathering of data on a trajectory.

<u>**Example 2**</u>

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 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.

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 within 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.

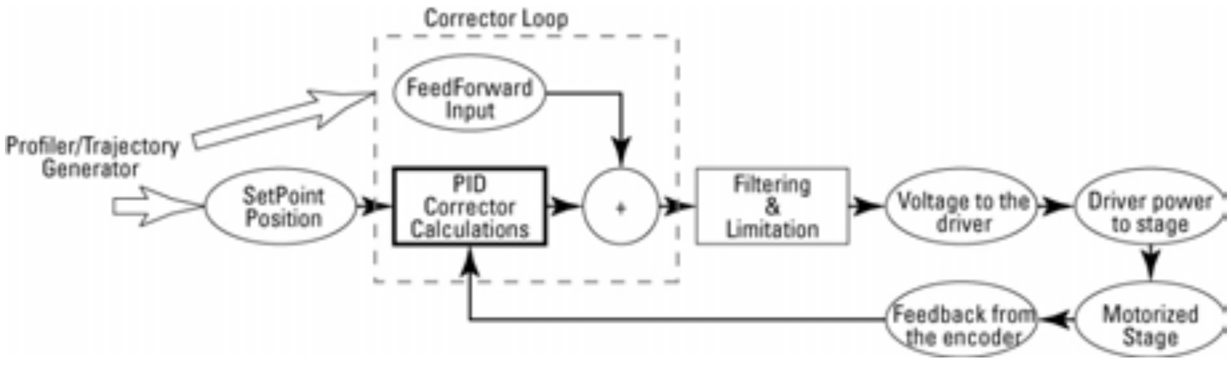


Figure 51: Servo structure and Basics.

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 an internal driver like 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 a rate of 2.5 kHz.

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 a rate of 10 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 on PID tuning for more information and tips on PID tuning. The PID corrector loop and trajectory generation loop 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, the PID loop is updated 4 times 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 a sample of typical stages 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-DRVM1, 2, 3

  - ±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-DRVM4

  - ±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, ILSCCHA):

  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 (UTMPP, 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 (CMA, 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 should 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.

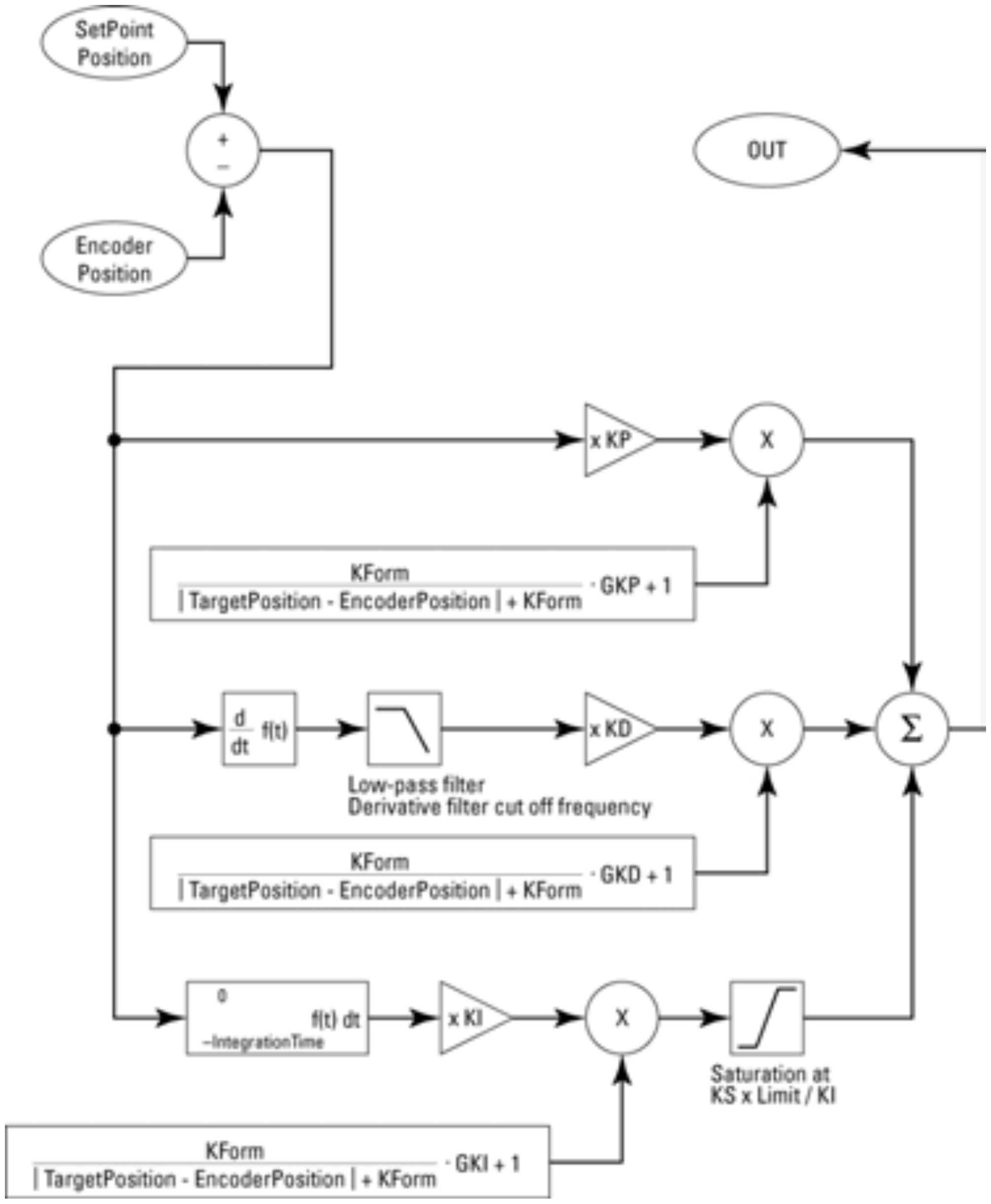

SetPoint Position
Encoder Position
+
–
OUT
x KP
X
KForm / | TargetPosition - EncoderPosition | + KForm · GKP + 1
d/dt f(t)
Low-pass filter
Derivative filter cut off frequency
x KD
Σ
KForm / | TargetPosition - EncoderPosition | + KForm · GKD + 1
0
f(t) dt
–IntegrationTime
x KI
Saturation at KS x Limit / KI
KForm / | TargetPosition - EncoderPosition | + KForm · GKI + 1

Figure 52: 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:

$$\frac{d}{dt} = \left[\sin(2 \ \ Fr\,t\ )\right] = 2 \ \ Fr \cos\left(2 \ \ \ Fr\,t\right)$$

The result is that the derived term becomes preponderant at high frequencies, compared to 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:

$$\left[\sin(2 \ \ Fr\,t\ )\right] = \ \frac{1}{2} \ \ Fr \ \ \sin(2 \ \ \ Fr\,t)''$$

The result is that integral term becomes preponderant at low frequencies, compared to the proportional and derivate 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 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. Refer to the Programmers manual for more information on this function.

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 value of Integration Time 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 then 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.

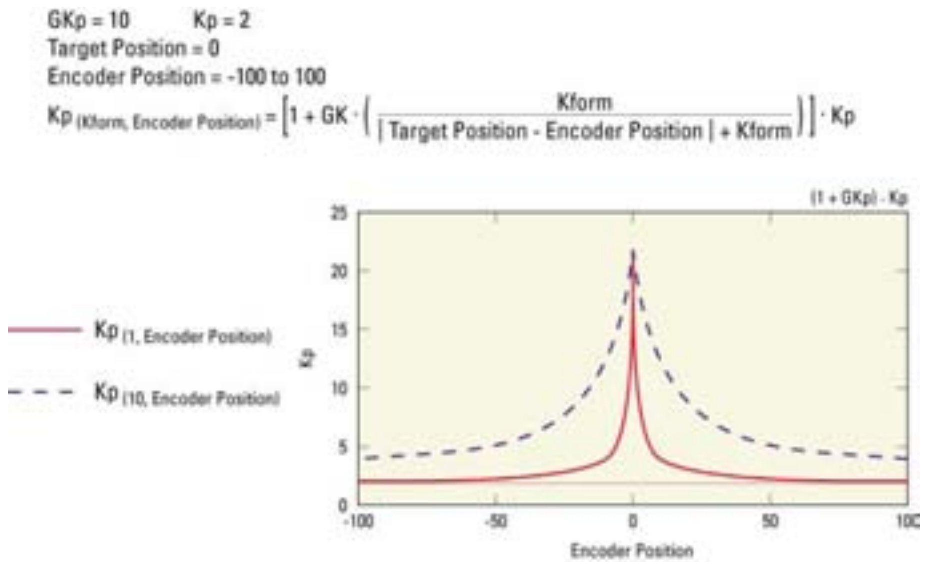


Figure 53: Variable Gains.

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 it's 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, the filtering and limitation parameters also have the same structure for all the correctors (PIDFFVelocity, PIDFFAcceleration and PIDFFDualVoltage, etc).

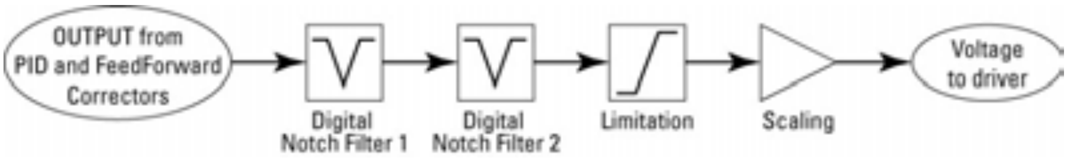


Figure 54: Filtering and Limitation.

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 for 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 mechanics natural frequencies, by lowering the gain at these frequencies. When they are implemented, these filters add some phase shift on 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 in the 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.

14.3 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.

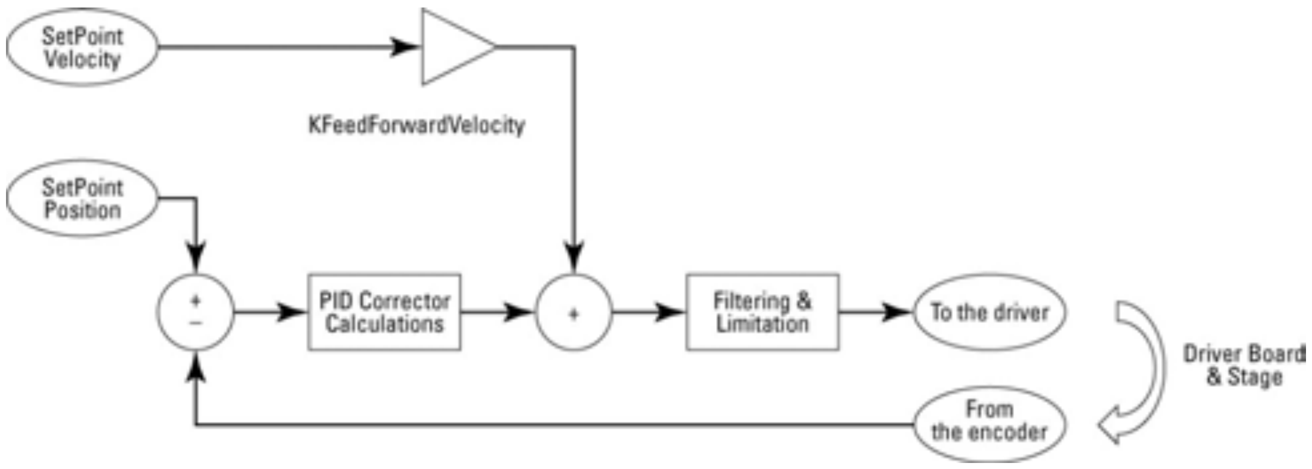


Figure 55: Corrector = PIDFFVelocity.

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:

$$\text{Gain} = \frac{\text{Ki}}{(2\lozenge \ \lozenge\text{Frq})^2}$$

This gain is equal to one at FrqI:

$$\text{FrqI} = \frac{1}{2\lozenge}\lozenge\sqrt{\text{Ki}}$$

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

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.

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).

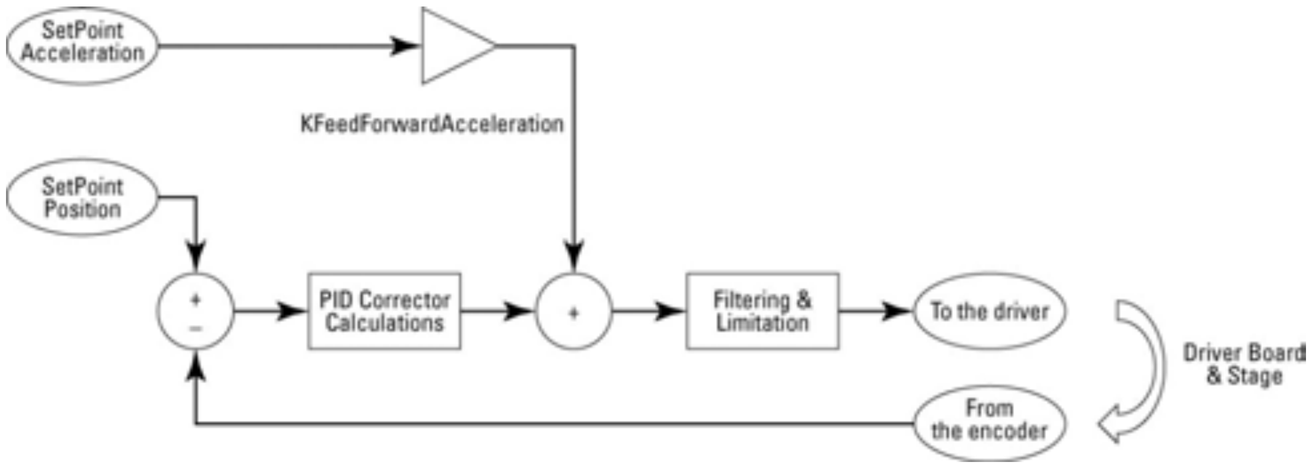


Figure 56: Corrector = PIDFFAcceleration.

14.3.2.1 Parameters

FeedForward:

A feed forward in acceleration is used.

KFeedForwardAcceleration is a gain that can be apply 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 volts input on 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:

$$\text{Gain} = \frac{\text{Kp}}{(2\Diamond\ \Diamond\text{Frq})^2}$$

This gain is equal to one at FrqP:

$$\text{FrqP} = \frac{1}{2\Diamond}\Diamond\sqrt{\text{Kp}}$$

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:

$$\text{Gain} = \frac{\text{Ki}}{(2\Diamond\ \Diamond\text{Frq})^3}$$

This gain is equal to one at FrqI:

$$\text{FrqI} = \frac{1}{2\Diamond}\Diamond\text{Ki}^{\frac{1}{3}}$$

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

1. Verify the AccelerationFeedForward in open loop (adjustment done using ScalingAcceleration).

   Close the loop, set Kd, increase it to minimize following errors until vibrations appears 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 appearance of overshoot/oscillations.

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.

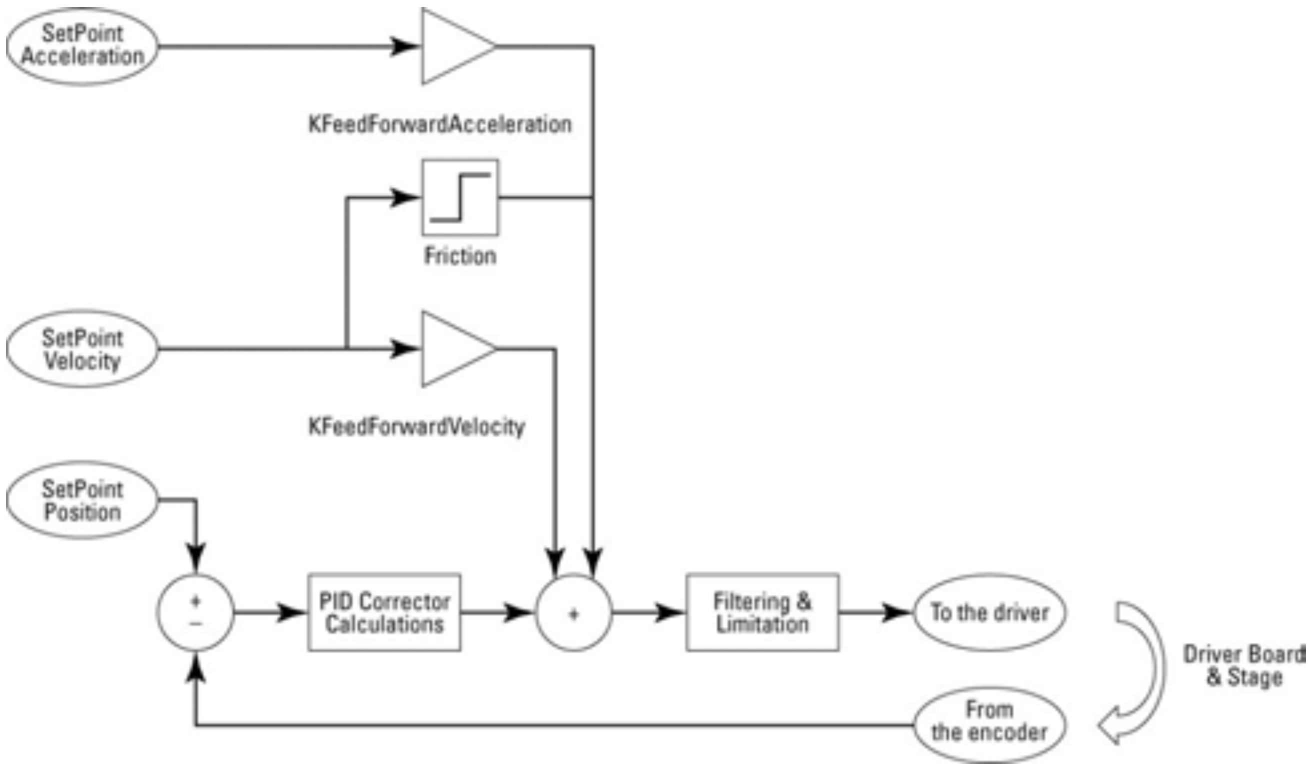


Figure 57: Corrector = PIDDual FFVoltage.

14.3.3.1 Parameters

FeedForward:

3 feed forwards are used: Speed, Acceleration and Friction.

KFeedForwardAcceleration is a gain that can be apply to the feed forward in acceleration.

KFeedForwardVelocity is a gain that can be apply 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.

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

1. Adjust KFeedForwardVelocityOpenLoop to optimize the following 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.

AnalogPosition interface is to be used with a driver having a position input (example = piezo driver).

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).

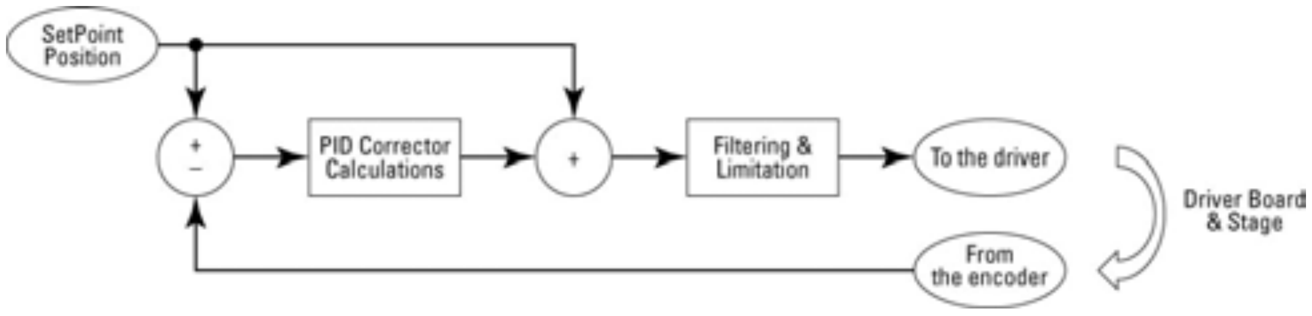


Figure 58: Corrector = PIPosition.

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:

$$\text{Gain} = \frac{\text{Ki}}{2\Diamond\ \ \Diamond\text{Frq}}$$

This gain is equal to one at:

$$\text{FrqI} = \frac{\text{Ki}}{2\Diamond}$$

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, sine to cosine amplitude differences, and phase shift.

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

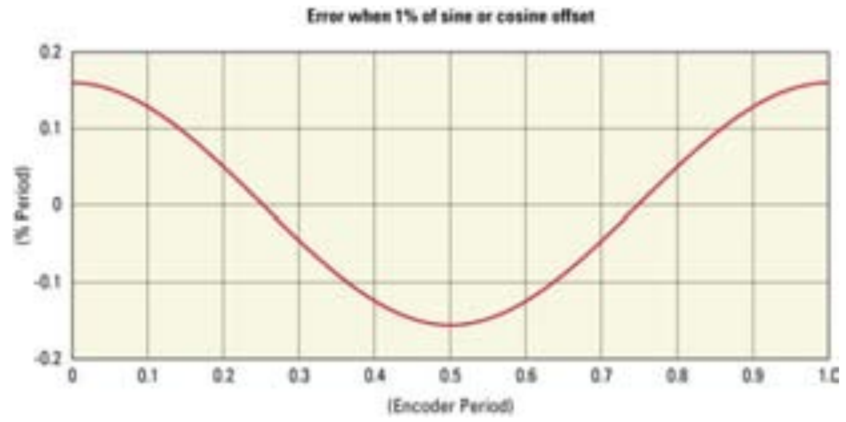


Figure 59: Offset Error.

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.

Amplitude Mismatch

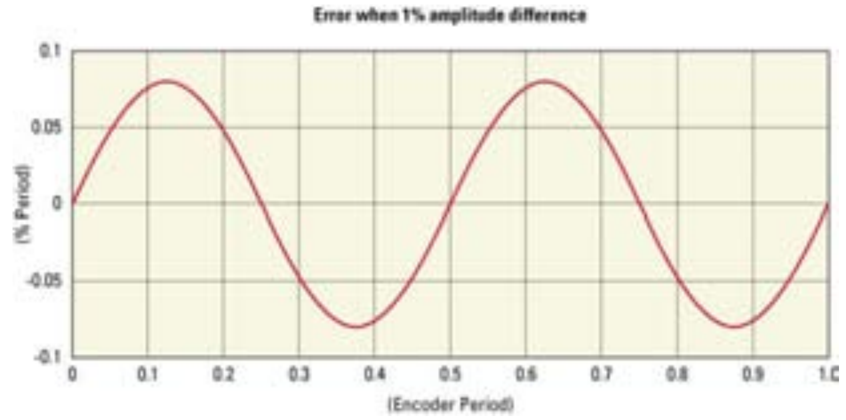


Figure 60: Amplitude Mismatch.

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.

Phase Shift

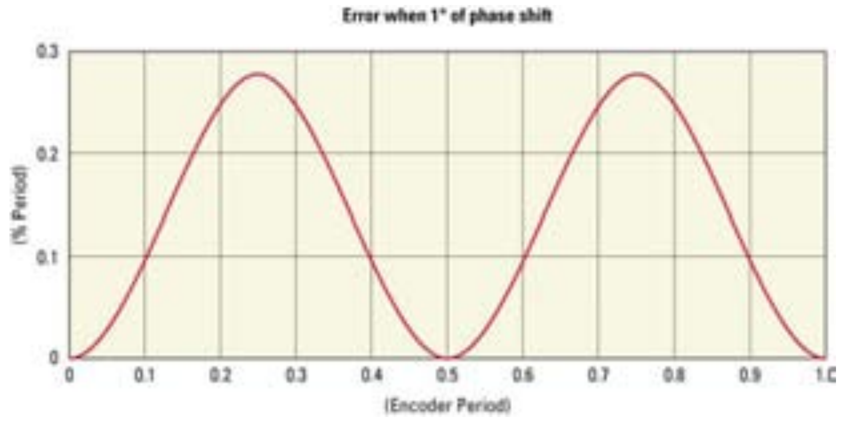


Figure 61: Phase Shift.

The phase shift between sine and cosine generates 0.28% interpolation error per degree phase shift. With a 20 μm scale pitch, 1 degree phase shift between sine and cosine generates 55.5 nm peak to peak error.

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% phase mismatch and 1 degree phase error between sine and cosine generates 132.5 nm peak to peak error.

$$2000\sqrt{(0.32\%)^2 + (0.32\%)^2 + (0.164\%)^2 + (0.28\%)^2} = 111.373$$

Note that the calculated value, 111.373 nm is lower than the measured 132.5 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

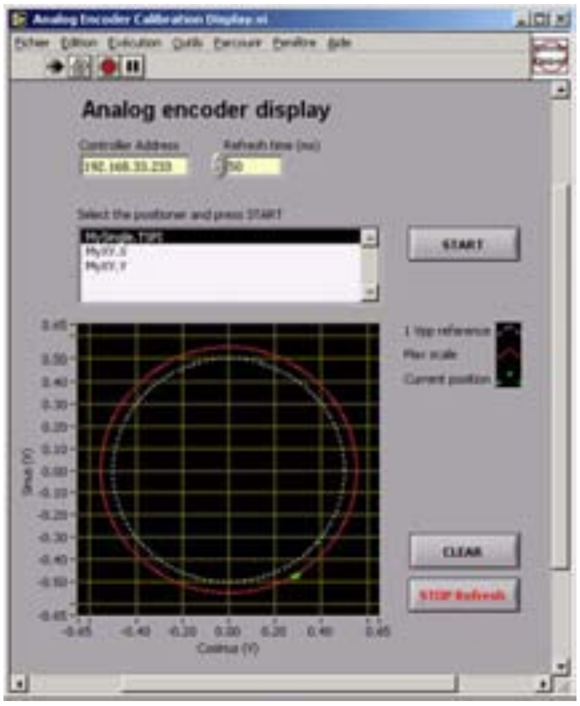
EncoderDifferentialGain = 0

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 determine 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.

In the folder ..\Admin\Public\Drivers\LabView\XPS-C8 of the XPS controller, 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.



<u>Example of the use of the functions</u>

GroupInitializeWithEncoderCalibration(MyGroup)

PositionerEncoderCalibrationParametersGet(MyGroup.MyStage)

This function returns the encoder calibration parameter values: encoder sinus signal offset, encoder cosinus signal offset, encoder differential gain, and encoder phase compensation. These values need to be entered in the appropriate section of the stages.ini.

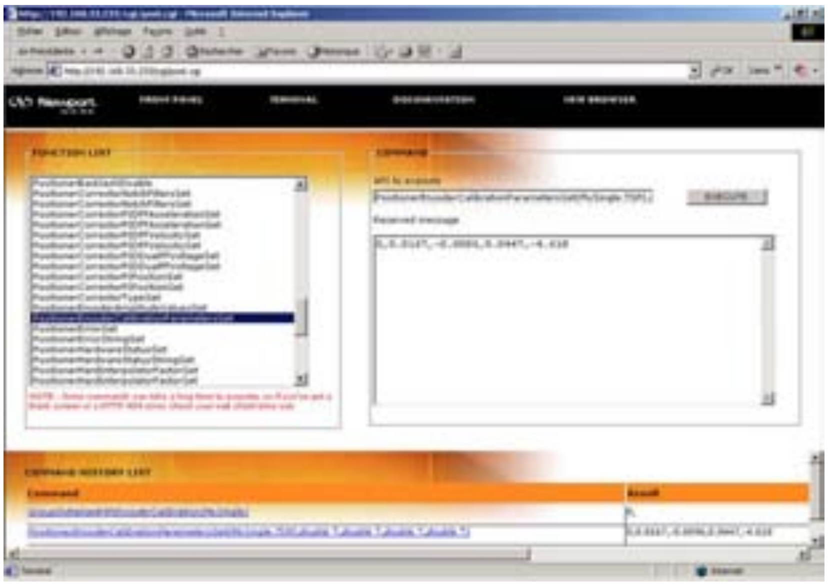
PositionerEncoderAmplitudeValuesGet(MyGroup.MyStage)

This function returns the encoder amplitude values: encoder sinus signal maximum amplitude value, encoder sinus signal current amplitude value, encoder cosinus signal maximum amplitude value and encoder cosinus signal current amplitude value.

Following is the complete process for calibrating a stage with an analog encoder interface:

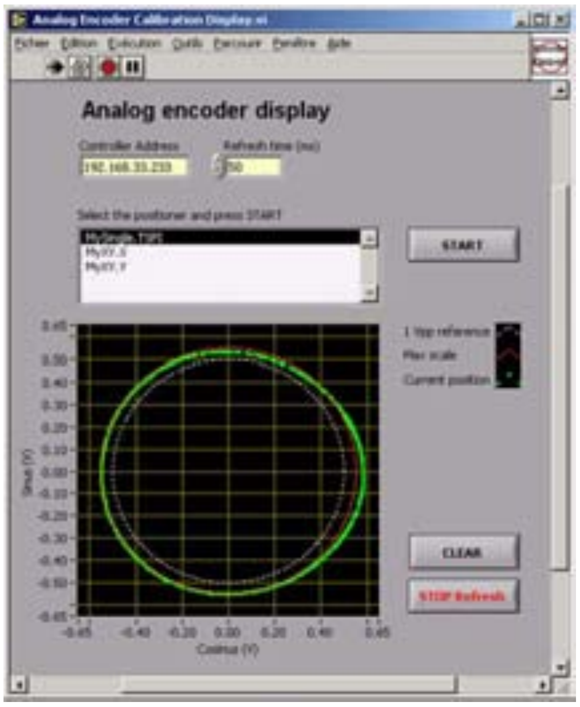
Step 1

Initialize the positioner and run the calibration routine.



Step 2

Start the AnalogEncoderCalibrationDiplay VI. Move the positioner at very low speed.


Analog encoder display

Controller Address: 192.168.33.233

Refresh time (ms): 50

Select the positioner and press START

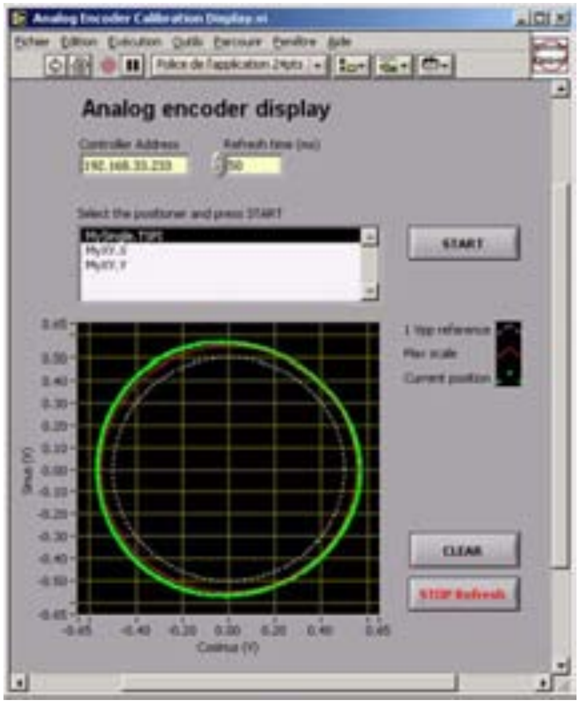
MySingle.TSP1
MyXY.X
MyXY.Y

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.



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 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 processes can be written in different ways, but essentially can be distinguished between host-managed processes, processes controlled from a PC with communication to the XPS via the Ethernet TCP/IP interface and XPS-managed processes, processes controlled directly by the XPS controller via a TCL script.

Host-managed processes

Host-managed processes are recommended for applications that require a lot of data management or a lot of digital communication with other 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 gets) information to (and from) the XPS controller via the Ethernet TCP/IP communication interface. Communication to the XPS controller can be established from almost any PC and is independent of the PC's operating system (Windows, Linux, Unix, Mac OS…) and programming language (LabView, C++, C, VisualBasic, Delphi, etc.). The XPS controller supports the development of host-managed processes with a Windows communication DLL, a complete set of LabView drivers and a number of example programs in C++, VisualBasic and LabView. A few basic examples are provided in this section. For more details, please refer to the Software Drivers 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-source string-based command language. With only a few fundamental constructs, it is very easy to learn and it is almost as powerful as C. Users of XPS can use TCL to write complete application code and XPS allows them to include any function to a TCL script. When developed, the TCL script can be executed in real time on the motion controller in the background, utilizing time that the controller does not need for servo or communication. Multiple TCL programs run in a time sharing mode. To learn more about TCL, refer to the TCL Manual which is accessible from the web site of the XPS controller.

The advantage of XPS-managed processes compared to host-managed processes is faster execution and better synchronization in many cases without any time taken from the communication link. XPS-managed processes or sub-processes are particularly valuable with 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 systems use.

The XPS controller has real-time multi-tasking functionality, and with most applications it 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 today's most complex and critical applications in terms of time and precision. It is recommended to spend a lot of thought to the proper definition of the best process approach.

The aim of this section is to provide a brief introduction to the different ways of XPS programming. This section, however, cannot address all details. For further information, refer to the TCL and the software drivers manual of the XPS controller which are accessible via the XPS web site interface.

16.1 TCL Generator

The TCL generator provides a convenient way of generating simple executable TCL scripts. These scripts may serve also as a good place to start for the development of more complex scripts.

The TCL generator is accessible from the terminal menu of the XPS web site. Pressing the TCL generator button generates a TCL script that includes the commands previously executed and listed in the Command history list. Note that the order in the TCL script is the same as executed and inverse to the order of the Command history list. The name of the TCL script is History.tcl and is stored in the ..\Admin\Public\Scripts folder of the controller.

<u>**Example**</u>

This is an example using three stages, two of them in an XY group and one in a SingleAxis group.

The following functions were executed:

KillAll()

GroupInitialize(S)

GroupInitialize(XY)

GroupHomeSearch(S)

GroupHomeSearch(XY)

GroupMoveAbsolute(S, 70)

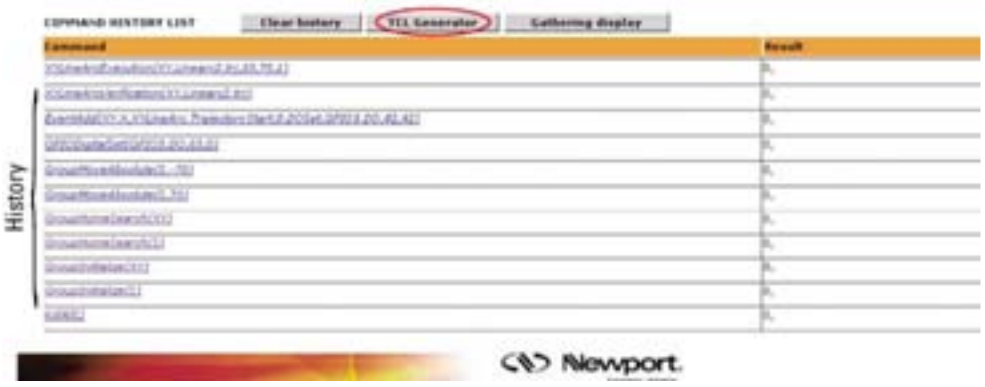
GroupMoveAbsolute(S, -70)

GPIODigitalSet(GPIO3.DO, 63, 0)

EventAdd(XY.X, XYLineArc.TrajectoryStart, 0, DOSet, GPIO3.DO, 42, 42)

XYLineArcVerification(XY, Linearc2.trj)

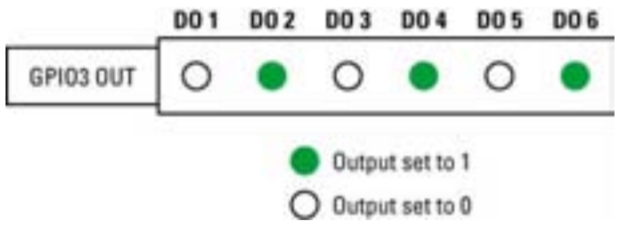
XYLineArcExecution(XY, Linearc2.trj, 10, 70, 1)



To execute the script, use the function TCLScriptExecute(History.tcl, task1, 0).

In this example, after initializing and homing both groups, the TCL script moves the single axis stage to the position 70 units, then to the position –70 units. It also sets the digital output GPIO3 to 0.

Once checked, the line arc trajectory defined in the Linearc2.trj file gets executed with a velocity of 10 units/sec and an acceleration of 70 units/sec. When this trajectory starts, more precisely when the positioner of the X axis starts moving, the bits #2, #4 and #6 of the output GPIO3 are set to 1 (4210 = 1010102).



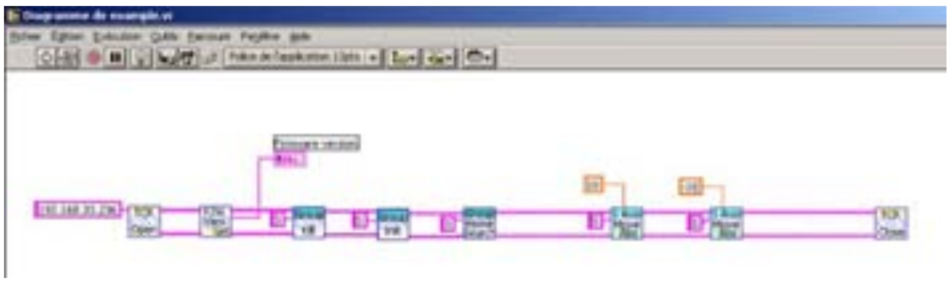
NOTE

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 learn more TCL programming, refer to the TCL Manual accessible from the documentation menu of the XPS web site. The TCL manual provides a complete description of all TCL commands and some more complex examples of TCL scripts.

16.2 LabView VIs

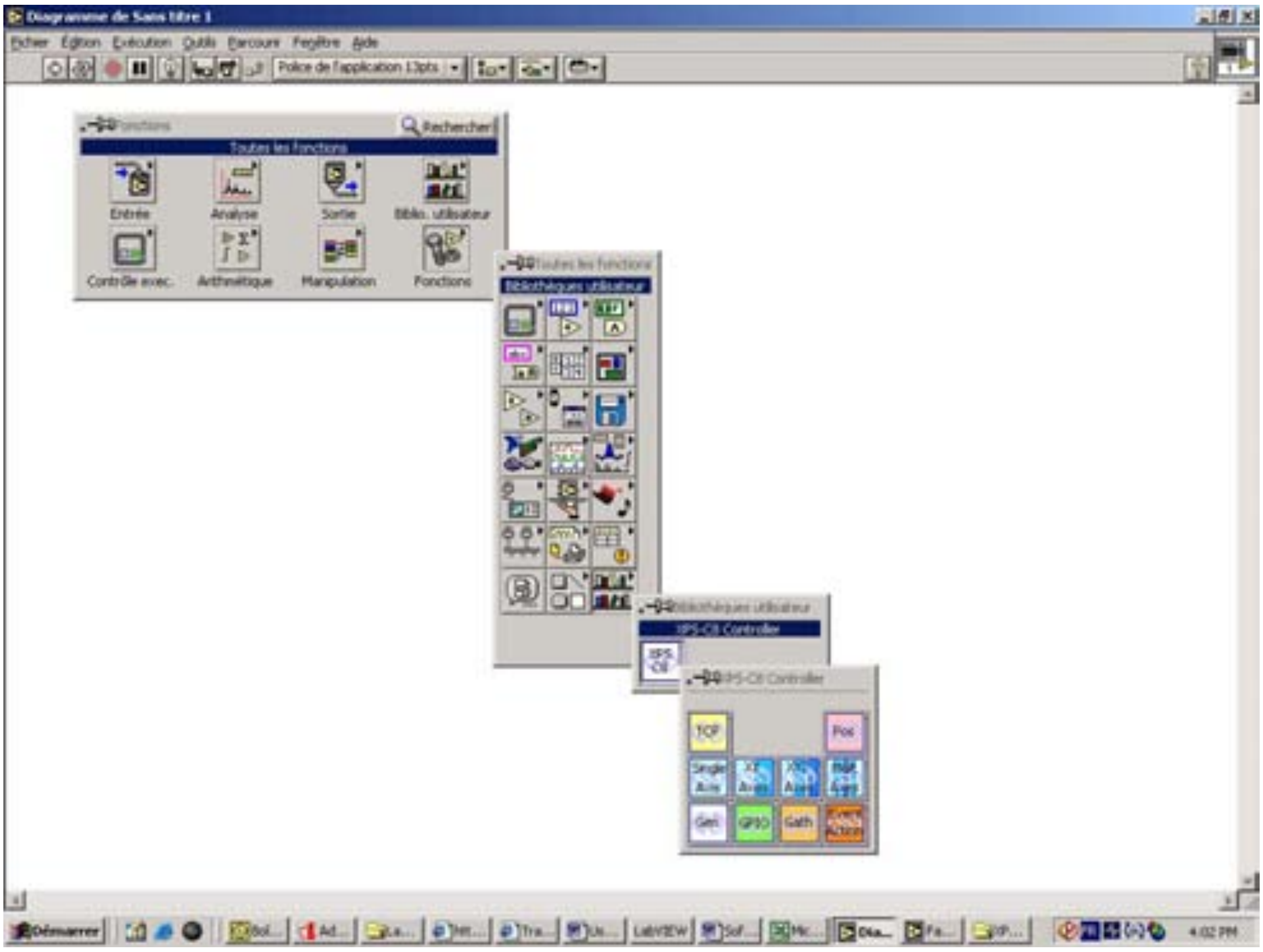
LabView is one of the most popular programming languages for the XPS controller. Newport provides a complete list of LabView drivers for the XPS controller, which means that LabView VIs exist for all XPS commands. In this section, a simple LabView application was developed that sequentially sends a number of commands to the XPS. The final application is illustrated as follow:



To use the XPS LabView drivers, the library files from the *../Admin/Public/Drivers/LabView/XPS-C8* Controller folder of the XPS controller must be copied to the folder *Program Files/National Instruments/LabVIEW6.1/user.lib/XPS-C8* Controller of your host computer.

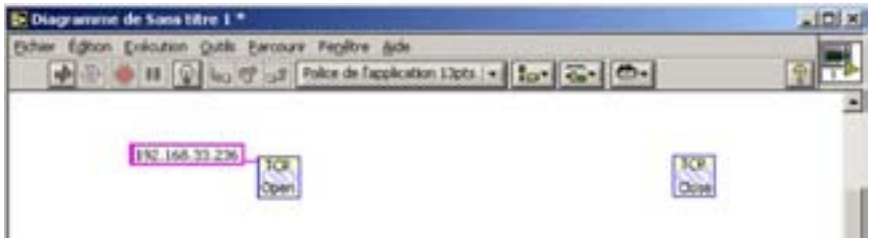
Start the LabView software and open an empty VI.

All drivers are located in the menu: *Window->Functions Pallet->Functions-> User library->XPS C8*. The drivers are classified in groups: TCP, General, Single axis, XY axes, XYZ axes, Multiple axes, Positioner, GPIO, Gathering and Events actions).

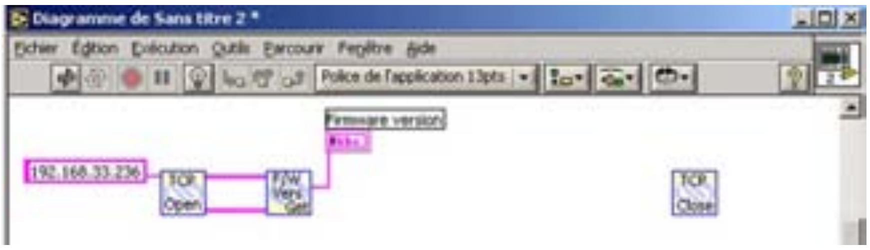


All LabView routines must begin with a *TCP Open* and must end with a *TCL Close*. If the TCP connection is not well managed, there will be no communication with the XPS. Then, add a constant chain (*Window->Functions Pallet->Functions->Chain*) to indicate the IP address of the controller and link it to the TCP Open driver.

When passing the cursor on the in/out panes of the driver, the required information is displayed.



Add a Firmware Version Get from the general group and associate an indicator to its second output. Link TCP Open to Firmware Version Get.



Then, add the initialization part: Add Group Kill, Group Initialize and Home search drivers from the single axis group and indicate the name of the group to each driver.

All drivers, except TCP Open and TCP Close, require a connection ID in (top left pane), a connection ID out (top right pane), an error in (bottom left pane) and an error out (bottom right.

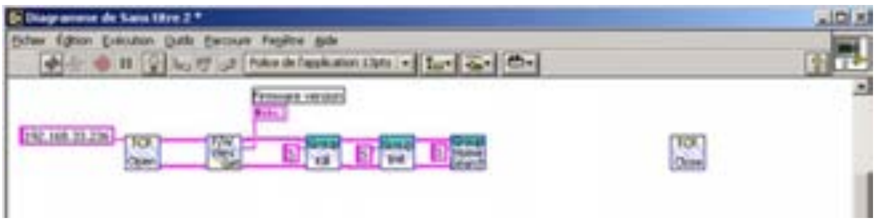


Diagramme de Sans titre 2 *
Fichier Édition Exécution Outils Parcourir Fenêtre Aide
Police de l'application 13pts
Firmware version
192.168.33.236
TCP Open
TCP Close

Add two Group move absolutes from the single axis group and indicate the name of the group. On the pane Target position, add a constant in which the desired absolute positions is written. Finally, link the drivers together and link the last move to the TCP Close.

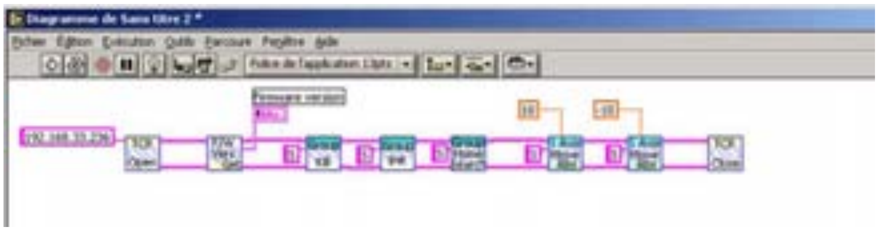
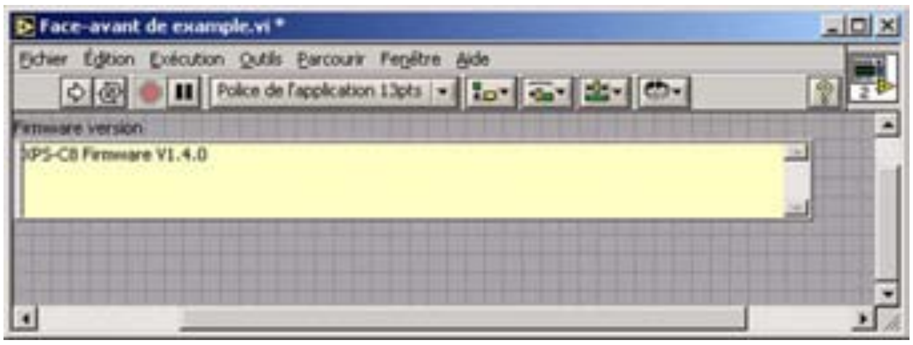


Diagramme de Sans titre 2 *
Fichier Édition Exécution Outils Parcourir Fenêtre Aide
Police de l'application 13pts
Firmware version
192.168.33.236
10
-10
TCP Open
TCP Close

To test the program, press the button .

This example displays the firmware version of the XPS controller and executes a motion of the single axis group from 0 to +10, and then to –10 units.

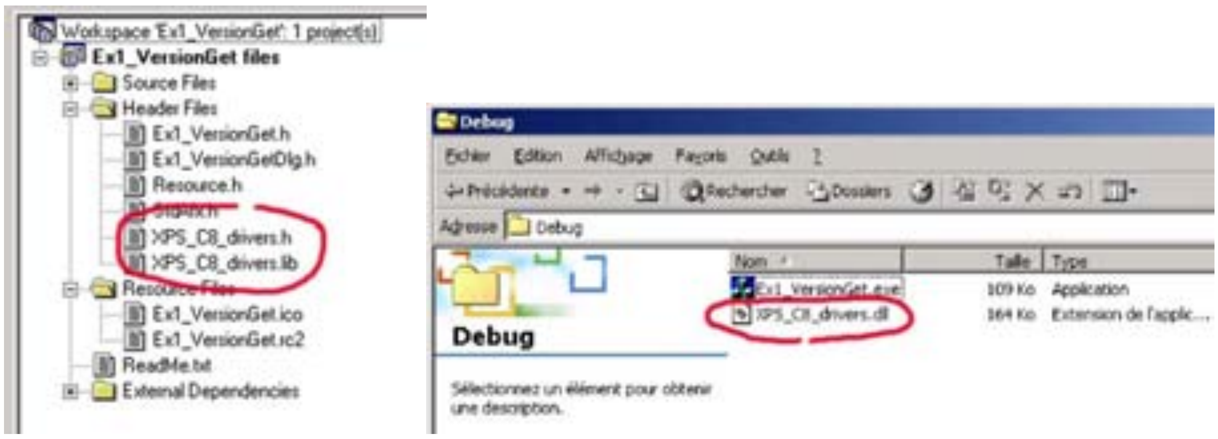

Face-avant de example.vi *
Fichier Édition Exécution Outils Parcourir Fenêtre Aide
Police de l'application 13pts
Firmware version

XPS-C8 Firmware V1.4.0

To learn more the XPS LabView drivers and their use, refer to the Software drivers manual accessible from the documentation menu of the XPS web site.

16.3 DLL Drivers

A DLL simplifies function calls from most programming languages. The DLL of the XPS controller is located in the ..Admin/Public/Drivers/DLL folder of the XPS controller. The files XPS_C8_drivers.h and XPS_C8_drivers.lib must be copied to the project folder and the file XPS_C8_drivers.dll to the folder of the executable file.


Workspace 'Ex1_VersionGet': 1 project(s)
Ex1_VersionGet files
Source Files
Header Files
Ex1_VersionGet.h
Ex1_VersionGetDlg.h
Resource.h
StdAfx.h
XPS_C8_drivers.h
XPS_C8_drivers.lib
Resource Files
Ex1_VersionGet.ico
Ex1_VersionGet.rc2
ReadMe.txt
External Dependencies
Debug
Fichier Edition Affichage Favoris Outils ?
Précédente Rechercher Dossiers
Adresse Debug
Nom Taille Type
Ex1_VersionGet.exe 109 Ko Application
XPS_C8_drivers.dll 164 Ko Extension de l'applic...
Debug
Sélectionnez un élément pour obtenir une description.

Once these files are added, for instance to a C++ project, the prototypes of the functions can be called in the program with the respective syntax of the functions (parameters number, type…). The file XPS_C8_drivers.h can be opened to see the list of the available functions and their prototypes.

For instance, the prototype of the function FirmwareVersionGet is as follow:

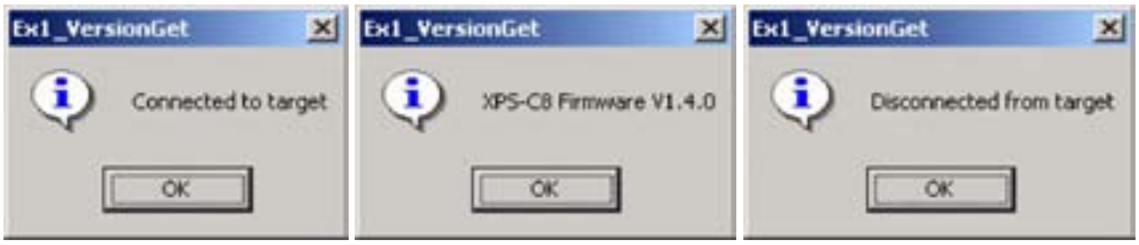
```
DLL int __stdcall FirmwareVersionGet (int SocketIndex, char * Version);
```

It requires two arguments (int and char*).

<u>Example of C++ sequence</u>

```
char  buffer [256] = {'\0'};
char pIPAddress[15] = {"192.168.33.236"};
int  nPort = 5001;
double dTimeOut = 60;
int  SocketID = -1;

// TCP / IP connection
SocketID = TCP_ConnectToServer(pIPAddress, nPort, dTimeOut);
if (SocketID == -1)
{
  sprintf (buffer, "Connection to @ %s, port = %ld failed\n", pIPAddress,
nPort);
  AfxMessageBox (buffer, MB_ICONSTOP);
}
else
{
  AfxMessageBox("Connected to target", MB_ICONINFORMATION);

  // Get controller version
FirmwareVersionGet (SocketID, buffer); // Get controller version
AfxMessageBox (buffer, MB_ICONINFORMATION);

  // TCP / IP disconnection
TCP_CloseSocket(SocketID);  // Close Socket
AfxMessageBox("Disconnected from target", MB_ICONINFORMATION);
}
```

This example opens a TCP connection, gets the firmware version of the XPS controller and closes the connection. The execution is displayed in message boxes:



Ex1_VersionGet

Connected to target

OK

Ex1_VersionGet

XPS-C8 Firmware V1.4.0

OK

Ex1_VersionGet

Disconnected from target

OK

To learn more about the DLL prototypes, refer to the *Programmer's Manual*, accessible from the web site interface of the XPS controller. All DLL prototypes are described there.

The *Software drivers manual*, also accessible from the XPS web site interface, provides further information about the use of the DLL and additional C++ programming examples.

16.4 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.

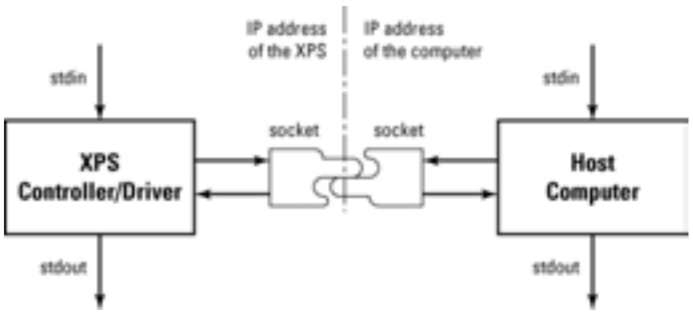


Figure 62: Running Processes in Parallel.

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 it in the received message window of the Terminal menu ('0' if command has been well executed or the error number in case of an error). That way, commands are executed sequentially as each command always waits for a feedback before 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 has 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 30 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 30.

User's 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 properly executed.

Examples of the use of parallel sockets

The following examples illustrate how to open several sockets via the web site interface, TCL scripts, LabView VIs and C++ programs.

Web site interface

The simplest way to open several sockets in parallel is to open several windows on the IP address of the controller. This is completely transparent to the user. Two or more groups of stages can be commanded for instance from two terminal menus at the same time to execute different motions (multitasking).

TCL scripts

A TCL script is carried out sequentially: the commands are executed one by one following the order they are written in the script. Consequently, there is no great interest to open several sockets in a single TCL script.

However, it is possible to start a TCL script from another TCL script. That way, as many sockets and parallel processes can be started in parallel as needed. Below is an example with 3 open sockets:

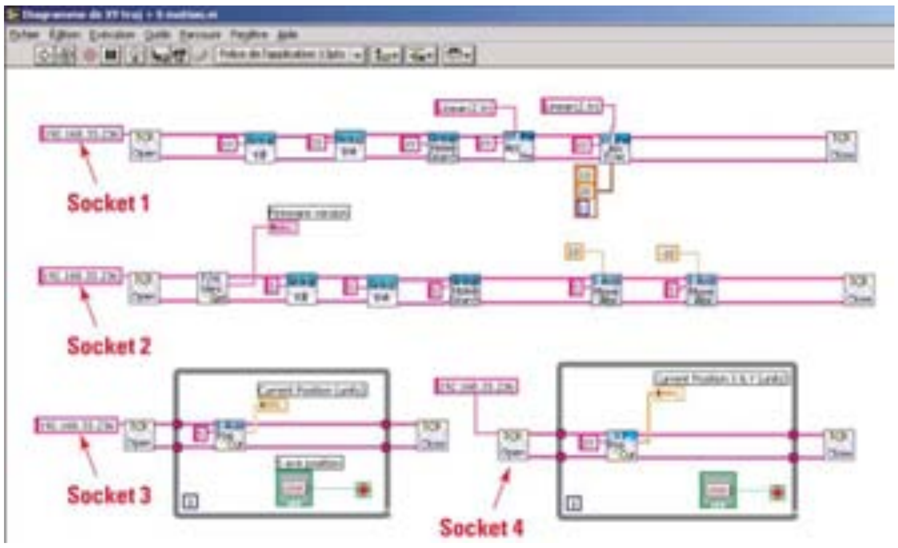
```
######################
# TCL program : GEN #
######################

set TimeOut 10
set code 0
set Prog1 “ProgRV.tcl”
set Task1 “Task1”
set Prog2 “ProgXY.tcl”
set Task2 “Task2”

# open TCP socket
set code [catch “OpenConnection $TimeOut socketID”]

if {$code == 0} {
   puts stdout “ProgGen : TCP_ConnectToServer OK => $code  ID =
$socketID”        <– Socket 1

set code [catch “TCLScriptExecute $socketID $Prog1 $Task1 0”]
puts stdout “ProgGen : TCLScriptExecute => error = $code”
                  <– Socket 2

set code [catch “TCLScriptExecute $socketID $Prog2 $Task2 0”]
puts stdout “ProgGen : TCLScriptExecute => error = $code”
                 <– Socket 3

# close TCP socket
set code [catch “TCP_CloseSocket $socketID”]
puts stdout “ProgGen : TCP_CloseSocket => $code  ID = $socketID”

} else  {
   puts stdout “ProgGen : TCP_ConnectToServer NOT OK => $code”
}
```

NOTE

Socket 2 and Socket 3 are not opened by the TCLExecuteScript function, but we supposed these scripts open some sockets on their own.

LabView VIs

In a VI file, several processes can easily be created, all beginning with a TCP Open and all finishing with a TCP Close. Each TCP Open will open its own socket. Shown below is a simple VI that opens 4 sockets at the same time.



C++ program

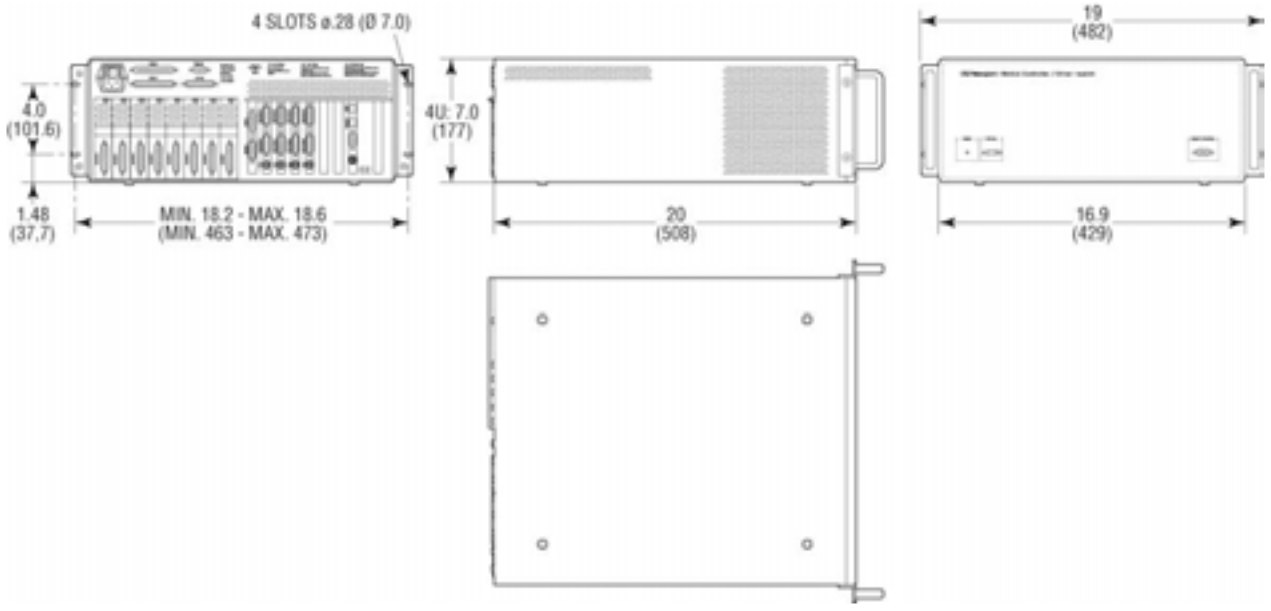
A C++ program is executed sequentially. Even if it calls many functions, they are always executed one by one following the order they are written. In order to open several sockets for multitasking, the C++ multithreading functionality must be used.

The XPS driver DLL allows a maximum number of 100 simultaneously opened sockets. One XPS controller supports a maximum number of 30 simultaneously opened sockets, but a program could control several XPS controllers.

Appendices

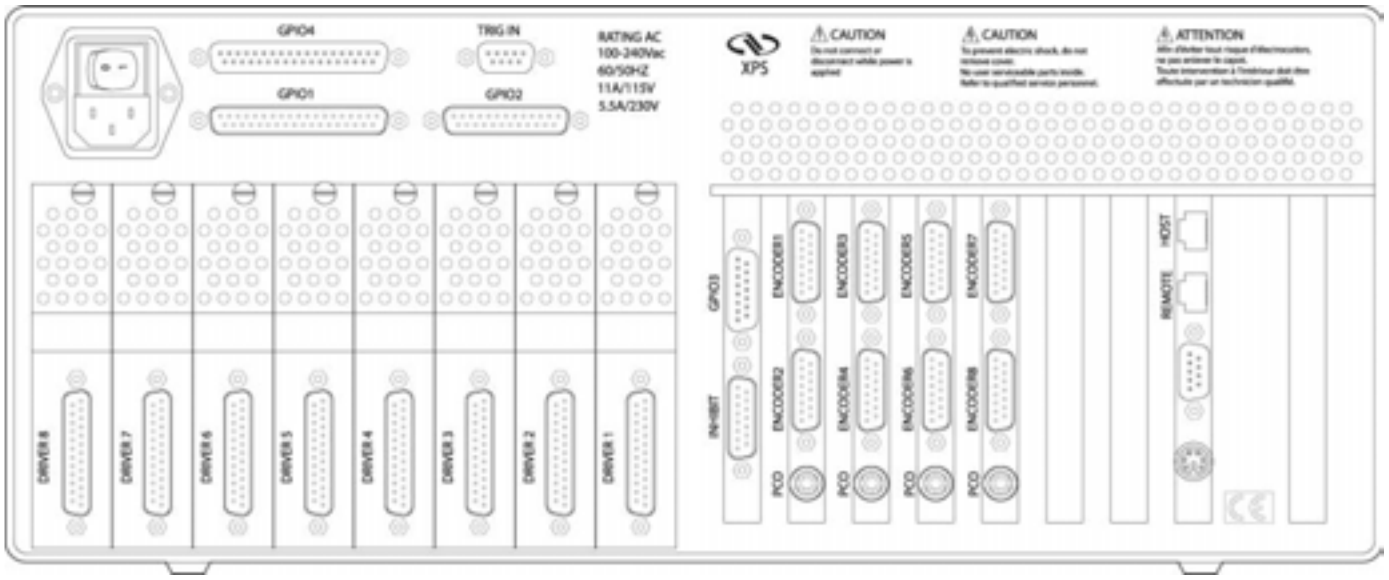
17.0 Appendix A: Hardware

17.1 Controller



Weight:	16 kg (32 lb)
Input voltage:	100–240 VAC
Input current:	11 A/115 V 5.5 A/230 V
Frequency:	60/50 Hz

17.2 Rear Panel Connectors



17.3 Environmental Requirements

Temperature range:

Storage:	-20 to +80 °C
Operating:	+5 to +35 °C

Relative Humidity (Non-condensing):

Storage:	10 to 95% RH
Operating:	10 to 85% RH

Altitude:

Storage:	To 10,000 ft
Operating:	To 5,000 ft

18.0 Appendix B: General I/O Description

This paragraph briefly describes all XPS signal types.

Description of each XPS connector interface is detailed in further paragraphs.

18.1 Digital I/Os (All GPIO, Inhibit and Trigger In and PCO Connectors)

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 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 refreshment rate.

Typical availability delay is 100μs due to function treatment.

All digital inputs are identical, except for GPIO3 inhibition input (described with GPIO3).

All digital inputs are in negative logic and have internal +5 V pull up resistors.

18.1.1 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

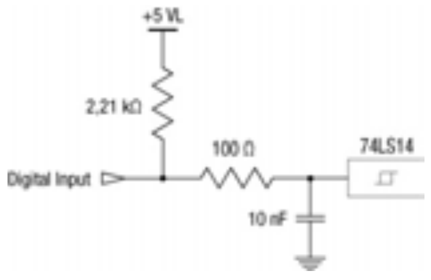


Figure 63: Digital TTL Input.

GPIOn inputs (n = 1 to 4) can be accessed via the GPIODigitalGet(GPIOn.DI, …) function.

All digital outputs are identical.

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.

18.1.2 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

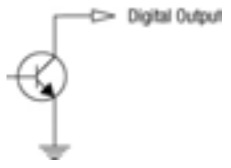


Figure 64: Open Collector Digital Output.

GPIOn outputs (n = 1 to 4) can be accessed via the GPIODigitalSet(GPIOn.DO, …) function.

18.2 Digital Encoder Inputs (Driver Boards & DRV00)

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.

18.3 Digital Servitudes (Driver Boards, DRV00 & 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 control loop (10 kHz).

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.

18.4 Analog Encoder Inputs (Analog Encoder Connectors)

Analog encoder interface comply with the Heidenhain LIF481 glass scales wiring standard.

18.5 Analog I/O (GPIO2 Connector)

18.5.1 Analog Inputs

The 4 analog inputs have ±10 V range, 14 Bit resolution, and a 15 kHz 2nd order low pass filter front end.

In all cases, the analog input values must be within the ±10 V. The analog input impedance is typically 22 kΩ. The maximum input current is ±500 μA.

1 LSB = 20 V/16384 ≈ 1.22 mV

The maximum offset error is ±17,1 mV.

18.5.2 Analog Outputs

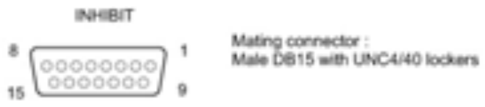
The 4 analog outputs have ±10 V range and 16 Bit resolution. The maximum offset error is ±2 mV, and the maximum gain error is ±6 LSB. The output settling time is typically 50 μsec at 1% of the target value (output filter is a 15 kHz 1st order low pass filter).

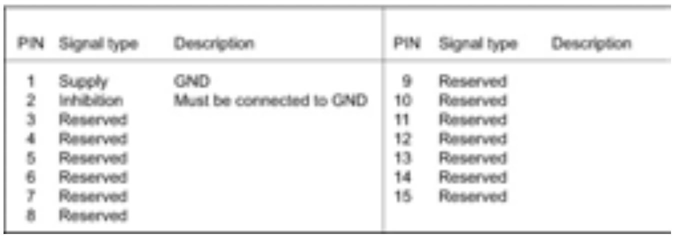
Analog outputs are voltage outputs (output current less than 1 mA), so to use them properly, they must be connected to impedance higher than 10 kΩ.

1 LSB =: 20 V/65536 ≈ 0.3 mV.

Analog outputs can be accessed via the GPIOAnalogSet(GPIO2.DACn,…) function.

19.0 Appendix C: Power Inhibit Connector





PIN	Signal type	Description	PIN	Signal type	Description
1	Supply	GND	9	Reserved	
2	Inhibition	Must be connected to GND	10	Reserved	
3	Reserved		11	Reserved	
4	Reserved		12	Reserved	
5	Reserved		13	Reserved	
6	Reserved		14	Reserved	
7	Reserved		15	Reserved	
8	Reserved				

Figure 65: Inhibition connector.

This connector is provided for the wiring of a remote STOP ALL switch.

It has the same effect as the front panel STOP ALL button.

Inhibition input is a standard TTL input.

Inhibition (Pin #2), must always be connected to GND during normal controller operation.

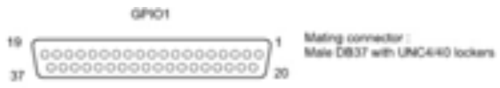
An open circuit is equivalent to pressing STOP ALL on the front panel. Wire the switch contacts normally closed.

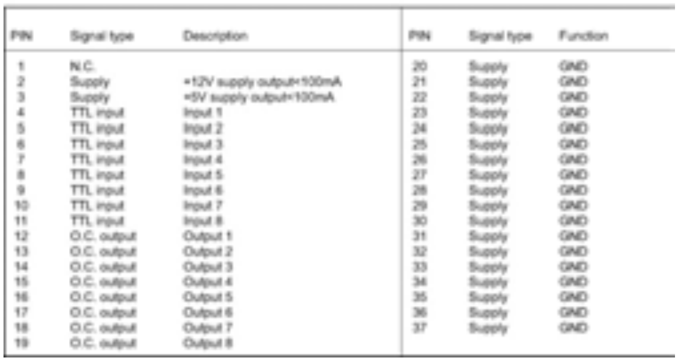
NOTE

Connecting more than one switch is not recommended on this input.

20.0 Appendix D: GPIO Connectors

20.1 GPIO1 Connector



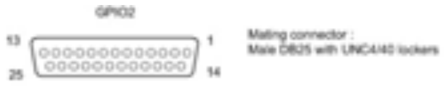


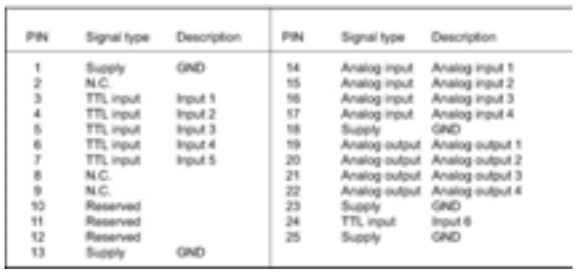
PIN	Signal type	Description	PIN	Signal type	Function
1	N.C.		20	Supply	GND
2	Supply	+12V supply output<100mA	21	Supply	GND
3	Supply	+5V supply output<100mA	22	Supply	GND
4	TTL input	Input 1	23	Supply	GND
5	TTL input	Input 2	24	Supply	GND
6	TTL input	Input 3	25	Supply	GND
7	TTL input	Input 4	26	Supply	GND
8	TTL input	Input 5	27	Supply	GND
9	TTL input	Input 6	28	Supply	GND
10	TTL input	Input 7	29	Supply	GND
11	TTL input	Input 8	30	Supply	GND
12	O.C. output	Output 1	31	Supply	GND
13	O.C. output	Output 2	32	Supply	GND
14	O.C. output	Output 3	33	Supply	GND
15	O.C. output	Output 4	34	Supply	GND
16	O.C. output	Output 5	35	Supply	GND
17	O.C. output	Output 6	36	Supply	GND
18	O.C. output	Output 7	37	Supply	GND
19	O.C. output	Output 8			

Figure 66: GPIO1 Digital I/O Connector.

General Purpose Inputs Outputs GPIO1 is the main XPS digital I/O connector.

20.2 GPIO2 Connector





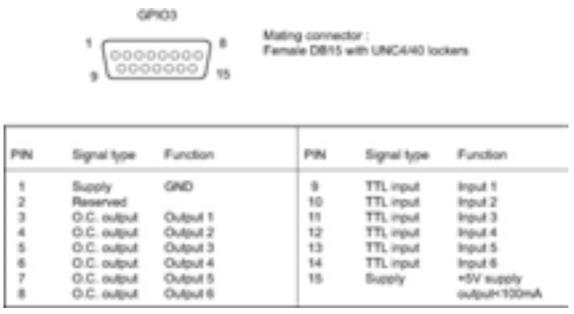
PIN	Signal type	Description	PIN	Signal type	Description
1	Supply	GND	14	Analog input	Analog input 1
2	N.C.		15	Analog input	Analog input 2
3	TTL input	Input 1	16	Analog input	Analog input 3
4	TTL input	Input 2	17	Analog input	Analog input 4
5	TTL input	Input 3	18	Supply	GND
6	TTL input	Input 4	19	Analog output	Analog output 1
7	TTL input	Input 5	20	Analog output	Analog output 2
8	N.C.		21	Analog output	Analog output 3
9	N.C.		22	Analog output	Analog output 4
10	Reserved		23	Supply	GND
11	Reserved		24	TTL input	Input 6
12	Reserved		25	Supply	GND
13	Supply	GND			

Figure 67: GPIO2 Analog & Digital Connector.

General Purpose Inputs Outputs GPIO2 is an additional digital input connector.

This connector is also the main analog I/O connector with 4 analog inputs and 4 analog outputs.

20.3 GPIO3 Connector

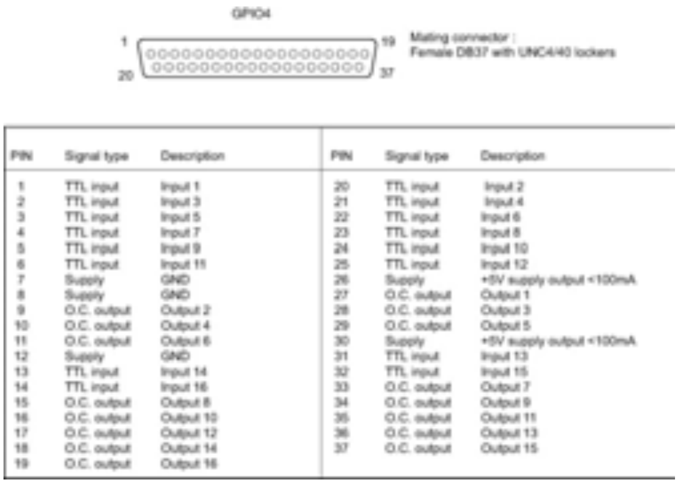


PIN	Signal type	Function	PIN	Signal type	Function
1	Supply	GND	9	TTL input	Input 1
2	Reserved		10	TTL input	Input 2
3	O.C. output	Output 1	11	TTL input	Input 3
4	O.C. output	Output 2	12	TTL input	Input 4
5	O.C. output	Output 3	13	TTL input	Input 5
6	O.C. output	Output 4	14	TTL input	Input 6
7	O.C. output	Output 5	15	Supply	+5V supply output<100mA
8	O.C. output	Output 6			

Figure 68: GPIO3 Digital I/O Connector.

General Purpose Inputs Outputs GPIO3 is a digital I/O connector.

20.4 GPIO4 Connector

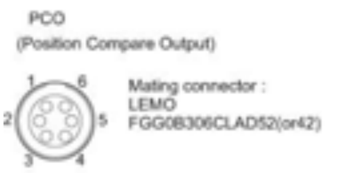


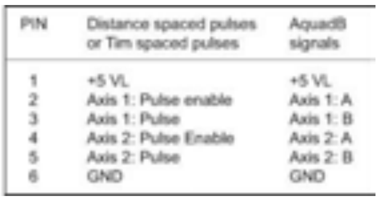
PIN	Signal type	Description	PIN	Signal type	Description
1	TTL input	Input 1	20	TTL input	Input 2
2	TTL input	Input 3	21	TTL input	Input 4
3	TTL input	Input 5	22	TTL input	Input 6
4	TTL input	Input 7	23	TTL input	Input 8
5	TTL input	Input 9	24	TTL input	Input 10
6	TTL input	Input 11	25	TTL input	Input 12
7	Supply	GND	26	Supply	+5V supply output <100mA
8	Supply	GND	27	O.C. output	Output 1
9	O.C. output	Output 2	28	O.C. output	Output 3
10	O.C. output	Output 4	29	O.C. output	Output 5
11	O.C. output	Output 6	30	Supply	+5V supply output <100mA
12	Supply	GND	31	TTL input	Input 13
13	TTL input	Input 14	32	TTL input	Input 15
14	TTL input	Input 16	33	O.C. output	Output 7
15	O.C. output	Output 8	34	O.C. output	Output 9
16	O.C. output	Output 10	35	O.C. output	Output 11
17	O.C. output	Output 12	36	O.C. output	Output 13
18	O.C. output	Output 14	37	O.C. output	Output 15
19	O.C. output	Output 16			

Figure 69: GPIO4 Additional Digital I/O Connector.

General Purpose Inputs Outputs GPIO4 is an additional digital I/O connector.

21.0 Appendix E: PCO Connector





PIN	Distance spaced pulses or Tim spaced pulses	AquadB signals
1	+5 VL	+5 VL
2	Axis 1: Pulse enable	Axis 1: A
3	Axis 1: Pulse	Axis 1: B
4	Axis 2: Pulse Enable	Axis 2: A
5	Axis 2: Pulse	Axis 2: B
6	GND	GND

Figure 70: Position Compare Output Connector.

There is one PCO connector for every two axes. 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 section 13, Output trigger, for more details.

The state of the enable signal is low when the stage is inside the programmed position compare window.

Note also, that only the falling edge of the trigger pulse is precise and only this edge should be used for synchronization irrespactable from the PCOPulseWidth setting.

The duration of the pulse is 200 nsec by default and can be modified using the function **PositionerPositionComparePulseParametersSet**(). Possible values for the PCOPulseWidth are: 0.2 (default), 1, 2.5 and 10 (μs). Successive trigger pulses should have a minimum time lag equivalent to the PCOPulseWidth time times two.

The signals are open collector type and accept up to 30 Volts and 40 mA

The +5V output provided on the PCO connector can be used to pull-up these outputs and can supply 50 mA max.

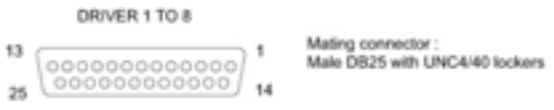
NOTE

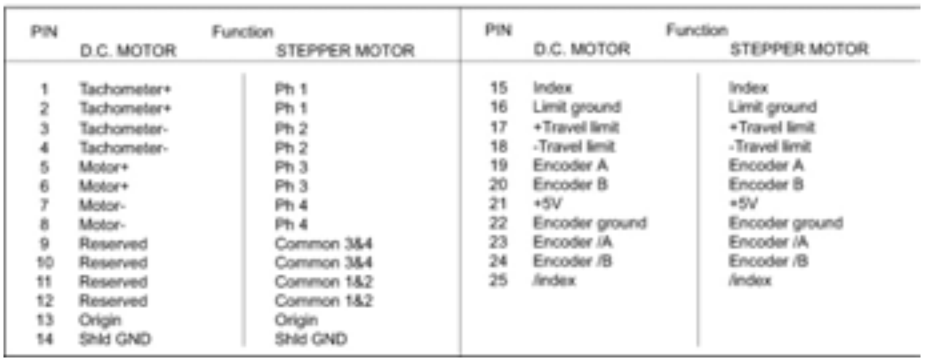
To ensure fast transitions with an open collector, it is necessary to have enough current to speed-up the transistor's junction capacitor charge / discharge. A good value is around 10 mA. So to pull-up the PCO signals to +5 V a 470 Ω resistor can be used.

Refer to section B.1 Digital I/Os, § Digital Outputs for detailed electrical description.

22.0 Appendix F: Motor Driver Cards

22.1 DC and Stepper Motor Driver XPS-DRV01





PIN	Function D.C. MOTOR	Function STEPPER MOTOR	PIN	Function D.C. MOTOR	Function STEPPER MOTOR
1	Tachometer+	Ph 1	15	Index	Index
2	Tachometer+	Ph 1	16	Limit ground	Limit ground
3	Tachometer-	Ph 2	17	+Travel limit	+Travel limit
4	Tachometer-	Ph 2	18	-Travel limit	-Travel limit
5	Motor+	Ph 3	19	Encoder A	Encoder A
6	Motor+	Ph 3	20	Encoder B	Encoder B
7	Motor-	Ph 4	21	+5V	+5V
8	Motor-	Ph 4	22	Encoder ground	Encoder ground
9	Reserved	Common 3&4	23	Encoder /A	Encoder /A
10	Reserved	Common 3&4	24	Encoder /B	Encoder /B
11	Reserved	Common 1&2	25	/index	/index
12	Reserved	Common 1&2			
13	Origin	Origin			
14	Shld GND	Shld GND			

Figure 71: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.

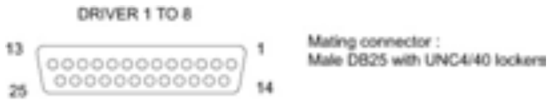
Shield GND Motor cable shield ground.

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.

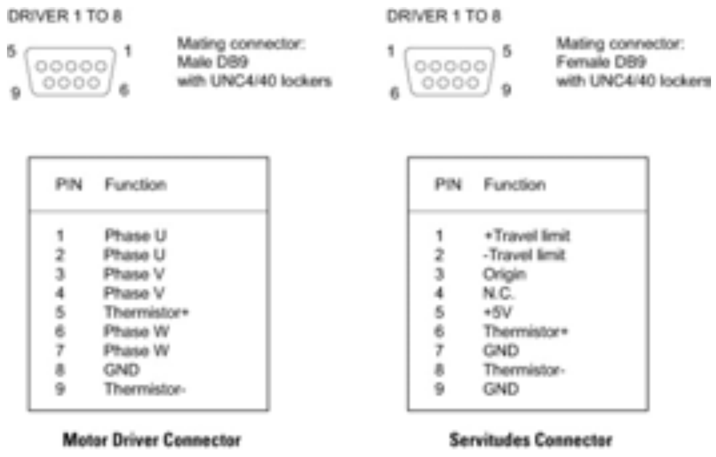
22.2 DC Motor Driver XPS-DRVM



PIN	Function	PIN	Function
1	Tachometer+	15	Index
2	Tachometer+	16	Limit ground
3	Tachometer-	17	+Travel limit
4	Tachometer-	18	-Travel limit
5	Motor+	19	Encoder A
6	Motor+	20	Encoder B
7	Motor-	21	+5V
8	Motor-	22	Encoder ground
9	Brake+	23	Encoder /A
10	N.C.	24	Encoder /B
11	Brake-	25	/index
12	N.C.		
13	Origin		
14	Shld GND		

Figure 72: XPS-DRVM Motor Driver Connectors.

22.3 Three phases AC brushless driver XPS-DRV02



PIN	Function
1	Phase U
2	Phase U
3	Phase V
4	Phase V
5	Thermistor+
6	Phase W
7	Phase W
8	GND
9	Thermistor-

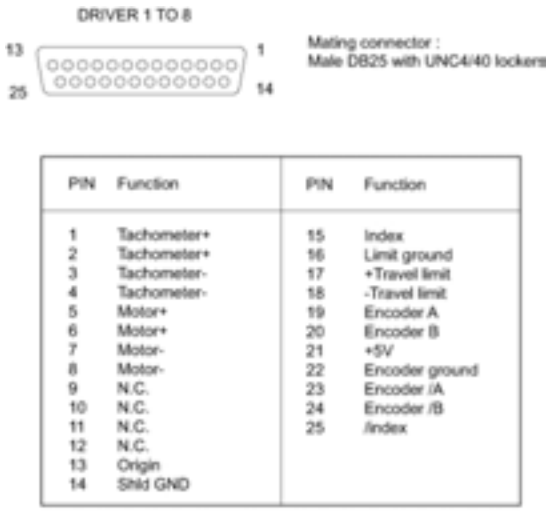
Motor Driver Connector

PIN	Function
1	+Travel limit
2	-Travel limit
3	Origin
4	N.C.
5	+5V
6	Thermistor+
7	GND
8	Thermistor-
9	GND

Servitudes Connector

Figure 73: XPS-DRV02 Motor Driver Connectors.

22.4 DC Motor Driver XPS-DRV03



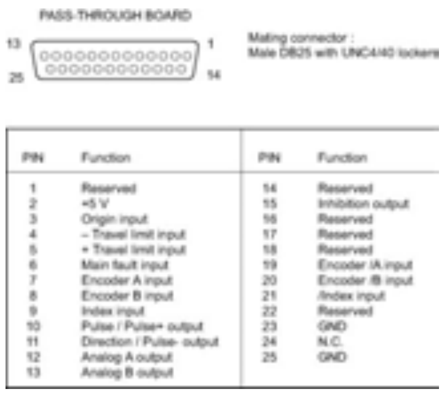
PIN	Function	PIN	Function
1	Tachometer+	15	Index
2	Tachometer+	16	Limit ground
3	Tachometer-	17	+Travel limit
4	Tachometer-	18	-Travel limit
5	Motor+	19	Encoder A
6	Motor+	20	Encoder B
7	Motor-	21	+5V
8	Motor-	22	Encoder ground
9	N.C.	23	Encoder /A
10	N.C.	24	Encoder /B
11	N.C.	25	/Index
12	N.C.		
13	Origin		
14	Shld GND		

Figure 74: XPS-DRV03 Motor Driver Connectors.

22.5 Pass-Through Board Connector (25-Pin D-Sub) XPS-DRV00

WARNING

This pass-through board connector takes the place of the motor interface connector only if this axis is connected to an external motor driver.

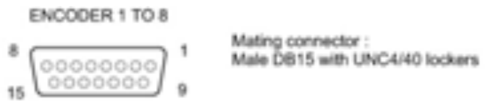


PIN	Function	PIN	Function
1	Reserved	14	Reserved
2	+5 V	15	Inhibition output
3	Origin input	16	Reserved
4	– Travel limit input	17	Reserved
5	+ Travel limit input	18	Reserved
6	Main fault input	19	Encoder /A input
7	Encoder A input	20	Encoder /B input
8	Encoder B input	21	/Index input
9	Index input	22	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 75: DRV00 Pass-Through Connector.

Analog A output and Analog B output have 16 bits resolution and are ±10 V output. These signals are used to command an external driver.

23.0 Appendix G: Analog Encoder Connector



PIN	Function	PIN	Function
1	Analog VA (0.5Vpp)	9	Analog /VA (0.5Vpp)
2	0V	10	0V
3	Analog VB (0.5Vpp)	11	Analog /VB (0.5Vpp)
4	+5VA	12	+5VL
5	N.C.	13	N.C.
6	Limit (TTL input)	14	Analog VI (0.5Vpp)
7	Analog /VI (0.5Vpp)	15	N.C.
8	Home (TTL input)		

Figure 76: 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 his complement entering in a differential amplifier (Sine = Analog VA - Analog /VA).

Analog VA, Analog /VA, Analog VB, Analog /VB, Analog VI and Analog /VI:

Levels for these signals must be 0.5 Vpp.

VA, /VA, VB and /VB inputs are the sine and cosine signals from the encoder glass scale.

VI and /VI inputs are used to receive Index information from the encoder glass scale.

+5 VA:

This +5 V DC supply is provided for supplying the encoder.

+5 VL:

This +5 V DC supply is provided for supplying digital circuits (Limit and Home).

Limit and Home are TTL inputs for Limit switch management and homing purposes directly from the encoder glass scale.

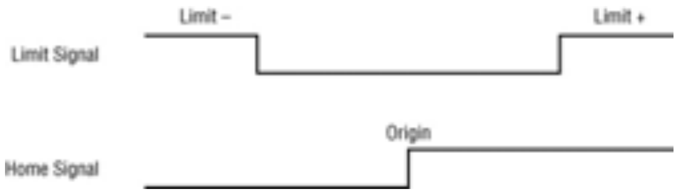
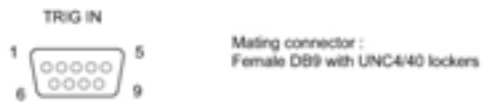
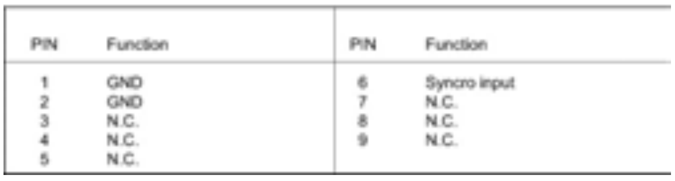


Figure 77: Heidenhain Servitude TTL Input Signals.

24.0 Appendix H: Trigger IN Connector





PIN	Function	PIN	Function
1	GND	6	Syncro input
2	GND	7	N.C.
3	N.C.	8	N.C.
4	N.C.	9	N.C.
5	N.C.		

Figure 78: Trigger Input Connector.

Synchro is a TTL input. It is used to trig the XPS controller acquisition (External gathering).

A low to high transition will latch all encoders and analog inputs inside the controller.

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®

Experience | Solutions

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: rma.service@newport.com

Europe
MICRO-CONTROLE Spectra-Physics S.A.S
1, rue Jules Guesde – Bât. B
ZI Bois de l'Épine – BP189
91006 Evry Cedex
France

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

Technical Support
e-mail: tech_europe@newport.com

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