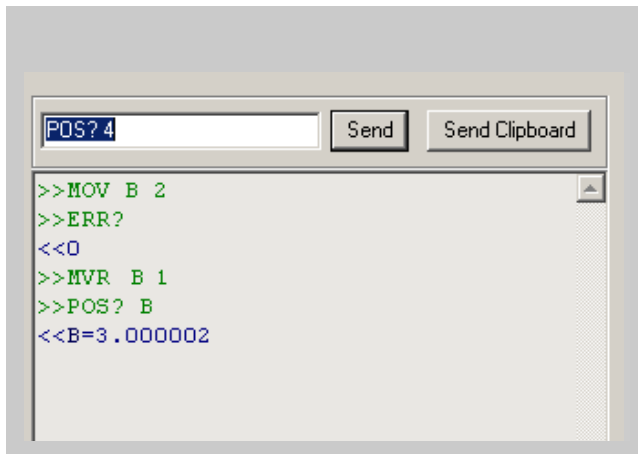


MS163E Software Manual

Mercury™ GCS Commands

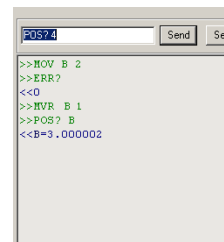
PI General Command Set

Release: 1.1.0 Date: 2009-08-05



This document describes software for use with the following products:

- C-663
Mercury™ Step Networkable Single-Axis Stepper Motor Controller
- C-863
Mercury™ Networkable Single-Axis DC-Motor Controller



Physik Instrumente (PI) GmbH & Co. KG is the owner of the following company names and trademarks: PI®, PIC®, PICMA®, PILine®, PIFOC®, PiezoWalk®, NEXACT®, NEXLINE®, NanoCube®, NanoAutomation®

The following designations are protected company names or registered trademarks of third parties:
Microsoft, Windows, LabView

Copyright 2009 by Physik Instrumente (PI) GmbH & Co. KG, Karlsruhe, Germany.
The text, photographs and drawings in this manual enjoy copyright protection. With regard thereto, Physik Instrumente (PI) GmbH & Co. KG reserves all rights. Use of said text, photographs and drawings is permitted only in part and only upon citation of the source.

Document Number MS163E, Release 1.1.0
Mercury_GCS_Commands_MS163E

Subject to change without notice. This manual is superseded by any new release. The newest release is available for download at www.pi.ws.

About This Document

Users of This Manual

This manual assumes that the reader has a fundamental understanding of basic servo systems, as well as motion control concepts and applicable safety procedures.

The manual describes the syntax of the PI General Command Set (GCS) and the individual commands supported by Mercury™ Class controllers. With present firmware, all software which accepts these commands must pass them to the controller via the Mercury™ Class GCS DLL or COM Server.

This document is available as PDF file on the product CD. Updated releases are available for download from www.pi.ws or by email: contact your Physik Instrumente Sales Engineer or write info@pi.ws.

Conventions

The notes and symbols used in this manual have the following meanings:



CAUTION

Calls attention to a procedure, practice, or condition which, if not correctly performed or adhered to, could result in damage to equipment.

NOTE

Provides additional information or application hints.

Related Documents

The Mercury™ controller and the software tools which might be delivered with the controller are described in their own manuals (see below). All documents are available as PDF files via download from the PI Website (www.pi.ws) or on the product CD. For updated releases or other versions contact your Physik Instrumente sales engineer or write info@pi.ws.

Hardware user manuals

Mercury™ GCSLabVIEW_MS149E
Mercury™ GCS DLL_MS154E
PIMikroMove User Manual SM148E
PIStageEditor_SM144E

Mercury™ Native Commands MS176E
MMCRun MS139E
Mercury™ Native DLL & LabVIEW MS177E

User Manual for each hardware component

LabView VIs based on PI GCS command set
Windows DLL Library (GCS commands)
PIMikroMove® Operating Software (GCS-based)
Software for managing GCS stage-data database

Native Mercury™ Commands
Mercury™ Operating Software (native commands)
Windows DLL Library and LabView VIs
(native-command-based)

Contents

1	Introduction	3
1.1	Software for GCS Commands.....	3
1.1.1	Software Overview	3
1.1.2	Installation.....	4
1.1.3	Updates	5
1.2	Axes and Stages	6
1.2.1	Axis Designators.....	6
1.2.2	I/O Line Designators.....	6
1.2.3	Controller Joystick Connections	7
1.3	Units and GCS	7
1.3.1	Hardware, Physical Units and Scaling	7
1.3.2	Rounding Considerations	8
2	Quick Start	9
3	Referencing	11
3.1	Reference Mode.....	11
3.2	Perform a Reference Move	11
3.3	Set Absolute Position	11
4	Joystick Control	12
5	Trackball Control	15
6	Macro Storage on Controller	16
6.1	Features and Restrictions	16
6.2	Macro Creation in GCS	16
6.3	Listing Stored Macros	19
6.4	Macro Translation & Listing Examples	19
7	System Details	21
7.1	Control Options—DC-Motor Controller Only	21
7.1.1	Position Control	21
7.1.2	Force Control	21
7.1.3	Position and Force Control	22
7.1.4	Notch Filter	23
7.1.5	Parameters and Commands for Closed-Loop Control	23
7.2	On-Target Detection.....	25

7.3	Limit Switches and “Soft Limits”	25
7.4	Input/Output Lines	26
7.4.1	Overview.....	26
7.4.2	Trigger Output.....	27
8	GCS Commands	28
8.1	Command Format	28
8.2	Command List (Alphabetical)	29
8.3	Command Reference (Alphabetical)	31
9	Motion Parameter Overview	69
9.1	Parameter Handling	69
9.2	Parameter Databases	70
9.3	Parameter List.....	71
9.4	Transmission Ratio and Scaling Factor	76
9.5	Travel Range Adjustment.....	77
9.6	Updating PISTages2.dat	80
10	Error Codes	82

1 Introduction

Mercury™ Class controllers include the C-663 Mercury Step™ open-loop, stepper motor controller as well as the C-863 Mercury™ DC-motor servo-controller.

GCS (General Command Set) is the PI standard command set, offering compatibility between different controllers. With current Mercury™ firmware, GCS command support is implemented by a Windows DLL which translates the GCS commands to the native commands the current firmware understands (for details, see the PI Mercury GCS DLL manual, MS154E). Most GCS commands and the corresponding DLL function calls are characterized by a three-letter mnemonic.

NOTES

Mixing native and GCS commands is not recommended. GCS move commands, for example, may not work properly after the position is changed by a native command.

GCS commands can be typed directly into the *Command entry* window of GCS-compatible PI user-interface software, like *PIMikroMove™* (see the *PIMikroMove™* manual on the product CD).

1.1 Software for GCS Commands

1.1.1 Software Overview

GCS commands are translated by the PI Mercury GCS DLL to the native commands the current Mercury™ firmware understands directly. With Mercury™ Class controllers, all motion of the connected motors and mechanical stages is software controlled. To offer maximum flexibility, GCS-based software interfaces at a number of different levels are provided and documented. Most of the individual programs and driver libraries are described in separate manuals. Updated releases are available on the PI Website or via email: contact your PI Sales Engineer or write info@pi.ws.

- *PITerminal* is a Windows program which can be used as a simple terminal with almost all PI controllers. It supports both direct and via-GCS-DLL connection to Mercury controllers via RS-232 and USB (the USB link also looks like a COM port to host software when the USB drivers are installed and the connection is active). *PITerminal* also handles controller selection in a Mercury™ network (by device numbers ranging from 1 to 16). When used without the GCS DLL,

firmware-native commands can be used with the connected controllers .

- *PIMikroMove™* (application for Microsoft Windows platforms) is operating software for this and many other PI controllers. With *PIMikroMove™* you can start your motion system—host PC, controller and stage(s)—immediately without writing customized software. *PIMikroMove™* offers motion control displays and features that in many cases make it unnecessary to deal with ASCII command formats. It also has a complete command input facility, which lets you experiment with various GCS commands easily. *PIMikroMove™* uses the GCS DLL described below to command the controller or controller network.
- GCS LabVIEW drivers to communicate with the Mercury™ controller from the National Instruments' LabVIEW environment (not included) using the Mercury™ GCS-DLL (see MS149E for details).
- GCS DLL (Windows DLL Library): The PI Mercury General Command Set Dynamic Link Library (PI Mercury GCS DLL) is an intermediate layer providing easy access to the controller from Windows programs. The use of the DLL and the functions it contains is described in a separate manual (MS154E). Most of the DLL functions correspond directly with the commands of the PI General Command Set.


NOTE

The GCS LabVIEW drivers and the GCS library are also available for Linux operating systems (kernel 2.6, GTK 2.0, glibc 2.4).

1.1.2 Installation

If you wish to use any of the GCS-based software, proceed as follows.

Windows operating systems:

- 1 Insert the Mercury™ CD in your host PC
- 2 If the Setup Wizard does not open automatically, start it from the root directory of the CD with the  icon
- 3 Follow the on-screen instructions and select the “typical” installation. Typical components are LabView drivers, GCS DLL, *PIMikroMove™*

Linux operating systems:

- 1 Insert the Mercury™ CD in the host PC
- 2 Open a terminal and go to the /linux directory on the Mercury™ CD
- 3 Log in as superuser (root)
- 4 Start the install script with ./INSTALL
Keep in mind the case sensitivity of Linux when typing the command
- 5 Follow the on-screen instructions. You can choose the individual components to install

If the installation fails, make sure you have installed the kernel header files for your kernel.

The PIStages2.dat stage database file needed by the GCS-based host software is installed in the ...\\PI\\GcsTranslator directory. In that directory, also the MercuryUserStages2.dat database will be located which is created automatically the first time you connect stages in the host software (i.e. the first time VST? or CST are used).

The location of the PI directory is that specified upon installation, by default C:\\Documents and Settings\\All Users\\Application Data (Windows XP) or C:\\ProgramData (Windows Vista). If this directory does not exist, the EXE file that needs the stage databases will look in its own directory.

NOTE

The PI host software is improved continually. It is therefore recommended that you visit the PI website (www.pi.ws) regularly to see if updated releases of the software are available for download. Updates are accompanied by information (readme files) so that you can decide if updating makes sense for your application. You need a password to see if updates are available and to download them. This password is provided on the Mercury™ CD in the Mercury Releasenews PDF file in the \\Manuals directory. See “Software Updates” in the controller User manual for download details.

1.1.3 Updates

For firmware and software updates please see the C-863 or C-663 User manual. The current firmware revision of your Mercury™ controller is contained in the response to the VER? command.

For updates of the PIStages2.dat stage database see “Updating PIStages2.dat” on p. 80.

1.2 Axes and Stages

Mercury™ Class controllers can be chained together on an RS-232 bus network and all controlled through one port of the host computer (USB or RS-232). On that network, the commands and responses are always sent between the host computer and one selected controller.

The GCS based software makes a network of Mercury™ controllers connected to one port look like one controller with up to 16 axes (if host's RS-232 port is used, number of usable axes may be limited to as few as 6 by current available). See the controller User Manual for information on setting the device number (1 to 16) of Mercury™ controllers using the address DIP switches on the front panel. The device number determines the default identifiers of the corresponding axes, I/O channels and joystick connections.

1.2.1 Axis Designators

By default the axes are named "A" to "P". The axis connected to the Mercury™ controller with device number 1 will be addressed as axis "A" with GCS commands, the Mercury™ No. 5 will provide axis "E", etc. If these two controllers are the only ones connected, only the two axes "A" and "E" are available.

The default identifiers can be changed using SAI (p. 57). The new identifiers must then be used with all axis commands and in macro names, even for macros that were previously stored using different names.

1.2.2 I/O Line Designators

Each Mercury™ and Mercury™ Step controller provides four digital input and four digital output lines on the "I/O" socket. These channels are named with the characters

```
ABCD EFGH IJKL MNOP QRST UVWX YZ12 3456 7890 @?>=
<:~ ^]\ [/.- ,+*) ('&% $#"!

```

in groups of 4, one group for each of the 16 possible controller addresses. Note that when the digital output line 4 of a Mercury™ controller is used with the CTO command to trigger other devices (pin 8 of the "I/O" socket, see User manual for pinout), its ID corresponds with the device number of the controller to which it belongs.

The four digital input lines of Mercury™ and Mercury™ Step controllers can also be used for analog input (0 to 5 V). In regard to analog input, these input channels have IDs from A1 to A64, again depending on the controller's address settings, and skipping values associated with any missing addresses.

Example: A network consists of a C-863 DC Motor Controller with device number 1 (DIP switches 1 to 4 are all ON) and a C-663 Stepper Controller

with device number 3 (DIP switches 1 to 4 are set ON ON OFF ON). In GCS commands the identifiers to be used then are as follows:

- Axes “A” and “C”
- Digital I/O using channel IDs A, B, C, D and I, J, K, L
- Analog input using channel IDs A1, A2, A3, A4 and A9, A10, A11, A12
- Trigger output channels: 1 and 3

1.2.3 Controller Joystick Connections

C-863 and C-663 Mercury™ controllers can be connected to an analog joystick device using their “Joystick” socket (see controller User manual for pinout). For that purpose, PI provides C-819.20 2-axis or C-819.30 3-axis joystick models. C-819.20 2-axis joystick devices have only one connection cable. Therefore an Y-cable (C-819.20Y) must be used to connect one axis and one logical button of the joystick to one controller and the other axis and other button to another controller. C-819.30 3-axis joystick devices are already equipped with separate connection cables for 3 different controllers. Since Mercury™ controllers support only one joystick axis and button, the corresponding identifiers to be used in GCS commands are always 1. The distinction between the individual axes and buttons is in fact made by the ID of the joystick device. The joystick device ID is identical to the device ID of the controller to which the joystick is connected (set with the DIP switches 1 to 4, can be 1 to 16).

Example: A network consists of a C-863 DC Motor Controller with device number 1 (DIP switches 1 to 4 are all ON) and a C-663 Stepper Controller with device number 3 (DIP switches 1 to 4 are set ON ON OFF ON). In GCS commands the identifiers to be used then are as follows:

- Joystick devices: 1 and 3
- Joystick axes: 1 and 1
- Joystick buttons: 1 and 1

1.3 Units and GCS

1.3.1 Hardware, Physical Units and Scaling

The GCS (General Command Set) system uses basic physical units of measure. The default conversion factors chosen to convert hardware-dependent units (e.g. encoder counts or steps) into millimeters or degrees, as appropriate (see SPA and SPA? command descriptions, parameters 0xE and 0xF) are found in the Pistages2.dat stage database. From there, they are transferred to the controller. An additional scale factor can be applied (see DFF command) to the basic physical unit making a

working physical unit available without overwriting the conversion factor for the first. This is the unit referred to by the term "physical unit" in the rest of this manual. See also Section "Transmission Ratio and Scaling Factor" on p. 76.

1.3.2 Rounding Considerations

When converting move commands in physical units to the hardware-dependent units required by the motion control layers, rounding errors can occur. The GCS software is so designed, that a relative move of x physical units will always result in a relative move of the same number of hardware units. Because of rounding errors, this means, for example, that 2 relative moves of x physical units may differ slightly from one relative move of $2x$. When making large numbers of relative moves, especially if moving back and forth, either intersperse absolute moves, or make sure that each relative move in one direction is matched by a relative move of the same size in the other direction.

Examples

With, for example, 5 hardware units = 33×10^{-6} physical units:

Relative moves:	cause	move of
smaller than 0.000003 physical units		0 hardware units
of 0.000004 to 0.000009 physical units		1 hardware unit
of 0.000010 to 0.000016 physical units		2 hardware units
of 0.000017 to 0.000023 physical units		3 hardware units
of 0.000024 to 0.000029 physical units		4 hardware units

Hence:

2 moves of 10×10^{-6} physical units followed by 1 move of 20×10^{-6} in the other direction cause a net motion of 1 hardware unit forward.

100 moves of 22×10^{-6} followed by 200 of -11×10^{-6} result in a net motion of -100 hardware units

5000 moves of 2×10^{-6} result in no motion

2 Quick Start

- 1 Set the device address:
Before power-on, verify correct setting of the address DIP switches (1 to 4) on the front panel of the Mercury™ controller.
Using a single controller, set its address to zero (switches 1 to 4 in ON (upper) position). Using multiple controllers in a daisy chain network, set each controller to a unique device address. Changes of the settings require power cycling the device. See the controller User manual for details.
- 2 Connect RS-232 or USB cable (not both!).
Connect the controller to a COM port or the USB bus on the host PC. Use the cables included.
If using multiple controllers, connect the first controller to the PC and daisy chain the other controllers via the short serial cables (C-862.CN).
Note that a USB connection will appear as an extra COM port when the controller is connected, powered up, and the USB drivers are installed from the product CD. Installing the USB drivers requires administrator rights on the host PC.
- 3 Stage connection:
Connect a suitable stage to the “DC Motor only” or “Stepper Motor only” socket. Do not connect DC-motor stages to C-663 stepper motor controllers, and do not connect stepper motor stages to C-863 DC motor controllers!
Connect the stage to the stage power supply if required.
- 4 Connect the controller power supply.

C-863 DC motor controllers:
+15 V is the nominal operating voltage (provided by the included C-890.PS power supply). The controller itself may work within the range of +12 to +30 V, but most of PI's motorized mechanics are equipped with 12 V DC-motors, so the +15 V supply will allow to operate the stages within their specifications. Using drives with 24 V DC-motors, a power supply with higher output voltage should be used.

C-663 stepper motor controllers:
+24 V is the nominal operating voltage. Use the included C-663.PS power supply.
- 5 Start the GCS-based Mercury™ operating *PIMikroMove™* software or the PITerminal. See “Starting Operation” in the controller User manual for the first steps with *PIMikroMove™*. Read more about *PIMikroMove™* in the *PIMikroMove™* manual (SM148E).

If you are working with the PITerminal, first use the CST command to determine which stage is connected to the Mercury™ controller, then use the INI command to initialize the stage and switch the servo on. When this is done, the stage must be referenced before you can make absolute moves with commands like MOV. By default, referencing must be done using the REF, MNL or MPL commands, depending on the connected mechanics. See “Referencing” on p. 11 for more information.

NOTES

A network of multiple Mercury™ controllers appears to the PI Mercury GCS DLL user as a single, multi-axis controller and is usually referred to in this manual as a “controller network”.

3 Referencing

Because the signals (encoder counts or motor steps) used for position determination provide only relative motion information, the controller cannot know the absolute position of an axis upon startup. This is why a referencing procedure is required before absolute target positions can be commanded and reached.

For the implementation of the referencing functionality in the individual host software components, see the appropriate manuals.

3.1 Reference Mode

The current reference mode setting of the controller (ask with RON?, p. 56) determines how referencing can be performed. In general, a reference move must be performed (see Section 3.2), but it is also possible to set absolute positions manually (see Section 3.3). To switch between the two reference modes, use the RON command (p. 56).

3.2 Perform a Reference Move

When the reference mode is set to "1", referencing is done by performing a reference move with REF (p. 55), MPL (p. 52), or MNL (p. 51).

NOTES

When referencing mode = "1" neither relative nor absolute targets can be commanded until referencing has been successfully performed.

REF requires that the axis have a reference switch (ask with REF?, p. 56), and MPL and MNL require that the axis have limit switches.

For best repeatability, always reference in the same way. The REF command always approaches the reference switch from the same side, no matter where the axis is when the command is issued.

When referencing mode = "0" only relative targets but no absolute targets can be commanded as long as referencing has not been successfully performed.

3.3 Set Absolute Position

When the reference mode is set to "0", referencing can be done by entering an absolute position value using the POS command (p. 54).

4 Joystick Control

CAUTION

Do not enable a joystick axis here when no joystick is connected to the controller hardware. Otherwise the corresponding controller axis may start moving and could damage your application setup.

Mercury™ motor controllers offer convenient manual motion control by using analog joysticks connected to the “Joystick” socket. When using an C-819.20 joystick device, its axes can be connected through the C-819.20Y cable to two compatible Mercury™ controllers. C-819.30 3-axis analog joysticks are equipped with separate connection cables for three controllers.

When a joystick is connected directly to the controller and enabled, it is the velocity of the motion axes that is determined by the displacement of the corresponding joystick axes. For each joystick axis there is a response table that defines the velocity response for a certain amplitude of the joystick axis. The values in a response table are factors which will during joystick control be applied to the velocity set with VEL (p. 65) for the controller axis, the range is -1.0 to 1.0. To change a response table, you can load default profiles provided by the controller, or you can write a custom profile to the response table point-by-point (256 values).

During joystick control, the target position is set to the travel range limits given by the Max_Travel_Range_pos and Max_Travel_Range_neg parameters (0x15 and 0x30, see “Travel Range Adjustment” (p. 77) for more information). When disabling a joystick, the target position is set to the current position for joystick-controlled axes.

NOTES

If the Y-cable is used to connect a joystick to two controllers, the X-branch must be connected to a powered-up controller because joystick power is taken from that side.

Before a joystick can be operated correctly, a calibration routine may need to be performed. Activating the joystick before calibration may cause the motor to start moving even though the joystick is in the neutral position.

To calibrate the joystick X- or Y-axis turn the corresponding “Adjust” knob on the joystick until the motor stops.

To calibrate the joystick Z-axis of C-819.30 3-axis joystick devices, a special procedure is required. See the Mercury™ User manual for details.

Commands for joystick handling:

Command	Function
JON (p. 46)	<p>Enables or disables a specified joystick axis for joystick operation.</p> <p>While a joystick is active on a controller axis, move commands are neither accepted from the command line nor from macros for that axis, and no trackball control is accepted.</p>
JON? (p. 47)	Queries the current activation state of the joystick axes.
JAX? (p. 42)	Queries the current assignment of controller axes to joystick axes.
JAS? (p. 41)	Queries the current status of joystick axes. The response corresponds to the current displacement of the joystick axis and is the factor which is currently applied to the current valid velocity setting of the controlled motion axis, according to the response table.
JBS? (p. 42)	Queries the current status of joystick buttons. The response indicates if the joystick button is pressed; 0 = not pressed, 1 = pressed.
JDT (p. 43))	<p>Sets default response tables for the joystick axes.</p> <p>The current valid response table for the specified joystick axis is overwritten by the selection made with JDT. To ensure that the loaded response table is applied, use JDT before you enable joystick operation with JON.</p> <p>Mercury controllers provides the following default response tables for special applications:</p> <p>1 = linear 3 = cubic</p>
JLT (p. 44)	Fills the response table for a joystick axis point-by-point. This overwrites the current valid response table content for this joystick axis. To ensure that the loaded response table is applied, use JLT before you enable joystick operation with JON.
JLT? (p. 45)	Reads the current valid response table values for the joystick axes.
TNJ? (p. 63)	Tell number of joystick axes. Note: the software can not determine if a joystick is actually connected to a Mercury™ controller. This is the maximum possible number of joystick axes that can be connected to the network.

Parameters for joystick handling (can be changed with SPA):

Parameter ID	Function
0x120 / 288	Joystick offset Gives an offset to the analog input provided by the joystick. This allows to correct the neutral (center) value of the joystick reading, e.g. if the actual joystick reading is 120, with bias 8 the reading used would be 128.
0x49 / 73	Current closed-loop velocity (user unit/s) Gives the current velocity, limited by parameter 0xA. Can also be changed with VEL. The values in a response table are factors which will during joystick control be applied to the velocity set with VEL for the controller axis, the range is -1.0 to 1.0.

5 Trackball Control

A trackball device can be used to set the target position. Connect the digital TTL signals A and B (also referred to as quadrature signals) provided by the trackball to the digital input lines 3 and 4 of the "I/O" socket (pinout see User manual) on the Mercury™ controller. The lines are terminated by 10k to GND.

Each signal transition shifts the target position by a given number of counts in positive or negative direction. This makes possible extremely fine motion control with high position resolution.

Parameters for trackball control (can be changed with SPA):

Parameter ID	Function
0x100 / 256	Trackball mode 0 = trackball disabled 1 = trackball enabled While the trackball mode is active, move commands or joystick control are not accepted.
0x101 / 257	Trackball increment (counts) Gives the increment length for trackball operation, minimum value = 1. Each signal transition on the trackball lines will shift the target by the given number of counts if the trackball is enabled by parameter 0x100.
0x102 / 258	Trackball filter (C-863 DC motor versions only) 0 to 255 The parameter determines how often the same signal level must be read before the signal transition on the trackball lines is accepted. This suppresses transient noise and allows for stable reading. If 0, the filter is disabled.

6 Macro Storage on Controller

Software that uses the Mercury™ GCS DLL can take advantage of the GCS Macro Architecture. However, because controller macros are stored in the command language of the controller, the DLL must translate each complete GCS macro to a non-GCS native macro before sending anything to the controller. Details of the native command macro architecture are given in the Mercury™ Native Commands manual, MS176.

6.1 Features and Restrictions

The hardware macro storage capability has the following features, which result in certain restrictions:

- Each macro can contain up to 16 such commands
- The macros are identified by numbers 0 to 31
- Macro 0, if defined, is the autostart macro, which is executed automatically upon power-up or reset
- Macros are executed on the controller where they are stored, so commands in a macro may address only the axis and/or I/O channels associated with that controller (there is no command-interface communication between controllers). Interaction between separate axes is conceivable only through suitable programming and hardwiring of I/O lines
- The position values stored in the macros are in counts or (micro)steps. This means that a macro may not work properly if run when different stage types are connected to the controller. A different stage could have a very different travel ratio and thus move to a position far different from the one intended.

6.2 Macro Creation in GCS

The GCS macro creation mechanism involves placing a GCS controller in macro-recording mode, sending it commands, and then exiting macro recording mode. While in macro-recording mode, the controller neither executes nor responds to commands, but simply stores them in the macro.

Macro Translation

In normal operation, the GCS DLL translates GCS-command-based functions to Mercury™ native commands. The GCS macro-recording mechanism is easily translated to native commands with the use of a macro-recording flag in the DLL. While the flag is set, DLL function calls

create native commands as usual but they are saved rather than sent to the controller. When recording is completed (MAC END), the saved commands are assembled into a compound command beginning with MD, given a cursory check, and, if they are acceptable, the macro definition compound command is sent to the controller.

Here are some of the implications:

- The DLL may decide not to send the macro to the controller at all. Whether or not the macro was sent can be checked with ERR? after MAC END: If the macro was not sent, error -1010 will be set. (Admittedly, the error-description text can be misleading)
- Referencing with REF is allowed, because with the Mercury™ native command set it is possible to tell how to move toward or away from the reference switch, but because REF is not implemented as single commands in the native command set, it will occupy more than one command slot in the macro (see examples below).
- A total of only 32 (native) commands may be stored in a macro on a Mercury™ Class controller. That means that when using GCS commands which translate to multiple native commands (e.g. REF, INI), little space may be left for other commands.
- The way in which a GCS command is translated into a native command can depend on the stage connected and how it was referenced. A macro made under one set of conditions will not function properly if run under others*. As a result:
 - Macros are only valid for the stage type that was connected when the macro was created.
 - Only relative moves can be used without concern in macros
 - Absolute moves require the axis to have been referenced with exactly the same sequence of referencing commands when the macro is run as when it was created. (Note that having the software save positions at shutdown and restore them from saved values involves RON/POS referencing.)**
- The macro names used at the GCS level are assigned using the following strict convention: aMC0nn where a is the current axis designator associated with the controller and nn is a two-digit number between 00 and 31. In addition, all the MAC commands take an axis designator as an argument. The macros AMC000, BMC000, etc. (for axes A, B,..., respectively)

*For example, position values in millimeters or degrees in GCS motion commands are converted to steps or counts. The values are calculated when the macro is created using the parameters for the stage configured on the corresponding axis (controller).

**Because it is not possible to set the current absolute position to a desired value but only to 0, the count values in the controller's internal position counter after a GCS move to a given position may be very different depending on how the axis was referenced (with REF, MNL, MPL or a RON/POS combination),

are the autostart macros; they are executed automatically upon startup or reset of the individual axis controller. The name thus already specifies the axis which the macro addresses.

- Only the following GCS commands are allowable when the macro recording flag is set. Use of a disallowed command will cause the next MAC END to set an error
 - BRA
 - DEL
 - DFH
 - DIO
 - GOH
 - HLT
 - INI (generates a large number of native commands in the macro, see below)
 - JON
 - MAC START (macro called must reside on the same controller)
 - MEX DIO? <ch> =
 - MEX JBS? <joystk> 1 =
 - MVR
 - REF (generates a large number of native commands in the macro, see below)
 - SPA
 - Access to the following SPA parameters by macros is permitted: all others will be ignored (parameter IDs in hexadecimal / decimal format):
 - 0x1 / 1: P-Term
 - 0x2 / 2: I-Term
 - 0x3 / 3: D-Term
 - 0x4 / 4: I-Limit
 - 0x8 / 8: Max.Position Error
 - 0xA / 10: Max. Velocity
 - 0xB / 11: Max Acceleration (muss >200 sein)
 - 0x1824: Limit Switch Mode
 - 0x32 / 50: No Limit Switch
 - 0x40 / 64: Stepper motor hold current (HC native command) in mA
 - 0x41 / 65: Stepper motor drive current (DC native command) in mA
 - 0x42 / 66: Stepper motor hold time (HT native command) in ms
 - STP
 - SVO
 - VEL
 - WAC ONT? <axis> = 1
 - WAC DIO? <ch> =

6.3 Listing Stored Macros

When the MAC? command is used with a macro name to list the contents of a macro, the native commands stored on the unit are translated back to GCS commands, with all the implications that entails.

Functions that cause several native commands to be stored in the macro may not be recognized when the macro is listed, making it possible to see the GCS versions of the individual functions (see INI example below).

The native-command versions can, of course, always be listed by send the native command TMn or TZ (Tell Macro n, Tell Macro Zero) with Mercury™_Sendnongcsstring() DLL function (see Native Commands manual for details).

Native commands that have no equivalent in GCS (e.g. FE3) are listed in their original form as follows:

“<non GCS: FE3>”

6.4 Macro Translation & Listing Examples

INI

When converted to native commands, INI is separated into all of its separate functions; when the stored macro is listed with MAC? they are shown as a long list of separate GCS commands. From the list it is obvious that when INI is used, not many commands are left before the macro is full. With an M-505.4PD, the dialog in which a macro containing INI is stored and then listed can look as follows:

```
>>CST DM-505.4PD
>>ERR?
<<0
```

```
>>MAC BEG DMC003
>>INI D
>>MAC END
>>ERR?
<<0
```

```
>>MAC? DMC003
<<SPA D50 0
<<SPA D24 0
<<BRA D0
<<SPA D1 200
<<SPA D2 150
<<SPA D3 100
<<SPA D8 2000
<<SPA D4 2000
```

```
<<SVO D1  
<<VEL D25  
<<SPA D11 4000000  
<<STP
```

REF

Similarly, REF A, is stored as the following sequence (shown this time in the native command set):

```
"SV40000,FE2,WS,MR-40000,WS,FE,WS,SV100000"
```

This sequence, when read with MAC?, is recognized by the DLL and translated back to "REF A", obscuring the fact that it occupies 8 of the 16 possible command slots. It can thus be seen, that INI and REF will not both fit in the same macro!

MVR

The relative move sizes entered with MVR and converted into counts using the parameters of the currently configured stage before being stored. So, if a macro containing MVR A2 is created with an M-111.2DG configured on axis A and later an M-505.4PD is configured for A with CST, the macro will read out as MVR A 58.2542.

7 System Details

7.1 Control Options—DC-Motor Controller Only

The C-863 Mercury™ DC-motor controller features closed-loop control of the connected mechanics (= operation with servo ON). Closed-loop control can be enabled and disabled with the SVO command (p. 60).

The following control options can be selected using the SPA command (p. 58) with certain parameters:

- **Position control**, power-on default, see p. 21
- **Force control** (with firmware revision 2.21 and newer), see p. 21
- **Position and force control** (with firmware revision 2.21 and newer), see p. 22

7.1.1 Position Control

Position control is the power-on default. The control variable is the position of the stage. The sensor feedback used by the control-loop is that of the incremental position encoder in the stage.

The position-control-loop features dedicated parameters (P-, I-, D-term, I-limit) which can be set by parameters. See Section 7.1.5 on p. 23 for an overview.



Fig. 1: Position control algorithm, with sensor feedback (position encoder in the stage) and PID correction

7.1.2 Force Control

With firmware revision 2.21 and newer, it is possible to control the force or another alternative variable measured by an additional sensor connected to the “Joystick” socket, e.g. a force sensor.

This option can be used, for example, to maintain the force the stage imposes upon an object. If the force increases, the stage moves back, and if the force decreases, the stage moves forward until the current force matches

the target force again.

Sensor signal range: -10 V to +10 V

Sensor signal input: "Joystick" socket, pin 2

Resolution: 12 bit

The force control-loop features dedicated parameters (P-, I-, D-terms, I-limit) which can be set via parameters.

The low pass filter for the sensor input can also be set via a parameter. The frequency range is 40 Hz to 10,000 Hz.

See Section 7.1.5 on p. 23 for an overview.

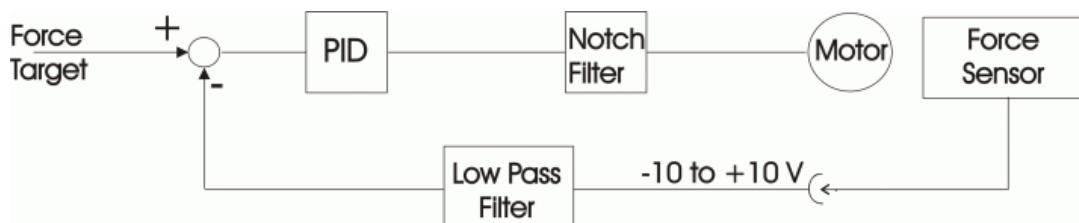


Fig. 2: Force control algorithm with PID correction and notch filter. The feedback of the additional (force) sensor is corrected by a low pass filter.

7.1.3 Position and Force Control

With firmware revision 2.21 and newer, position control with an embedded (force) control-loop is possible, i.e. the position and an additional variable are controlled. The sensor feedback used for position-control is that of an incremental position encoder in the mechanics, while the sensor feedback used for the embedded control-loop is that of an additional sensor connected to the "Joystick" socket, e.g. a force sensor. See Sections 7.1.1 and 7.1.2 for further details.

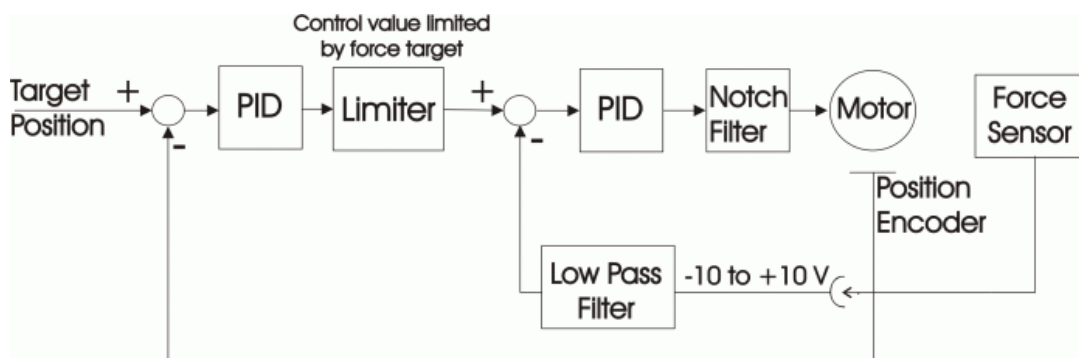


Fig. 3: Control algorithm maintaining position and force, with feedback of position encoder and additional (force) sensor; two control loops with separate PID corrections; the force target is used as limit for the control value resulting from the position control loop

7.1.4 Notch Filter

Mechanical resonances of the system exaggerate the response to certain frequencies in the control signal. The C-863 has a notch filter on the output of the control-loop to compensate for resonances in the mechanics by reducing the corresponding frequency components in the control signal. The frequency range for the notch filter is 40 Hz to 10,000 Hz. Notch filter frequency and edge can be set via parameters (see Section 7.1.5 on p. 23 for an overview). Default values are 10,000 Hz and a notch edge of 0.8.

7.1.5 Parameters and Commands for Closed-Loop Control

Parameters for closed-loop control (can be changed with SPA):

Parameter ID	Function
0x1 / 1	P-term for position control 0 to 32767
0x2 / 2	I-term for position control 0 to 32767
0x3 / 3	D-term for position control 0 to 32767
0x4 / 4	I-limit for position control 0 to 32767
0x8 / 8	Maximum position error (user unit) 0 to 32767 Used for stall detection. If the position error (i.e. the absolute value of the difference between current position and commanded position) in closed-loop operation exceeds the given maximum, the PI Mercury GCS DLL sets error code -1024 ("Motion error"), the servo will be switched off and the axis stops.
0x94 / 148	Notch filter frequency (Hz) 40 to 10000 The corresponding frequency component in the control value is reduced to compensate for unwanted resonances in the mechanics. Only active in closed-loop operation. Should normally not be changed (try to change only with very high loads).
0x95 / 149	Notch filter edge 0.4 to 10 Gives the slope of the filter edge. Do not change.
0x110 / 272	Control mode Gives the control mode to be applied, see C-863 User manual for details 0 = Position control (force control is OFF) 1 = Force control (position control is OFF) 2 = Position control and force control are ONs

0x111 / 273	Force control target 0 to 1024 Used in “force control” or “position and force control” mode where it defines the force target that shall be reached and maintained by varying the position.
0x112 / 274	P-term for force control algorithm 0 to 65,535
0x113 / 275	I-term for force control algorithm 0 to 65,535
0x114 / 276	I-limit for force control algorithm 0 to 65,535
0x115 / 277	D-term for force control algorithm 0 to 65,535
0x116 / 278	Offset to force sensor input Gives an offset to the input of the additional (force) sensor. Note that this offset is already included if you query the current value of the additional sensor via parameter 0x11A.
0x117 / 279	Low pass filter for force sensor input (Hz) 40 to 10000 Gives the frequency of the low pass filter on the input of the additional (force) sensor used for the “force control” or “position and force control” modes.
0x11A / 282	Current force read-only parameter -2048 to 2048 Current value of the additional sensor used for the force control loop, i.e. the input signal on pin 2 of the "Joystick" socket. The input voltage range is -10 to +10 V. The voltage value is converted by a 12 bit A/D converter. The value reported includes the offset set with parameter 0x116, and was already filtered by the low pass filter set with parameter 0x117. Furthermore, any sign settings made with the FS (Set Force Sign) command of the native command set are already applied to the signal (see MercuryNativeCommands Manual MS176E for details).

Commands for closed-loop control:

Command	Function
SVO (p. 60)	Sets servo-control mode on or off for given axes (enable/disable closed-loop control).
SVO? (p. 61)	Gets servo-control mode for given axes.
MOV (p. 51)	Set new absolute target position for given axis.
MVR (p. 53)	Move given axes relative to their current position. The new target position is calculated by adding the given value to the last -commanded target value.
POS? (p. 55)	Returns the current position (of encoder)
MOV? (p. 52)	Returns last valid commanded target position.

7.2 On-Target Detection

By default, the on-target flag is set when the motion trajectory has finished. It is assumed that the motor then has reached the target position. But in fact, the motor can still be in the physical settling process so that the actual motor position may not yet match the target with count accuracy.

With C-863 DC motor controllers, you can combine the on-target flag with the physical motor position. To do this, set parameters for settle window (ID 0x36) and settle time (ID 0x3F) to define the appropriate conditions. The settle window defined is centered around the target position. The on-target status becomes "true" when the actual position stays in this window for at least the settle time. By default, the settle time is set to 0 which means that the axis is on target when the calculated motion trajectory has finished, irrespective of the settle window settings.

To report the on target state, use the ONT? command (p. 54).

Parameters for on-target detection (can be changed with SPA):

Parameter ID	Function
0x36 / 54	Settle window (counts) 0 to 2^{31}
0x3F / 63	Settle time (s) 0.000 to 1.000 s

7.3 Limit Switches and "Soft Limits"

During operation, limit sensors (switches) can be used to stop motion at the end of the allowable travel range. Each of the two switches will interrupt motion in a particular direction. If positioning equipment from PI is used, the limit switches or sensors are pre-wired for operation with compatible Mercury™ controllers.

The Mercury™ controller can be configured to accept either an active-high or an active-low stop signal from the limit sensors (both must be the same). On the C-663 and C-863, the limit switch signals are interpreted by both the controller hardware and software. DIP switch 7 changes the hardware interlock between active-high (OFF) and active-low (ON).

NOTE

If the hardware and software limit switch configurations are not compatible, no motion is possible. Standard PI stepper motor stages have active-low limit switches, whereas PI DC motor stages have active-high limit switches.

Using certain parameters, it is possible to set "soft limits" which establish a "safety distance" which the stage will not enter on both ends of the travel range. See "Travel Range Adjustment" on p. 77 for details.

7.4 Input/Output Lines

7.4.1 Overview

Mercury™ controllers offer 4 digital outputs and 4 inputs for either digital or (8-bit) analog use. A set of commands is available to handle these I/O lines: See "I/O Line Designators" on p. 6 for the identifiers to be used in the commands.

Command	Function
CTO (p. 33)	Configures the trigger output conditions for digital output line 4 (pin 8 of the "I/O" socket)
CTO? (p. 34)	Get the trigger output configuration for digital output line 4 (pin 8 of the "I/O" socket)
DIO (p. 37)	Set digital output channels "high" or "low". Do not use DIO on any physical trigger line for which trigger output is enabled with TRO.
DIO? (p. 37)	Get the states the digital input channel(s).
MEX (p. 49)	Stop macro execution due to a given condition. Can only be used in macros.
TAC? (p. 61)	Get the number of installed analog channels. With Mercury™ controllers, the response contains only the analog channels on the I/O connector.
TAV? (p. 61)	Read analog input.
TIO? (p. 62)	Get the number of digital input and output channels installed.
TRO (p. 63)	Enables or disables the trigger output mode which was set with CTO. Do not use DIO on any physical trigger line for which trigger output is enabled with TRO.
TRO? (p. 64)	Gets trigger output-mode enable-status for given trigger output line (the trigger output configuration is made with CTO).
WAC (p. 66)	Wait until a given condition occurs. Can only be used in macros.

Note that the MEX and WAC commands (p. 49 resp. p. 66) allow conditional exits from macros depending on the digital and joystick inputs. This makes possible pushbutton control and/or interaxis communication in stand-alone mode, i.e. without a PC connection.

7.4.2 Trigger Output

You can program the digital output line 4 of a Mercury™ controller to trigger other devices (pin 8 of the “I/O” socket, see User manual for pinout). Note that the ID to be used for the line corresponds with the device number of the controller to which it belongs.

The trigger mode selection and the settings for trigger position(s) can be made by command.

Command	Function
CTO (p. 33)	Configures the trigger output conditions for digital output line 4 (pin 8 of the “I/O” socket). The following trigger modes are available: 0 = PositionDistance: A trigger pulse is written whenever the axis has covered a given distance. 7 = Position + Offset: The first trigger pulse is written when the axis has reached a certain (trigger) position. The next trigger pulses each are written when the axis position equals the sum of the last valid trigger position and the given distance (increment) value. 8 = SingleTrigger: A trigger pulse is written when the axis has reached a given (trigger) position.
CTO? (p. 34)	Get the trigger output configuration for digital output line 4 (pin 8 of the “I/O” socket)
TRO (p. 63)	Enables or disables the trigger output mode which was set with CTO. Do not use DIO on any physical trigger line for which trigger output is enabled with TRO.
TRO? (p. 64)	Gets trigger output-mode enable-status for given trigger output line (the trigger output configuration is made with CTO).

Example: The trigger output line of the controller with device ID 1 is to be configured so that multiple trigger pulses are output using the “Position+Offset” trigger mode. The distance (increment value) for the individual trigger pulses is to be 2, the position where the first trigger pulse is to be output is 0. Send:

```
CTO 1 1 2 1 3 7 1 10 0
TRO 1 1
```

to change the trigger output configuration and to enable trigger output. The arguments of the CTO command are marked with different colors to clarify the individual sections: the first section sets the distance, the second sets the trigger mode, and the third section sets the position where the first trigger point is to be output. When motion of axis A ist started, the trigger pulses will be output at the appropriate positions.

8 GCS Commands

8.1 Command Format

GCS ASCII Commands have the format below. Exceptions are the single-character binary commands on p. 67 ff.

CMD **[SP]**X**[SP]**sV.V**[{[SP]X[SP]sV.V}]... [LF]**

where:

- CMD token (mnemonic) of the specific command
- [SP]** one space (ASCII char #32), can be omitted between the item identifier and the (signed) parameter
- X item identifier (see p. 6),
- s sign (positive values can be transmitted without sign)
- V.V parameter, values are doubles (double precision) or integers, depending on the command.
- [...] Square brackets “[]” indicate an optional entry or parameter.
- {...} Braces “{ }” indicate a repetition of parameters, i.e. that it is possible to access more than one item (e.g. several axes) in one command line.
- [LF]** LineFeed (Char #10).

Example:

Send: MOV**[SP]**A10.0**[SP]**B5.0

Moves axis A to position 10.0 mm and axis B to 5.0 mm

Format of answers:

Some commands deliver a report message having the following format:

X=sV.V**[LF]**

where:

- X item identifier (see p. 6)
- s sign (positive values are transmitted without sign)
- V.V parameter, values are doubles or integers depending on the command
- [LF]** LineFeed (ASCII char #10).

Example:

Send: POS? **[SP]**AB**[LF]**

Report: A=10.0000**[SP]** **[LF]**
B=5.0000**[LF]**

There is one space (**[SP]**, ASCII char #32) before the LineFeed character on all lines of the response *except* the last line.

The individual spaces and linefeed characters will not all be marked in the rest of this manual.

Floating Point Data Format

Some commands require parameters in floating point format. The following syntax is possible for these arguments:

```
sv
sv.v
sv.vEsxxx
```

where:

s	sign(positive values can be without sign)
v	integer parameter, will be converted into float by firmware
v.v	float parameter, the decimal separator must be a dot (.), not a comma (,)
E	exponent character
xxx	exponent value

The format in which floating point values are reported (output) is always:

```
sv.vvvv
```

where:

s	sign (positive values are reported without sign)
v.vvvvvv	the number of digits after the decimal point may vary

If the reply includes more than 2 floats, each will occupy one line.

8.2 Command List (Alphabetical)

- *IDN? (Get Identity), p. 31
- BRA (Set brake on or off), p. 31
- BRA? (Ask if axis has brakes), p. 31
- CST (Change Stage), p. 32
- CST? (get stagename), p. 32
- CTO (set trigger parameter), p. 33
- CTO? (get trigger parameter), p. 34
- DEL (Delay execution of macro), p. 35
- DFF (DeFine Factor), p. 35
- DFF? (get factor), p. 36
- DFH (DeFine Home), p. 36
- DFH? (get home positions), p. 37
- DIO (set Digital Output), p. 37
- DIO? (get Digital Output), p. 37
- ERR? (get ERRor), p. 38
- GOH (GO Home), p. 39
- HLP? (HeLP), p. 39
- HLT (HaLT), p. 40

HPA? (display parameter help message), p. 40
INI (INItialization), p. 41
JAS? (get joystick value), p. 41
JAX? (List joystick to motion-axis assignments), p. 42
JBS? (query if joystick button is pressed), p. 42
JDT (load joystick table), p. 43
JLT (fill joystick response table), p. 44
JLT? (query active joystick table), p. 45
JON (activate/deactivate joystick), p. 46
JON?(get joystick activation status), p. 47
LIM? (Indicates whether axes have limit switches), p. 47
MAC (macro), p. 47
MAC? (list macro), p. 49
MEX (Stop macro execution if specified condition true), p. 49
MNL (Move to Negative Limit), p. 51
MOV (MOVE absolute), p. 51
MOV? (read target position), p. 52
MPL (Move to Positive Limit), p. 52
MVR (MoVe relatiVe), p. 53
ONT? (axis ON Target), p. 54
POS (set real position), p. 54
POS? (read real POSition), p. 55
REF (move to REFerence position), p. 55
REF? (Lists axes which have a reference sensor), p. 56
RON (set Reference mode ON | off), p. 56
RON? (get Reference mode), p. 56
SAI (Set Axis Identifier), p. 57
SAI? (get axis identifier), p. 57
SPA (Set Parameter), p. 58
SPA? (Get Parameter), p. 58
SRG? (Read register), p. 59
STP (Stop Motion), p. 60
SVO (set SerVO on or off), p. 60
SVO? (Get servo status), p. 61
TAC? (Tell Analog Channels), p. 61
TAV? (Get Analog Input), p. 61
TIO? (Tell Digital I/Os), p. 62
TMN? (Tell Minimum Travel Value), p. 62
TMX? (Tell Maximum Travel Value), p. 63
TNJ? Tell number of joysticks), p. 63
TRO (enable/disable trigger output), p. 63
TRO? (get trigger output enable status), p. 64
TVI? (Tell Valid axis Identifiers) p. 64
VEL (Set Velocity), p. 65
VEL? (Get Velocity), p. 65
VER? (Get Version), p. 66
VST? (Get available Stages), p. 66
WAC (Wait for condition), p. 66
#5, (Poll Motion Status), p. 67
#7, (Controller Ready?), p. 67
#8, (Macro running?), p. 68
#24, (Stop), p. 68

8.3 Command Reference (Alphabetical)

*IDN? (Get Identity Number)

Description:	Reports an identity string of the PI Mercury GCS DLL
Format:	*IDN?
Arguments:	none
Response:	One-line string terminated by line feed, e.g.: Physik Instrumente,Mercury GCS Network,,1.9.2.4

BRA (Set brake on or off)

Description:	Sets BRAke on or off for an axis. Power-up factory default is ON; brake set to OFF (brake control line high) by INI, as brake ON disables motors of some stages (even if stage has no brake).
Format:	BRA <AxisID> <uint>[{ <AxisID> <uint>}]
Arguments:	<AxisID>: is a valid axis identifier <uint>: if 0 = set brake off, if 1 = set brake on
Response:	none
Troubleshooting:	Axis has no brake

BRA? (Ask if axis has brakes)

Description:	Lists the axes with brakes.
Format:	BRA?
Arguments:	none
Response:	[{ <AxisID>}] where <AxisID> are the identifiers of axes with brakes, e.g.: AC If no axis has a brake, the answer is an empty line.

CST (Change STage)

Description:	Assigns axes to stages. With this command the stage assignment of the connected axes can be changed. Valid stage names can be listed with VST? (p. 66). If no stage is connected, stage name should be "NOSTAGE".	
Format:	CST <AxisID> <stagename>[{ <AxisID> <stagename>}]	
Arguments:	<AxisID>: is a valid axis identifier <stagename>: name of the stage connected to the axis	
Response:	none	
Troubleshooting:	Unknown stage name	
Example:	Send:	CST A M-403.62S B M-110.1DG
	Note:	Assigns a stage of type M-403.62S to axis A and of type M-110.1DG to axis B
	Send:	SAI?
	Receive:	B
		A
	Send:	CST B NOSTAGE
	Note:	Disconnects axis B
	Send:	SAI?
	Receive:	A
	Send:	POS? B
	Send:	ERR?
	Receive:	200
	Note:	PI_CNTR_NO_AXIS
	Send:	GÖH
	Note:	Moves axis A (not axis B).
	Send:	CST?
	Receive:	A=M-403.62S
		B=NOSTAGE

CST? (get stage name)

Description:	Returns the name of the C onnected S Tage for queried axes. CST? will always return all axes, i.e. the answer also includes the axes set to NOSTAGE (see CST, p. 32). In contrast to this, SAI? (p. 57) will only return the axes which are assigned to stages.
Format:	CST? [{<AxisID>}]

Arguments: <AxisID>: is a valid axis identifier, if omitted all axes are queried

Response: {<axis>="<string> LF}
 where
 <string> is a stage name, i.e. the name of the stage assigned to an axis. Unconfigured axes will show the stage name "NOSTAGE".

CTO (Set Trigger Parameter)

Description: Configures the trigger output conditions for the given trigger output line. On Mercury™ controllers, the digital output line 4 can be configured for trigger output (pin 8 of the "I/O" socket, see User manual for pinout).

The trigger output conditions will become active when activated with TRO. Do not use DIO on any physical trigger line for which trigger output is enabled with TRO.

Format: CTO <TriggerOut> <uint> <ParamValue>
 [{<TriggerOut> <uint> <ParamValue>}]

Arguments: <TriggerOut> is one trigger output line of a Mercury™ network. Note that the ID of a trigger output line corresponds with the device number of the controller to which it belongs (the controller device number is set with DIP switches 1 to 4 on the controller front panel)

<uint> is the CTO parameter ID in decimal format:

- 1 = TriggerStep
- 2 = Axis
- 3 = TriggerMode
- 10 = TriggerPosition

<ParamValue> is the value to which the CTO parameter is set:

for TriggerStep: step size in physical units

for Axis: the axis to connect to the trigger output line. The assignment is fixed (the axis identifier must correspond to the controller address, see "Axis Designators" on p. 6 for details).

for TriggerMode:

- 0 = PositionDistance: A trigger pulse is written whenever the axis has covered the

distance given by TriggerStep.

7 = Position + Offset: The first trigger pulse is written when the axis has reached the trigger position given by TriggerPosition. The next trigger pulses each are written when the axis position equals the sum of the last valid trigger position and the increment value given by TriggerStep.

8 = SingleTrigger: A trigger pulse is written when the axis has reached the trigger position given by TriggerPosition.

for TriggerPosition: position where a trigger pulse is to be output, in physical units

Response: none

Troubleshooting: Invalid trigger line identifier, invalid parameter ID

Example: The trigger output line of the controller with device ID 1 is to be configured so that multiple trigger pulses are output using the "Position+Offset" trigger mode. The distance (increment value) for the individual trigger pulses is to be 2, the position where the first trigger pulse is to be output is 0.

Send: CTO 1 1 2 1 3 7 1 10 0

The trigger settings are to be activated so that the trigger pulses will be output at the appropriate positions when motion of axis A ist started;

Send: TRO 1 1

CTO? (Get Trigger Parameter)

Description: Replies with the value set for specified trigger output lines and parameters

Format: CTO? [{<TriggerOut> <uint>}]

Arguments:	<p><TriggerOut> is one trigger output line of a Mercury™ network. Note that the ID of a trigger output line corresponds with the device number of the controller to which it belongs (the controller device number is set with DIP switches 1 to 4 on the controller front panel)</p> <p><uint> is the CTO parameter ID in decimal format, see CTO</p> <p>If all arguments are omitted, all values for all output lines are given.</p>
Response:	<p>{<TriggerOut> <uint>="<ParamValue> LF}</p> <p>where</p> <p><ParamValue> is the value to which the CTO parameter is set, see CTO.</p>
Troubleshooting:	Invalid trigger line identifier, invalid parameter ID

DEL (DElay)

Description:	<p>DELays <uint> milliseconds.</p> <p>Delays execution of macro.</p> <p>DEL is used only within macros. Commands sent to the controller network during the delay will be queued.</p> <p>Do not mistake MAC DEL which deletes macros for DEL which delays.</p> <p>This command can be interrupted with #24.</p>
Format:	DEL <uint>
Arguments:	<uint> is the delay value in milliseconds.
Response:	none

DFF (DeFine Factor)

Description:	<p>Scale factor for physical units, e.g. a factor of 25.4 sets the physical units to inches. Changing the scale factor will change the numerical results of other commands, but not the underlying physical magnitudes.</p> <p>See Section 9.4 on p. 76 and Section 1.3.1 on p. 7 for more information.</p>
--------------	---

Format: DFF <AxisID> <float>[{ <AxisID> <float>}]
 Arguments: <AxisID>: is a valid axis identifier
 <float>: is the value to set, can be in the form of v.vv
 Response: none
 Troubleshooting: Illegal axis identifier

DFF? (get factor)

Description: Gets the scale factors set by the DFF command for the queried axes
 Format: DFF? [{<AxisID>}]
 Arguments: <AxisID>: is a valid axis identifier, if omitted answers for all axes
 Response: {<AxisID>="<float> LF}
 where
 <float> is the scale factor of <AxisID>
 Troubleshooting: Illegal axis identifier

DFH (DeFine Home

Description: Defines the current position of given axes as the axis home position (by setting the position value to 0.00). If <AxisID>= all axes are affected.
 Due to the redefinition of the home (zero) position the numeric limits of the travel range are changed.
 The home position is reset to its default location by REF (p. 55), MNL (p. 51) and MPL (p. 52).
 Format: DFH [{<AxisID>}]
 Arguments: <AxisID>: is a valid axis identifier
 Response: none
 Troubleshooting: Illegal axis identifier

DFH? (get home positions)

Description:	Gets home position (offset)
Format:	DFH? [{<AxisID>}]
Arguments:	<AxisID>: is a valid axis identifier
Response:	{<AxisID>="<float> LF} where <float> is the distance from the default home position to the current home position
Troubleshooting:	Illegal axis identifier

DIO (set Digital I/O)

Description:	Switches the specified digital output line(s) to specified state(s). Use TIO? (p. 62) to get the number of installed digital I/O lines (four for each controller). The output line designators depend on the device number of the controller to which the lines belong. For the controller with device number 1, they will be A, B, C, D (see "I/O Line Designators" on p. 6 for more details). Do not use DIO on any physical trigger line for which trigger output is enabled with TRO (p.).
Format:	DIO <OutLineID> <uint>[{ <OutLineID> <uint>}]
Arguments:	<OutLineID> is one digital output line designator If <uint>=1 the line is set to HIGH/ON, if <uint>=0 it is set to LOW/OFF.
Response:	none
Troubleshooting:	

DIO?

Description:	Lists the states of the specified digital input lines. Can be used to query externally generated signals.
Format:	DIO? [{ <InLineID>}]

Arguments: <InLineID> is the identifier of the input line to use with DIO?. Use TIO? (p. 62) to get the number of installed digital I/O lines (four for each controller). The input line designators depend on the device number of the controller to which the lines belong. For the controller with device number 1, they will be A, B, C, D (see “I/O Line Designators” on p. 6 for more details).

Response: {<InLineID>="<uint> LF}
 when
 <uint>=0 digital input is LOW/OFF
 <uint>=1 digital input is HIGH/ON

Troubleshooting:

ERR? (get ERRor)

Description: Get **ERR**or code <int> of the last error and reset the error to 0.
 Only the last error is buffered. Therefore you might wish to call ERR? after each command.
 Negative error codes (< 0) are DLL-related, positive (> 0) command- or controller-related. The error codes and their description are fully listed in Section 10 on p. 82.

Format: ERR?

Arguments: none

Response: The error code of the last occurred error (int).

Troubleshooting: Communication breakdown

The following table shows a selection of possible controller errors:

0	No error
1	Parameter syntax error
2	Unknown command
5	Unallowable move attempted on unreferenced axis, or move attempted with servo off
7	Position out of limits
8	Velocity out of limits
10	Controller was stopped by command
15	Invalid axis identifier
16	Unknown stage name
17	Parameter out of range
18	Invalid macro name
19	Error while recording macro
20	Macro not found
22	Axis identifier specified more than once

23	Illegal axis
24	Incorrect number of parameters
25	Invalid floating point number
26	Parameter missing
34	Command not allowed for selected stage(s)
50	Attempt to reference axis with referencing disabled
54	Unknown parameter
1000	Too many nested macros
-1001	Unknown axis identifier

GOH (GO Home)

Description:	Move given axes to home position. GOH [{<AxisID>}] is the same as MOV {<AxisID> 0} This command can be interrupted by #24 and STP.
Format:	GOH [{<AxisID>}]
Arguments:	<AxisID>: is a valid axis identifier, if omitted, both axes are affected
Response:	none
Troubleshooting:	Illegal axis identifier

HLP? (HeLP)

Description:	List a HeLP string which contains all available commands.
Format:	HLP?
Arguments:	none
Response:	List of commands available
Troubleshooting:	Communication breakdown

HLT (HaLT)

Description:	<p>HaLT the motion of given axes smoothly. Only commands causing non-complex motion (e.g. MOV, GOH) can be interrupted with HLT.</p> <p>Error code 10 is set.</p> <p>Use #24 instead to stop complex motions like referencing, etc.</p> <p>HLT stops motion with given system deceleration with regard to system inertia.</p> <p>STP (p. 60) in contrast aborts current motion as fast as possible for the controller without taking care of systems inertia or oscillations.</p> <p>After the axis was stopped, the target position is set to the current position.</p>
Format:	HLT [{ <AxisID>}]
Arguments:	<AxisID>: is a valid axis identifier, if omitted all axes are halted
Response:	none
Troubleshooting:	Illegal axis identifier

HPA? (Display Parameter Help Message)

Description:	<p>Responds with a help string which contains all available parameters with short descriptions. See "Motion Parameter Overview" on p. 69 for further details.</p> <p>The listed parameters can be changed and/or saved using the SPA command (p. 58). SPA affects the parameter settings in volatile memory (RAM).</p>
Format:	HPA?
Arguments:	none
Response:	<p>{<PamID>="<string> LF}</p> <p>where</p> <p><PamID> is the ID of one parameter, hexadecimal format</p> <p><string> is a string which describes the corresponding parameter. The string has following format:</p>

<CmdLevel>TAB<MaxItem>TAB<DataType>TAB<ParameterDescription>

<CmdLevel> and <MaxItem> can be ignored,
<ParameterDescription> gives the parameter name.

Troubleshooting:

INI (INItialization)

Description:	Initializes the axis: sets reference state to "not referenced," switches servo on, sets the brake control line in the "brake off" state and if axis was under joystick-control, disables the joystick.
Format:	INI [{ <AxisID>}]
Arguments:	<AxisID>: is a valid axis identifier
Response:	none
Troubleshooting:	Illegal axis identifier

JAS? (Get joystick value)

Description:	Get the current status of the given axis of the given joystick device which is directly connected to the controller. The reported factor is applied to the velocity set with VEL, the range is -1.0 to 1.0.
Format:	JAS? [{<JoystickID> <JoystickAxis>}]
Arguments:	<p><JoystickID> is the device number of the Mercury™ controller to which the joystick is connected; set with the DIP switches 1 to 4 on the controller front panel, can be 1 to 16</p> <p><JoystickAxis> is one axis of the joystick; must be 1 for Mercury™ controllers, which only support one joystick axis per device</p>
Response:	<p>{<JoystickID> <JoystickAxis>}"="<Amplitude>}</p> <p>where</p> <p><Amplitude> is the factor which is currently applied to the current valid velocity setting of the controlled motion axis, corresponds to the current displacement of the joystick axis.</p>

JAX? (Get joystick-to-axis assignments)

Description:	Reports correspondence between joystick port numbers (device numbers) and axis identifiers for axes with joystick ports.
Format:	JAX?
Arguments:	none
Response:	{<JoystickID> <JoystickAxis>="{"<AxisID> }LF}
	where
	<JoystickID> is the device number of the Mercury™ controller to which the joystick is connected; set with the DIP switches 1 to 4 on the controller front panel, can be 1 to 16
	<JoystickAxis> is one axis of the joystick; is always 1 for Mercury™ controllers which only support one joystick axis per device
	<AxisID> is the axis of the Mercury™ controller to which the joystick is connected, can be A to P

Troubleshooting:

JBS? (Query If Joystick Button Is Pressed)

Description:	Get the current status of the given button of the given joystick device which is directly connected to the controller.
Format:	JBS? [{<JoystickID> <JoystickButton>}]
Arguments:	<JoystickID> is the device number of the Mercury™ controller to which the joystick is connected; set with the DIP switches 1 to 4 on the controller front panel, can be 1 to 16
	<JoystickButton> one button of the joystick; must be 1 for Mercury™ controllers, which only support one joystick button per device

Response: {<JoystickID> <JoystickButton>="<State>}

where

<State> indicates if the joystick button is pressed; 0 = not pressed, 1 = pressed

JDT (load Joystick response Table)

Description: Load pre-defined joystick response table. The table type can be either 1 for linear or 3 for cubic response curve. The cubic curve offers more sensitive control around the middle position and less sensitivity close to the maximum velocity.
To ensure that the loaded response table is used, call JDT before you enable joystick operation with JON.

Format: JDT {<JoystickID> <JoystickAxis> <TableType>}

Arguments: <JoystickID> is the device number of the Mercury™ controller to which the joystick is connected; set with the DIP switches 1 to 4 on the controller front panel, can be 1 to 16

<JoystickAxis> is one axis of the joystick; is always 1 for Mercury™ controllers which only support one joystick axis per device

<TableType>: 1 for linear, 3 for cubic

Response: none

Troubleshooting: Illegal joystick axis number, unsupported table type

JLT (Fill Joystick Response Table)

Description:	<p>Fill the response table for the given axis of the given joystick device which is directly connected to the controller.</p> <p>The amplitudes of the joystick axes (i.e. their displacements) are mapped to the current valid velocity settings of the controller axes. For each joystick axis there is a response table that defines this mapping. With JLT this table can be written. (A default table provided by the controller can be loaded with the JDT command).</p> <p>To ensure that the loaded response table is used, call JLT before you enable joystick operation with JON.</p>
Format:	JLT <JoystickID> <JoystickAxis> <StartPoint> {<floatn>}
Arguments:	<p><JoystickID> is the device number of the Mercury™ controller to which the joystick is connected; set with the DIP switches 1 to 4 on the controller front panel, can be 1 to 16</p> <p><JoystickAxis> is one axis of the joystick; is always 1 for Mercury™ controllers which only support one joystick axis per device</p> <p><StartPoint> is the start point in the response table, starts with 1</p> <p><floatn> is the value of point n; 256 values must be written. The first point corresponds to the maximum joystick axis displacement in negative direction, the 256th point to the maximum displacement in positive direction. The <floatn> values are factors which will during joystick control be applied to the velocity set with VEL, the range is -1.0000 to 1.0000.</p>
Response:	none

Example: In the current lookup table, point 1 has the value -1, hence the controlled axis will move with full velocity in negative direction at the maximum negative displacement of the joystick. Points 124 to 133 have the value 0, i.e. at the center position of the joystick and in a small area around the center, the velocity is 0 and the controlled axis will not move. Point 236 has the value 0.8369, i.e. when the displacement of the joystick axis is about 2/3 in positive direction, the controlled axis will move in positive direction with about 4/5 of the full velocity. Point 256 has the value 1, i.e. the controlled axis will move with full velocity in positive direction at the maximum positive displacement of the joystick.

JLT? (Get Joystick Response Table Values)

Description:	Reading of the current valid response table values.
Format:	JLT? [<StartPoint> [<NumberOfPoints> [{<JoystickID> <JoystickAxis>}]]]
Arguments:	<p><StartPoint>: is the start point in the response table, starts with 1</p> <p><NumberOfPoints>: is the number of points to be read per joystick axis; maximum number is 256</p> <p><JoystickID> is the device number of the Mercury™ controller to which the joystick is connected; set with the DIP switches 1 to 4 on the controller front panel, can be 1 to 16</p> <p><JoystickAxis> is one axis of the joystick; is always 1 for Mercury™ controllers which only support one joystick axis per device</p>
Response:	The response table content in GCS array format, see the separate manual for GCS array, SM 146E, and the example below. The reported values are factors which will during joystick control be applied to the velocity set with VEL, the range is -1.0000 to 1.0000.

Example:

```

jlt? 122 20
# VERSION = 1
# TYPE = 1
# SEPARATOR = 32
# DIM = 2
# NDATA = 20
#
# NAME0 = Joystick 1 Axis 1
# NAME1 = Joystick 3 Axis 1
#
# END_HEADER
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000000 0.000000
0.000977 0.000977
0.000977 0.000977
0.001953 0.001953
0.001953 0.001953
0.002930 0.002930
0.003906 0.003906

```

JON (Activate/deactivate joystick control)

Description: Enable/disable direct joystick control for given joystick device (i.e. for the Mercury™ controller to which the joystick is connected). Do not enable controllers with no physical joystick connected, as uncontrolled motion could occur.

When an axis is controlled by a directly connected joystick, it can no longer be moved by motion commands or by trackball.

Format: JON [{ <JoystickID> <State>}]

Arguments: <JoystickID> is the device number of the Mercury™ controller to which the joystick is connected; set with the DIP switches 1 to 4 on the controller front panel, can be 1 to 16

<State>: 0 for disable, 1 for enable

Response: none

Troubleshooting: Illegal joystick number, unsupported state

JON? (Get Joystick Activation Status)

Description: Get activation state of the given joystick which is directly connected to the controller.

Format: JON? [{<JoystickID>}]

Arguments: <JoystickID> is the device number of the Mercury™ controller to which the joystick is connected; set with the DIP switches 1 to 4 on the controller front panel, can be 1 to 16

Response: {<JoystickID>="<uint>}

where

<uint> is the joystick activation state: 1 = joystick enabled, 0 = joystick disabled

LIM? (indicate LIMit switches)

Description: Indicates whether axes have limit switches.

Format: LIM? [{<AxisID>}]

Arguments: <AxisID>: is a valid axis identifier

Response: {<axis>="<uint> LF}
where

<uint> indicates whether the axis has limit switches (=1) or not (=0).

Troubleshooting: Illegal axis identifier

MAC (macro)

Description: Call a **MAC**ro function. Permits recording, deleting and running macros on the controller (see Macro Storage on Controller, p. 16 for details).

Format:	<p>MAC <keyword> {<parameter>}</p> <p>in particular:</p> <p>MAC BEG <macroname></p> <p>MAC DEL <macroname></p> <p>MAC END</p> <p>MAC NSTART <macroname> <uint></p> <p>MAC START <macroname></p>
Arguments:	<p><keyword> determines which macro function is called. The following keywords and parameters are used:</p> <p>MAC BEG <macroname> Start recording a macro on the controller to be named <i>macroname</i>, which must be of the form <i>aMC0nn</i> where <i>a</i> is the axis designation of the axis controlled by the controller on which the macro is to be stored and <i>nn</i> is the ID number for the macro, 0 to 31 (0 is used for the startup macro instead of “STARTMAC”, the designation understood by other GCS controllers). This command may not be used in a macro; the commands that follow become the new macro, so if successful, the error code cannot be queried. End the recording with MAC END. A macro cannot be overwritten, only deleted.</p> <p>MAC END Stop macro recording (cannot become part of a macro); any error during macro recording can be seen with ERR? after MAC END</p> <p>MAC DEL <macroname> Deletes specified macro</p> <p>MAC NSTART <macroname> <uint> Repeat the specified macro <uint> times. Each execution is started when the previous one has finished. See also MAC START for further details.</p> <p>MAC START <macroname> Starts execution of specified macro. A running macro sends no responses to any interface, and will continue even if the controller is deselected. This means query commands, if present in a macro, are useless. The only communication possible with a controller running a macro is with single-character commands.</p>
Response:	none

- Troubleshooting: Macro recording is active (keywords BEG, DEL) or inactive (END)
Macro contained a disallowed MAC command
- Examples: MAC BEG AMC000
Start recording a macro named AMC000. Macros with the number “000” are special in that they will be run by the controller on which they are stored upon power-up or reset, even without a host PC connected

MAC? (list macro)

- Description: List **MAC**ros or contents of a given macro.
- Format: MAC? [<macroname>]
- Arguments: <macroname>: name of the macro whose contents shall be listed; if omitted, the names of all stored macros are listed
- Response: <string>
if <macroname> is given, <string> is the contents of this macro as GCS commands, one per line;
if <macroname> is omitted, <string> is a list with the names of all macros stored on all controllers in the controller network, one per line
- Troubleshooting: Macro <macroname> not found

MEX (Stop macro execution if condition true)

- Description: Stop macro execution due to a given condition of the following type: a specified value is compared with a queried value according to a specified rule.

This command can only be used in macros.

When the macro interpreter accesses this command the condition is checked. If it is true the current macro is stopped, otherwise macro is execution is continued with the next line. Should the condition be fulfilled later, the interpreter will ignore it.

See also WAC, p. 66.
- Format: MEX <CMD?> <OP> <value>

Arguments: <CMD?> is a query command in its usual syntax. The answer must consist of a single value.
Supported is DIO?

 <OP> is the operator to be used. The following are allowable (controller firmware specific)
 "=" "<=" "<" ">" ">=" "!="

 <value> is the value to be compared with the response to <CMD?>

Response: none

Example: Send: MAC START AMC001

 Note: Macro "AMC001" has the following contents:

 MAC START AMC002

 MAC START AMC003

 MEX DIO? D = 1

 MAC START MAC1

 Macro "AMC002" has the following contents:

 MEX DIO? D = 1

 MEX DIO? A = 0

 MVR A 1.0

 DEL 100

 Macro AMC003" has the following content:

 MEX DIO? D = 1

 MEX DIO? B = 0

 MVR A -1.0

 DEL 100

Macro AMC001 forms an infinite loop by permanently calling AMC002, AMC003 and itself.

AMC002 checks the state of the digital input channel A. If it is not set (0), the macro is aborted, otherwise the macro will move axis A by 1.0 in positive direction (relative move).

AMC003 checks the state of the digital input channel B and moves axis A in negative direction accordingly.

Connecting the digital input channels A, B and D with pushbuttons, it is possible to implement interactive control of an axis without any software assistance. The delay (DEL 100) is required to avoid generation of multiple MVR commands while pressing the push-button for a short time.

Channel D is used as a global exit. Since MEX stops execution of the current macro only, it must also be included in the calling macro, which would otherwise continue.

MNL (Move to Negative Limit)

Description:	<p>Moves the given axis to its negative limit switch, sets the position counter to 0, and sets the reference state to "reference OK" (see Section 3 on p. 11 for more information regarding referencing).</p> <p>If <AxisID> is omitted, moves all axes. If multiple axes are affected by MNL, one axis after another is moved to its limit switch.</p> <p>This command can be interrupted by #24.</p> <p>Note that MNL resets the home position set with DFH, p. 36.</p>
Format:	MNL [{ <AxisID>}]
Arguments:	<AxisID>: is a valid axis identifier
Response:	<p>{<uint> LF}</p> <p><uint> indicates success of the referencing procedure: 0 = not successful 1 = successful</p>
Troubleshooting:	Illegal axis identifier

MOV (MOVE absolute)

Description:	<p>Set new absolute target position for given axis. Axes will start MOVing to the new positions only if ALL given targets are within the allowed ranges and ALL given axes can move.</p> <p>All given axes start moving simultaneously.</p> <p>This command can be interrupted by #24, STP and HLT.</p> <p>Servo-control must be enabled for all commanded axes prior to using this command, and the axes must be referenced.</p>
Format:	MOV <AxisID> <float>[{ <AxisID> <float>}]
Arguments	<p><AxisID> is a valid axis identifier</p> <p><float> is the target position in physical units.</p>
Response:	none
Troubleshooting:	Parameter out of limits, Illegal axis identifier, joystick or trackball enabled for axis

Example 1:	Send:	MOV A 10 B 2
	Note:	Axis A moves to 10, axis B moves to 2 (all target positions in the physical unit valid for the appropriate axis)
Example 2:	Send:	MOV A 243
	Send:	ERR?
	Receive:	7
	Note:	The axis does not move. The error code "7" replied by the ERR? command indicates that the target position given by the move command is out of limits.

MOV? (read target position)

Description:	Returns last valid commanded target position. The target position is changed by all commands that cause motion (e.g. MOV, MVR, REF, MNL, MPL, GOH). Note that MOV? gets the commanded positions. Use POS? (p. 55) to get the current positions.
Format:	MOV? [{ <AxisID>}]
Arguments:	<AxisID> is a valid axis identifier
Response:	{<AxisID>="<float> LF} where <float> is the last commanded target position in physical units
Troubleshooting:	Illegal axis identifier

MPL (Move to Positive Limit)

Description:	Moves the given axis to its positive limit switch, sets the position counter to the maximum position value, and sets the reference state to "reference OK" (see Section 3 on p. 11 for more information regarding referencing). If <AxisID> is omitted, moves all axes. If multiple axes are affected by MPL, one axis after another is moved to its limit switch. This command can be interrupted by #24. Note that MPL resets the home position set with DFH, p. 36.
Format:	MPL [{ <AxisID>}]

Arguments:	<AxisID>: is a valid axis identifier
Response:	{<uint> LF} <uint> indicates success of the referencing procedure: 0 = not successful 1 = successful
Troubleshooting:	Illegal axis identifier; reference mode must be "1" (see RON, p. 56)

MVR (MoVe relative)

Description:	<p>MoVe given axes Relative to their current position.</p> <p>The new target position is calculated by adding the given value <float> to the last -commanded target value. Axes will start moving to the new position if ALL given targets are within the allowed range and ALL given axes can move.</p> <p>This command can be interrupted by #24, HLT and STP. Servo must be enabled prior to using this command.</p>	
Format:	MVR <AxisID> < float >[{ <AxisID> <float>}]	
Arguments:	<p><AxisID> is a valid axis identifier.</p> <p><float> added to the last commanded target position is set as new target position in physical units.</p>	
Response:	none	
Example:	<p>Send: MOV A -0.5 B 12.3 Note: This is an absolute move Send: POS? A B Receive: A=-0.500000 B=12.300000</p> <p>Send: MVR A 1 B 2 Note: This is a relative motion. Send: POS? A B Receive: A=0.500000 B=14.300000</p> <p>Send: MVR A 1 B 2000 Note: Target position of axis B exceeds travel range. Command is not executed</p> <p>Send: POS? A B Receive: A=0.500000 B=14.300000</p>	

ONT? (axis ON Target)

Description:	Get ON-Target status of given axis. When <AxisID> is omitted, get all axes.
Format:	ONT? [{ <AxisID>}]
Arguments:	<AxisID> is a valid axis identifier.
Response:	{<AxisID>="<uint> LF} where <uint> = "1" when the specified axis is on-target, "0" otherwise.
Troubleshooting:	Illegal axis identifier

POS (set real POSition)

Description:	Sets the current POS ition (does not cause motion). Allowed only when the reference mode is set to "0", see RON (p. 56). An axis is considered as "referenced" when the position has been set with POS (see Section 3 on p. 11 for more information). The minimum and maximum commandable positions (TMN?, p. 62, TMX?, p. 63) are not adjusted when a position is set with POS. If the value set with POS is not correct, there will be target positions which are allowed by the software but cannot be reached by the hardware and others which could be reached by the hardware but are disallowed by the software. This command can change the physical location of the home position (zero point), perhaps putting it outside of the travel range.
Format:	POS [{ <AxisID> <float>}]
Arguments:	<AxisID> is a valid axis identifier. <float> is the new numeric value for the current position in physical units.
Response:	none
Troubleshooting:	Reference mode is "1", Illegal axis identifier

POS? (read real POSition)

Description:	<p>Returns the current POSition.</p> <p>If <AxisID> is omitted, all axes are queried.</p> <p>Response depends on the factor set by DFF (p. 35).</p>
Format:	POS? [{ <AxisID>}]
Arguments:	<AxisID> is a valid axis identifier.
Response:	<p>{<axis>="<float> LF}</p> <p>where</p> <p><float> is the current position in physical units</p>
Troubleshooting:	Illegal axis identifier

REF (move to REFerence position)

Description:	<p>Moves the given axis to its reference switch, sets the position counter to the reference position value (stage-type specific value stored on the controller), and sets the reference state to "reference OK" (see Section 3 on p. 11 for more information regarding referencing). If the move begins on the positive side of the reference switch, the switch will be passed and re-approached from the negative side.</p> <p>If <AxisID> is omitted, moves all axes. If multiple axes are affected by REF, one axis after another is moved to its switch.</p> <p>The REF command always approaches the reference switch from the same side, no matter where the axis is when it is issued.</p> <p>This command can be interrupted by #24.</p>
Format:	REF [{ <AxisID>}]
Arguments:	<AxisID> is a valid axis identifier.
Response:	<p>{<uint> LF}</p> <p><uint> indicates success of the referencing procedure:</p> <p>0 = not successful</p> <p>1 = successful</p>
Troubleshooting:	Illegal axis identifier; reference mode must be "1" (see RON, p. 56)

REF? (list axes with REference sensor)

Description:	Indicate whether specified axes have reference sensors.
Format:	REF? [{<AxisID>}]
Arguments:	<AxisID> is a valid axis identifier.
Response:	{<axis>="<uint> LF} where <uint> indicates whether the axis has a reference switch (=1) or not (=0).
Troubleshooting:	Illegal axis identifier

RON (set reference mode)

Description:	Set reference mode of given axes. Mode = 0: referencing moves with REF (p. 55), MNL (p. 51) and MPL (p. 52) are not possible, absolute position must be set with POS (p. 54) to reference the axis. Mode = 1: REF or MNL or MPL is required to reference the axis, usage of POS is not allowed. See Section 3 on p. 11 for more information.
Format:	RON { <AxisID> <mode>}
Arguments:	<AxisID> is a valid axis identifier. <mode> can be 0 or 1 (see description above for the meaning). Default is taken from stage database, usually 1.
Response:	none
Troubleshooting:	Illegal axis identifier

RON? (get reference mode)

Description:	Get reference mode of given axes.
Format:	RON? [{ <AxisID>}]
Arguments:	<AxisID> is a valid axis identifier.

Response: {<AxisID>="<mode> LF}
 where
 <mode> is the current reference mode of the axis, see
 RON

Troubleshooting: Illegal axis identifier

SAI (Set Axis Identifier)

Description: **Set old Axis Identifier to new identifier.**
 TVI? (p. 64) provides a list of valid axis identifiers.

Format: SAI <AxisID> <newaxis>[{ <AxisID> <newaxis>}]

Arguments: <AxisID> is a valid axis identifier.
 <newaxis> is the new identifier for <AxisID>

Response: none

Troubleshooting: Illegal axis identifier or duplicate axis identifier

SAI? (get axis identifier)

Description: Gets the axis identifiers.
 Note that SAI? without an argument will only return the axes which are assigned to stages (see CST, p. 32). In contrast to this, SAI? ALL and CST? (p. 32) will always return all axes, i.e. the answer also includes the axes set to NOSTAGE.

Format: SAI? [ALL]

Arguments: The parameter ALL is optional. If used, the answer includes the axes which are not connected to stages (stage name is NOSTAGE).

Response: {<AxisID> LF}
 <AxisID> is a valid axis identifier.

SPA (Set Parameter)

Description:	<p>Set a given PArameter of the given axis to given value in volatile memory.</p> <p>Parameter changes will be lost when the controller is powered off or rebooted. See the <i>PI Stage Editor</i> or <i>PIMikroMove™</i> manuals for ways to save parameter sets as user stages.</p> <p>CAUTION! The SPA command is for setting hardware-specific parameters. Incorrect values may lead to improper operation or damage of your hardware!</p>
Format:	SPA <AxisID> <ParamNumber> <ParamValue>[{<AxisID> <ParamNumber> <ParamValue>}]
Arguments	<p><AxisID>: is a valid axis identifier</p> <p><ParamNumber> is the parameter ID. Valid parameter IDs are listed on p. 71. Parameter IDs can be given in hexadecimal or decimal format.</p> <p><ParamValue> is the value to which the parameter <ParamNumber> of <AxisID> is to be set.</p>
Response:	none
Troubleshooting:	Illegal axis identifier, incorrect parameter ID
Example:	<p>Send: SPA A 0xA 0.05 B 0xA 0.08</p> <p>Note: Set the maximum velocity for axis A to 0.05 mm/s and for axis B to 0.08 mm/s</p>

SPA? (Get Parameter)

Description:	Get the value of specified parameters of specified axes
Format:	SPA? [{ <AxisID> <ParamNumber>}]
Arguments:	<p><AxisID>: is a valid axis identifier</p> <p><ParamNumber> is the parameter ID. Valid parameter IDs are listed on p. 71. Parameter IDs can be given in hexadecimal or decimal format.</p>
Response:	<p>{<AxisID> <ParamNumber>="<ParamValue> LF}</p> <p>where</p> <p><ParamValue> is the value to which the parameter <ParamNumber> of <AxisID> is set.</p>

Troubleshooting: Illegal axis identifier, incorrect parameter ID

Example: Send: SPA? A 0xA A 0xB A 0x12
 Receive: A10=0.1
 A11=10
 A18=1
 Note: Parameter IDs can be given in hexadecimal or decimal format in the query command. Irrespective of that, in the response they are always displayed in decimal format.

SRG? (Read register)

Description: Read the values of the specified registers.

Format: SRG? [{ <AxisID> <RegisterID>}]

Arguments: <AxisID> is a valid axis identifier
 <RegisterID> is the ID of the specified register (must be 3)

Response: {<AxisID> <RegisterID> = <Value>}
 where
 <Value> is the signal input lines register (byte 2 of the C-663 and byte 5 of the C-863):

C-863 DC-motor versions:

Bit 0: not used
 Bit 1: Reference signal (input)
 Bit 2: Positive limit signal (input)
 Bit 3: Negative limit signal (input)
 Bit 4: DIO 1
 Bit 5: DIO 2
 Bit 6: DIO 3
 Bit 7: DIO 4

C-663 stepper motor versions:

Bit 0: Limit negative
 Bit 1: Reference signal
 Bit 2: Limit positive
 Bit 3: no function
 Bit 4: Digital input 1
 Bit 5: Digital input 2
 Bit 6: Digital input 3
 Bit 7: Digital input 4

STP (Stop Motion)

Description:	SToPs the motion of all axes immediately. Error code 10 is set. After the axes were stopped, the target position is set to the current position. STP always does a hard stop. Use HLT (p. 40) instead to stop individual axes smoothly.
Format:	STP
Arguments:	none
Troubleshooting:	Communication breakdown

SVO (set SerVO on or off)

Description:	<p>Sets SerVO-control mode on or off for given axes. When servo-control is off for an axis:</p> <ul style="list-style-type: none"> • All positioning commands (e.g. MOV, MVR) are ignored. • With C-863 DC motor controllers, servo-loop and motor are deactivated. The current reference state is kept—encoder signals are still read so that the current position is always known. An axis can move in the usual way as soon as servo-control is switched on again. • With C-663 stepper motor controllers, the motor is deactivated (no servo-loop implemented). The current reference state is reset to "not referenced" because the C-663 can no longer know the current position when the motor is deactivated. This is why an axis must always be referenced to allow for positioning commands after servo-control was switched on again. <p>When servo is switched off while stage is moving, the stage stops.</p>
Format:	SVO <AxisID> <status>[{ <AxisID> <status>}]
Arguments:	<p><AxisID>: is a valid axis identifier</p> <p><status>= 0 set servo off, 1 set servo on</p>
Response:	none
Troubleshooting:	Illegal axis identifier

SVO? (get servo status)

Description: Gets **SerVO**-control mode for given axes.
Get all axes when <AxisID>=""

Format: SVO? [{ <AxisID>}]

Arguments: <AxisID>: is a valid axis identifier

Response: {<AxisID>=""<status> LF}
where
<status>= 0 servo is off, 1 servo is on

Troubleshooting: Illegal axis identifier

TAC? (Tell Analog Channels)

Description: Get the number of installed analog lines. With Mercury™ controllers, the response contains only the analog channels on the I/O connector.

Format: TAC?

Parameter: <none>

Response: <uint>
<uint> is the number of analog input lines

Troubleshooting:

TAV? (Get Analog Input)

Description: Get voltage at analog input.

Format: TAV? [{ <InputID>}]

Parameter: <InputID> ID to use to read corresponding input line in analog mode: A1 to A64 (see "I/O Line Designators" on p. 6 for details)

Response: {<InputID>=""<float> LF}
where:
<float> is the current voltage at the input channel.

TIO? (Tell Digital I/O Lines)

Description:	Tell number of installed digital I/O lines
Format:	TIO?
Arguments:	none
Response:	I=<uint1> O=<uint2> where <uint1> is the number of digital input lines. <uint2> is the number of digital output lines.

TMN? (Tell Minimum Travel Value)

Description:	<p>Get the minimum commandable position in physical units</p> <p>The minimum commandable position is defined by the MAX_TRAVEL_RANGE_NEG parameter, ID 0x30.</p> <p>When the physical unit of an axis is scaled with DFF (p. 35), or the zero-point shifted with DFH, the minimum commandable position is automatically adjusted to the appropriate new value.</p>
Format:	TMN? [{ <AxisID>}]
Arguments:	<AxisID>: is a valid axis identifier
Response	{<AxisID>="<float> LF} where <float> is the minimum commandable position in physical units

TMX? (Tell Maximum Travel Value)

Description:	<p>Get the maximum commandable position in physical units.</p> <p>The maximum commandable position is defined by the MAX_TRAVEL_RANGE_POS parameter, ID 0x15.</p> <p>When the physical unit of an axis is scaled with DFF (p. 35), or the zero-point shifted with DFH, the maximum commandable position is automatically adjusted to the appropriate new value.</p>
Format:	TMX? [{ <AxisID>}]
Arguments:	<AxisID>: is a valid axis identifier
Response	<p>{<AxisID>="<float> LF}</p> <p>where</p> <p><float> is the maximum commandable position in physical units</p>

TNJ? (Tell number of joysticks)

Description:	<p>Get the number of joysticks. Note: the software can not determine if a joystick is actually connected to a Mercury™ controller. This is the maximum possible number of joystick axes that can be connected to the network.</p>
Format:	TNJ?
Arguments:	none
Response	<uint> is the number of joystick axes that can be connected.

TRO (Enable/disable Trigger Output)

Description:	<p>Enables or disables the trigger output conditions which were set with CTO (p. 33) for the given trigger output line. On Mercury™ controllers, the digital output line 4 can be configured for trigger output (pin 8 of the "I/O" socket, see User manual for pinout).</p> <p>Do not use DIO (p. 37) on output lines for which the trigger output is activated with TRO.</p>
--------------	--

Format:	TRO {<TriggerOut> <Mode>}
Arguments:	<p><TriggerOut> is one trigger output line of a Mercury™ network. Note that the ID of a trigger output line corresponds with the device number of the controller to which it belongs (the controller device number is set with DIP switches 1 to 4 on the controller front panel)</p> <p><Mode> = 1 enables trigger output mode, 0 disables trigger output mode</p>
Response	none

TRO? (Get trigger output enable status)

Description:	Gets trigger output-mode enable-status for given trigger output line (the trigger output configuration is made with CTO).
Format:	TRO? [{<TriggerOut>}]
Arguments:	<p><TriggerOut> is one trigger output line of a Mercury™ network. Note that the ID of a trigger output line corresponds with the device number of the controller to which it belongs (the controller device number is set with DIP switches 1 to 4 on the controller front panel)</p>
Response	<p>{<TriggerOut>="<Mode>}</p> <p>where</p> <p><Mode> = 1 if trigger output mode enabled, 0 if trigger output mode disabled</p>

TVI? (Tell Valid axis Identifiers)

Description:	<p>Tell Valid set of characters which can be used as axis Identifiers.</p> <p>Note: Use SAI (p. 57) to set axis identifiers and SAI? ALL (p. 57) to get the axis identifiers which are currently used.</p>
--------------	---

Format: TVI?

Arguments: none

Response: <string> is a list of characters

Troubleshooting:

VEL (Set Velocity)

Description: Set **VE**LOCITY to use for moves of specified axes.

Notes:

The maximum velocity of an axis is given by SPA parameter 0xA (see SPA, p. 58, and parameter list on p. 71). VEL does not change this maximum but sets only the currently applied velocity (parameter 0x49).

When the physical unit of an axis is scaled with DFF (p. 35), the velocity is automatically adjusted to the appropriate new value.

VEL can be changed while the axis is moving.

Format: VEL <AxisID> <float>[{ <AxisID> <float>}]

Arguments: <AxisID>: is a valid axis identifier
<float> is the velocity value in physical units, it must be positive or zero.

Response: none

Troubleshooting: Illegal axis identifiers, given velocity exceeds the maximum velocity value (SPA param. 10), axis is under joystick control

VEL? (Get Velocity)

Description: Get current velocity settings of given axes.

Format: VEL? [{ <AxisID>}]

Arguments: <AxisID>: is a valid axis identifier, if omitted, all axes are queried

Response: {<axis>="<float> LF}
where:
<float> is the current velocity setting in physical units / s.

VER? (Get Version)

Description:	Returns the VER sion of the firmware and of the PI Mercury GCS DLL.
Format:	VER?
Arguments:	none
Response:	<string> is the version information of the firmware, e.g. PI_Mercury_GCS_DLL: 1.9.2.4 A:(c)2009 Physik Instrumente(PI) Karlsruhe, C-863, Ver. 2.30, 2009-21-07 C:(c)2007 Physik Instrumente(PI) Karlsruhe, C-663, Ver. 1.20, 2009-06-26

VST? (Get available Stages)

Description:	List the names of all stages which can be connected to the controller (with CST, p. 32).
Format:	VST?
Arguments:	none
Response:	{<string> LF} where <string> is a stage name.

WAC (Wait for Condition)

Description:	Wait until a given condition of the following type occurs: a specified value is compared with a queried value according a specified rule. Can only be used in macros. See also MEX, p. 49.
Format:	WAC <CMD?> <OP> <value>
Arguments:	<p><CMD?> is a query command in its usual syntax. The answer must consist of a single value. Supported commands are ONT? and DIO?</p> <p><OP> is the operator to be used. The following are allowable (controller firmware specific): "=" "<=" "<" ">" ">=" "!=".</p> <p><value> is the value to be compared with the response to <CMD?></p>

Response: none

Example: Send: MAC BEG AMC028
MVR A 1
WAC ONT? A = 1
MVR A -1
WAC ONT? A = 1
MAC START AMC028
MAC END
MAC START AMC028

Note: Macro AMC028 is recorded and then started. WAC ONT? A = 1 waits until the answer to ONT? A is A=1.

#5 (Poll Motion Status)

Description: Requests motion status of the connected axes.
Note that when no stage is connected to an axis (NOSTAGE is returned by CST?, p. 32), that axis is not included in the answer.

Format: #5 (single ASCII character number 5)

Arguments: none

Response: <uint> is the decimal sum of the following codes:
1=first connected axis is moving
2=second connected axis is moving
4=third connected axis is moving
etc. with 8, 16, 32, ... , 2^{15}

Examples: 0 indicates motion of all axes complete
3 indicates that the first and the second axis are moving (by default axes A and B)

#7 (Controller Ready?)

Description: Asks controller for ready status (tests if controller is ready to perform a new command).
Note: Use #5 instead of #7 to verify if motion has finished.

Format: #7 (single ASCII character number 7)

Arguments: none

Response:	B1h (ASCII character 177 = "±" in Windows) if controller network is ready B0h (ASCII character 176 = "°" in Windows) if controller network is not ready (e.g. performing a REF command)
Troubleshooting	The response characters may appear differently in non-Western character sets or other operating systems.

#8 (Macro running?)

Description:	Test whether a macro is running
Format:	#8 (single ASCII character number #8)
Arguments:	none
Response:	0 (ASCII character 48) no macro is running 1 (ASCII character 49) a macro is running on at least one of the controllers in the network

#24 (Stop)

Description:	<p>Stops all motion abruptly.</p> <p>This includes motion of all axes (MOV, MVR) and referencing motion (MNL, MPL, REF).</p> <p>Sets error code to 10.</p> <p>After all the axes are stopped, their target positions are set to the current positions.</p> <p>This command is identical in function to STP (p. 60), but only one character must be sent via the interface. Therefore #24 can also be used while the controller is performing time-consuming tasks.</p> <p>#24 always does a hard stop. Use HLT (p. 40) to stop a single axis or to stop axes smoothly.</p>
Format:	#24 (ASCII character 24)
Arguments:	none
Response:	none

9 Motion Parameter Overview

9.1 Parameter Handling

The hardware basics of the connected stage and—for C-863 DC motor controllers—the required closed-loop control settings are mirrored in parameters (provided by the PI Mercury GCS DLL). The parameter values have to be adjusted properly before initial operation of a stage.

With HPA? you can obtain a list of all available parameters with information about each (e.g. short descriptions). The current parameter values can be read with the SPA? command.

Using the "general" modification command SPA, parameters can be changed temporarily. In addition to the "general" modification command, there are some commands which change certain specific parameters (see table below).

CAUTION

Wrong values of the parameters may lead to improper operation or damage of your hardware. Be careful when changing parameters.

The interrelation of the hardware-dependent parameters 0x15, 0x16, 0x17, 0x2F and 0x30 is described in "Travel Range Adjustment" on p. 77.

C-863 DC motor controllers only: Find details regarding closed-loop control in "Control Options" in the C-863 User manual.

To store parameter values, save them to the *MercuryUserStages2.dat* stage database (see "Parameter Databases" on p. 70) or create an Autostart macro which sets the parameter values after each power-on or reset of the controller (see "Macro Storage on Controller" on p. 16 and the PIMikroMove™ manual for details).

9.2 Parameter Databases

NOTE

The GCS-based host software from PI uses multiple databases for stage parameters:

- PISTages2.dat contains parameter sets for all standard stages from PI and is automatically installed on the host PC with the setup. It cannot be edited; should changes in the file become necessary, you must obtain a new version from PI and install it on your host PC.
 - MercuryUserStages2.dat allows you to create and save your own stages. This database is created the first time you connect stages in the GCS-based host software (i.e. the first time the Mercury_qVST() or Mercury_CST() functions of the PI Mercury GCS library are used).
-

When you are working with the GCS-based host software from PI, you can select the suitable stage parameter set from one of the databases. The host software will then send the appropriate values to the controller.

In case you want to operate a stage with other parameters than stated in the PISTages2.dat or if you have a customized stage, add a new stage parameter set to MercuryUserStages2.dat. For further information, refer to “Functions for User-Defined Stages” in the PI Mercury GCS DLL manual, to “Tutorials - Frequently Asked Questions” in the PIMikroMove™ manual or to the PIStageEditor manual.

9.3 Parameter List

Parameter lines with white background apply to all controller versions, lines with light gray background apply to C-663 stepper motor versions only, and lines with dark gray background apply to C-863 DC motor versions only.

Parameter ID (hexa- decimal / decimal)	Data Type	Parameter Description	Possible Values/Notes
0x1 / 1	FLOAT	P-term for position control	0 to 32767
0x2 / 2	FLOAT	I-term for position control	0 to 32767
0x3 / 3	FLOAT	D-term for position control	0 to 32767
0x4 / 4	FLOAT	I-limit for position control	0 to 32767
0x8 / 8	FLOAT	Maximum position error (user unit)	0 to 32767 Used for stall detection. If the position error (i.e. the absolute value of the difference between current position and commanded position) in closed-loop operation exceeds the given maximum, the PI Mercury GCS DLL sets error code -1024 ("Motion error"), the servo will be switched off and the axis stops.
0xA / 10	FLOAT	Maximum closed-loop velocity (user unit/s)	> 0 Gives the maximum value for parameter 0x49.
0xB / 11	FLOAT	Current closed-loop acceleration (user unit/s ²)	Gives the current acceleration, limited by parameter 0x4A
0xE / 14	FLOAT	Numerator of the counts-per-physical-unit factor	1 to 2147483647 for each parameter. The counts-per-physical-unit factor determines the "user" unit for closed-loop motion commands. When you change this factor, all other parameters whose unit is based on the "user" unit must be adapted too, e.g. closed-loop velocity and parameters regarding the travel range. Note: To customize your physical unit use parameter 0x12 instead.
0xF / 15	FLOAT	Denominator of the counts-per-physical-unit factor	
0x11 / 17	FLOAT	Invert direction	-1 to invert the direction, else 1

Parameter ID (hexa- decimal / decimal)	Data Type	Parameter Description	Possible Values/Notes
0x12 / 18	FLOAT	Scaling factor	1.79769313486231E308 to 1.79769313486231E308 This factor can be used to change the physical unit of the stage, e.g. a factor of 25.4 converts a physical unit of mm to inches. It is recommended to use DFF to change this factor.
0x13 / 19	FLOAT	Rotary stage	1 = rotary stage, else 0
0x14 / 20	FLOAT	Stage has a reference	1 = the stage has a reference, else 0
0x15 / 21	FLOAT	MAX_TRAVEL_RANGE_ POS The maximum travel in positive direction (user unit)	"Soft limit", based on the home (zero) position. If the soft limit is smaller than the position value for the positive limit switch (which is given by the sum of the parameters 0x16 and 0x2F), the positive limit switch can not be used for referencing. Can be negative.
0x16 / 22	FLOAT	VALUE_AT_REF_POS The position value at the reference position (user unit)	The position value which is to be set when the mechanics performs a reference move to the reference switch. Must be set even if no reference switch is present in the mechanics because it is used to calculate the position values to be set after reference moves to the limit switches.
0x17 / 23	FLOAT	DISTANCE_REF_TO_N_ LIM The distance between reference switch and negative limit switch (user unit)	Represents the physical distance between the reference switch and the negative limit switch integrated in the mechanics. Must be set even if no reference switch is present in the mechanics because the position is set to the difference of VALUE_AT_REF_POS and DISTANCE_REF_TO_N_LIM when the mechanics performs a reference move to the negative limit switch.
0x18 / 24	FLOAT	Axis limit mode	0 = positive limit switch active high (pos-HI), negative limit switch active high (neg-HI) 1 = positive limit switch active low (pos-LO), neg-HI 2 = pos-HI, neg-LO 3 = pos-LO, neg-LO
0x19 / 25	FLOAT	Stage type	0 = DC motor 2 = Voice coil
0x1A / 26	FLOAT	Stage has brakes	0 = Stage has no brakes 1 = Stage has brakes

Parameter ID (hexa- decimal / decimal)	Data Type	Parameter Description	Possible Values/Notes
0x2F / 47	FLOAT	DISTANCE_REF_TO_P_LIM The distance between reference switch and positive limit switch (user unit)	Represents the physical distance between the reference switch and the positive limit switch integrated in the mechanics. Must be set even if no reference switch is present in the mechanics because the position is set to the sum of VALUE_AT_REF_POS and DISTANCE_REF_TO_P_LIM when the mechanics performs a reference move to the positive limit switch.
0x30 / 48	FLOAT	MAX_TRAVEL_RANGE_NEG The maximum travel in negative direction (user unit)	"Soft limit", based on the home (zero) position. If the soft limit is larger than the position value for the negative limit switch (which is given by the difference of the parameters 0x16 and 0x17), the negative limit switch can not be used for referencing. Can be negative.
0x31 / 49	FLOAT	Invert the reference	1 = invert the reference, else 0
0x32 / 50	FLOAT	Stage has limit switches; enables / disables the stopping of the motion at the limit switches	0 = Stage has limit switches 1 = Stage has no limit switches
0x36 / 54	FLOAT	Settle window (counts)	0 to 2^{31} The settle window is centered around the target position. The on-target status becomes "true" when the current position stays in this window for at least the settle time (parameter 0x3F).
0x3C / 60	FLOAT	Stage name	Maximum 31 characters
0x3F / 63	FLOAT	Settle time (s)	0.000 to 1.000 s; Used for on-target detection: The on-target status becomes "true" when the current position stays in the settle window (parameter 0x36) for at least the settle time. If the settle time is set to 0, then the axis is on target when the trajectory has finished, irrespective of the current position.
0x40 / 64	FLOAT	Hold current (mA)	When motion has finished, after a given delay time (parameter 0x42) the hold current is applied. Note that normally the hold current is about 25% of the drive current which allows to keep the temperature of the stepper motor down, close to room temperature.
0x41 / 65	FLOAT	Drive current (mA)	Gives the motor phase current (drive current) for moving state.
0x42 / 66	FLOAT	Hold time (ms)	Gives the hold time, that is the delay time between completion of a move and the activation of the hold current.

Parameter ID (hexa- decimal / decimal)	Data Type	Parameter Description	Possible Values/Notes
0x43 / 67	FLOAT	Max current (mA)	Maximum value for hold current (parameter 0x40) and drive current (parameter 0x41)
0x49 / 73	FLOAT	Current closed-loop velocity (user unit/s)	Gives the current velocity, limited by parameter 0xA Can also be changed with VEL
0x4A / 74	FLOAT	Maximum closed-loop acceleration (user unit/s ²)	Gives the maximum value for parameter 0xB
0x50 / 80	FLOAT	Velocity for reference move (user unit/s)	Gives the maximum velocity to be used for reference moves with REF, MPL, MNL; if set to 0, reference moves are not possible
0x94 / 148	FLOAT	Notch filter frequency (Hz)	40 to 10000 The corresponding frequency component in the control value is reduced to compensate for unwanted resonances in the mechanics. Only active in closed-loop operation. Should normally not be changed (try to change only with very high loads).
0x95 / 149	FLOAT	Notch filter edge	0.4 to 10 Gives the slope of the filter edge. Do not change.
0x100 / 256	FLOAT	Trackball mode	0 = trackball disabled 1 = trackball enabled Connect the digital TTL signals A and B (also referred to as quadrature signals) provided by the trackball to the digital input lines 3 and 4 of the "I/O" socket (pinout see User manual) on the Mercury™ controller. The lines are terminated by 10k to GND. While the trackball mode is active, move commands or joystick control are not accepted.
0x101 / 257	FLOAT	Trackball increment (counts)	Gives the increment length for trackball operation, minimum value = 1. Each signal transition on the trackball lines will shift the target by the given number of counts if the trackball is enabled by parameter 0x100.
0x102 / 258	FLOAT	Trackball filter	0 to 255 The parameter determines how often the same signal level must be read before the signal transition on the trackball lines is accepted. This suppresses transient noise and allows for stable reading. If 0, the filter is disabled.
0x110 / 272	FLOAT	Control mode	Gives the control mode to be applied, see C-863 User manual for details 0 = Position control (force control is OFF) 1 = Force control (position control is OFF) 2 = Position control and force control are ONs

Parameter ID (hexa- decimal / decimal)	Data Type	Parameter Description	Possible Values/Notes
0x111 / 273	FLOAT	Force control target	0 to 1024 Used in “force control” or “position and force control” mode where it defines the force target that shall be reached and maintained by varying the position.
0x112 / 274	FLOAT	P-term for force control algorithm	0 to 65,535
0x113 / 275	FLOAT	I-term for force control algorithm	0 to 65,535
0x114 / 276	FLOAT	I-limit for force control algorithm	0 to 65,535
0x115 / 277	FLOAT	D-term for force control algorithm	0 to 65,535
0x116 / 278	FLOAT	Offset to force sensor input	Gives an offset to the input of the additional (force) sensor. Note that this offset is already included if you query the current value of the additional sensor via parameter 0x11A.
0x117 / 279	FLOAT	Low pass filter for force sensor input (Hz)	40 to 10000 Gives the frequency of the low pass filter on the input of the additional (force) sensor used for the “force control” or “position and force control” modes.
0x11A / 282	FLOAT	Current force read-only parameter	-2048 to 2048 Current value of the additional sensor used for the force control loop, i.e. the input signal on pin 2 of the “Joystick” socket. The input voltage range is -10 to +10 V. The voltage value is converted by a 12 bit A/D converter. The value reported includes the offset set with parameter 0x116, and was already filtered by the low pass filter set with parameter 0x117. Furthermore, any sign settings made with the FS (Set Force Sign) command of the native command set are already applied to the signal (see MercuryNativeCommands Manual MS176E for details).
0x120 / 288	FLOAT	Joystick offset	Gives an offset to the analog input provided by the joystick. This allows to correct the neutral (center) value of the joystick reading, e.g. if the actual joystick reading is 120, with bias 8 the reading used would be 128.

9.4 Transmission Ratio and Scaling Factor

The physical unit used for the stages (i.e. for the axes of the controller) results from the following interrelation of some stage parameters:

$$PU = \left(Cnt / \frac{CpuN}{CpuD} \right) \times SF$$

$$Cnt = (PU / SF) \times \frac{CpuN}{CpuD}$$

Name	Number*	Description
<i>PU</i>	-	Physical Unit
<i>Cnt</i>	-	Counts
<i>CpuN</i>	0xE	Numerator of the counts per physical unit factor
<i>CpuD</i>	0xF	Denominator of the counts per physical unit factor
<i>SF</i>	0x12	Scaling factor**

*Number means the parameter ID for SPA and SPA? and in the list in Section 71.

**See the DFF command.

The "Counts per physical unit factor" which results from parameter 0xE divided by parameter 0xF includes the physical transmission ratio and the resolution of the stage.

CAUTION

To customize the physical unit of a stage do not change parameter 0xE and parameter 0xF but use DFF instead. Although DFF has the same effect as changing parameter 0x12 with SPA, you should only use DFF and not SPA to modify the scaling factor.

Example: If you set with DFF a value of 25.4 for an axis, the physical unit for this axis is converted from mm to inches.

9.5 Travel Range Adjustment

The figures below give a universal hardware scheme of a positioning stage with incremental sensor, reference and limit switches. To work with such a stage, the stage parameters (provided by the PI Mercury GCS DLL) must be adjusted properly (see "Parameter Handling" on p. 69 for how to modify parameter values).

In the example shown in the first figure, the travel range, i.e. the distance from negative to positive limit switch is 20 mm, the distance between the negative limit switch and the reference switch is 8 mm, and the distance between reference switch and positive limit switch is 12 mm. These hardware properties are represented by the following parameters:

DISTANCE_REF_TO_N_LIM (parameter ID 0x17) = 8

DISTANCE_REF_TO_P_LIM (parameter ID 0x2F) = 12

To allow for flexible localization of the home position (0), a special parameter is provided. It gives the offset between reference switch and home position which is to be valid for the stage after a reference move (see below). In the example, the home position is to be located at the negative limit switch after a reference move, and hence the offset between reference switch and home position is 8 mm.

VALUE_AT_REF_POS (parameter ID 0x16) = 8

To allow for absolute moves, either an absolute "initial" position can be set with the POS command, or the stage can perform a reference move to a known position where a defined position value will be set as the current position (see "Referencing" on p. 11 for further details). By default, a reference move is required. In the example, known positions for reference moves are given by the reference switch and the limit switches. Depending on the switch used for the reference move, a certain combination of the above-mentioned parameters is used to calculate the position to be set at the end of the move:

- Reference switch (REF command): the stage is moved to the reference switch, and the value of VALUE_AT_REF_POS is set as the current position.
- Negative limit switch (MNL command): the stage is moved to the negative limit switch and the difference of VALUE_AT_REF_POS and DISTANCE_REF_TO_N_LIM is set as the current position (can be negative).
- Positive limit switch (MPL command): the stage is moved to the positive limit switch and the sum of VALUE_AT_REF_POS and DISTANCE_REF_TO_P_LIM is set as the current position.

It is furthermore possible to set "soft limits" which establish a "safety distance" which the stage will not enter on both ends of the travel range. Those soft limits always refer to the current home position (0; in the example located at the negative limit switch after a reference move). The soft limits are to be deactivated in the example so that the corresponding parameters must be as follows:

MAX_TRAVEL_RANGE_POS (parameter ID 0x15) = 20 mm

MAX_TRAVEL_RANGE_NEG (parameter ID 0x30) = 0 mm

(This means that the stage can move 20 mm in positive direction, starting from the home position, and 0 mm in negative direction, starting from the home position.)

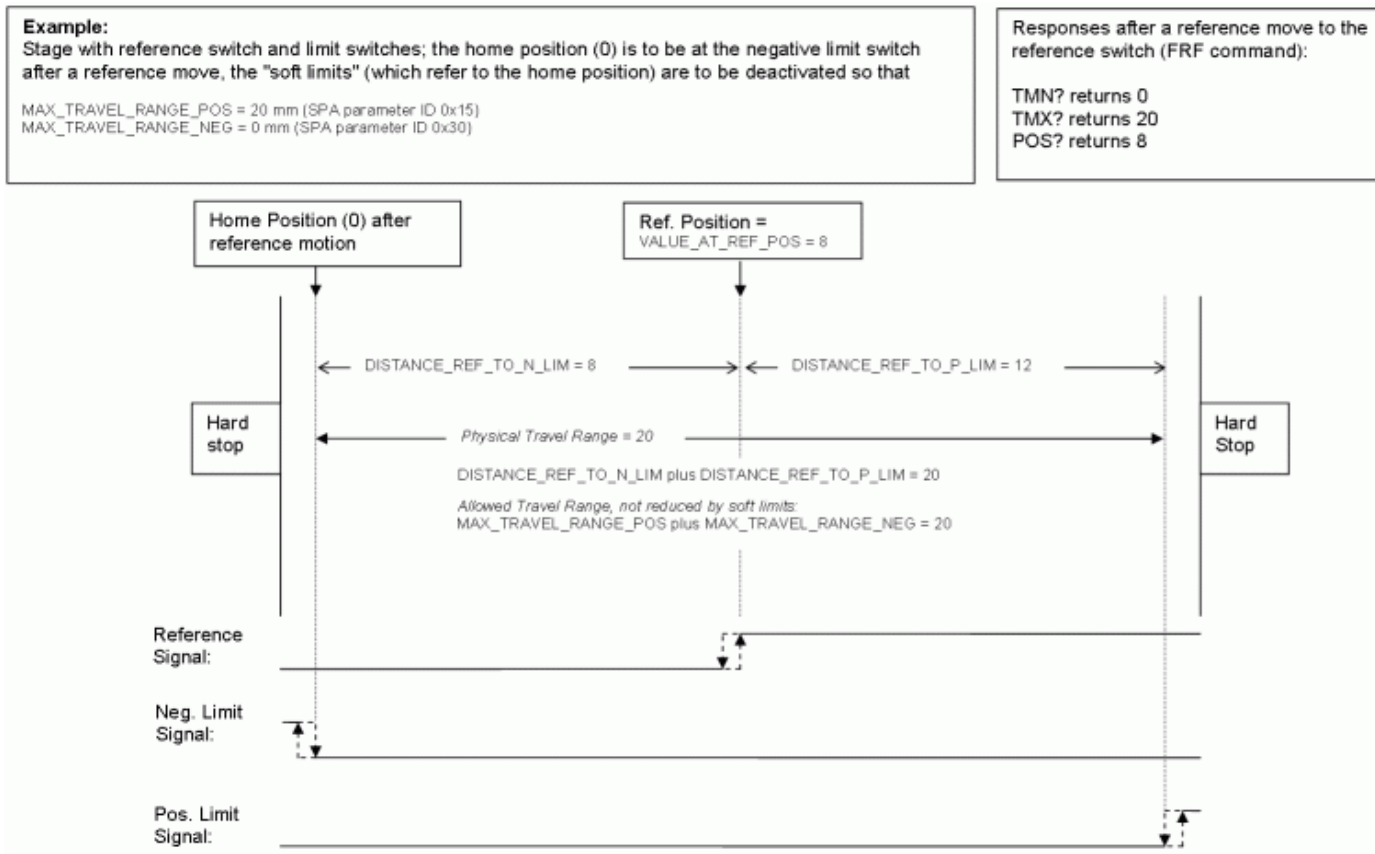


Figure 1: Positioning stage and corresponding controller parameters

Now in the same example, a "safety distance" is to be established on both ends of the travel range by setting soft limits, and the home position is to be located at about 1/3 of the distance between the new negative end of the travel range and the reference switch. The limit switches can not be used for reference moves anymore.

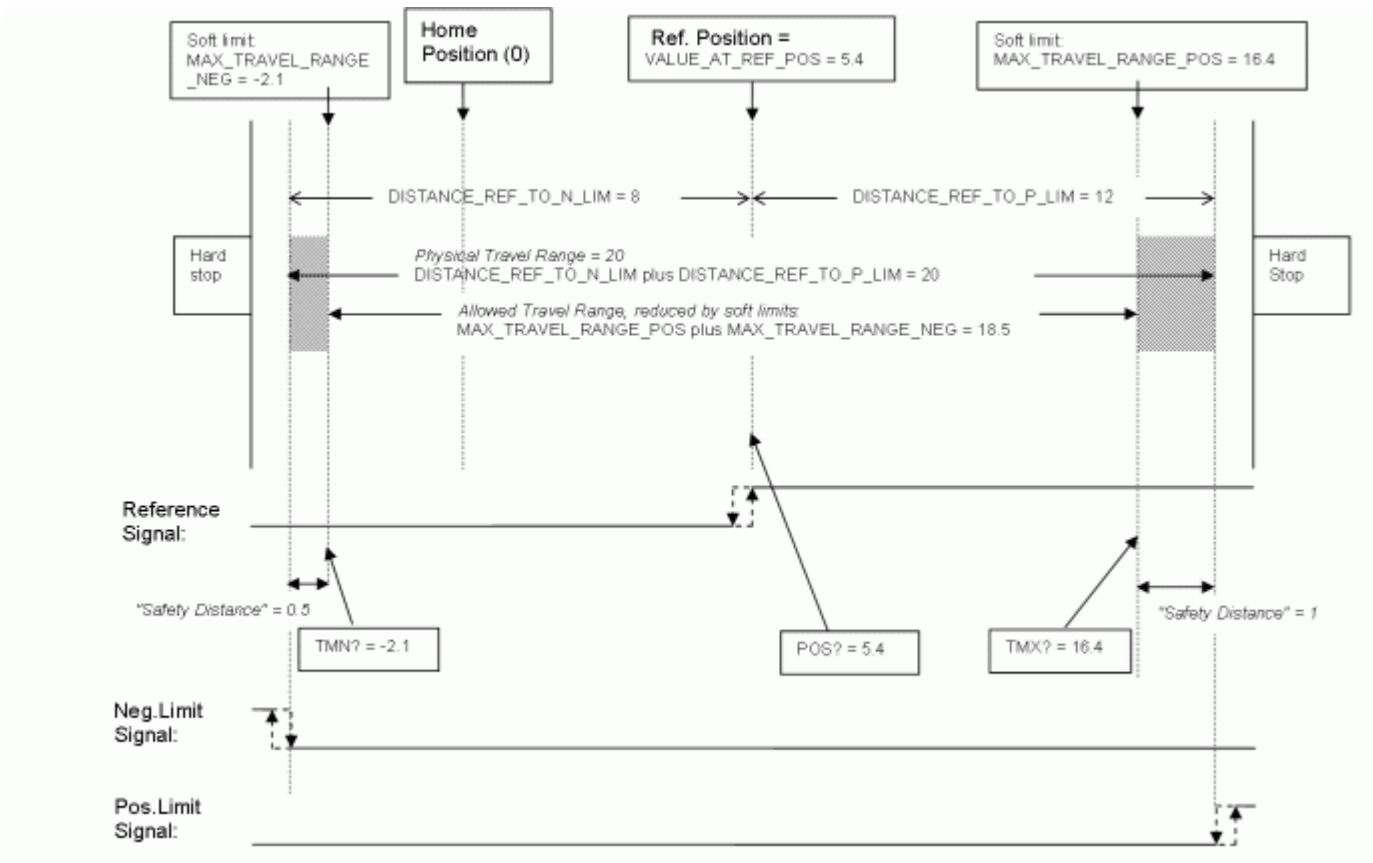


Figure 2: Positioning stage, soft limits set in the controller to reduce the travel range

After the stage was referenced again by moving it to the reference switch (with REF), the following responses will be given:

TMN? returns -2.1

TMX? returns 16.4

POS? returns 5.4

CAUTION

If the soft limits (MAX_TRAVEL_RANGE_POS and MAX_TRAVEL_RANGE_NEG) are used to reduce the travel range, the limit switches can not be used for reference moves. MNL and MPL will provoke an error message, and only the reference switch can be used for a reference move (REF).

Be careful when setting the values for VALUE_AT_REF_POS, MAX_TRAVEL_RANGE_POS and MAX_TRAVEL_RANGE_NEG because there is no plausibility check.

The soft limits may not be outside of the physical travel range:

$$\text{MAX_TRAVEL_RANGE_POS} \leq \text{DISTANCE_REF_TO_P_LIM} + \text{VALUE_AT_REF_POS}$$
$$\text{MAX_TRAVEL_RANGE_NEG} \geq \text{VALUE_AT_REF_POS} - \text{DISTANCE_REF_TO_N_LIM}$$

Otherwise, reference moves to the limit switches would have incorrect results because the values of the soft limits would be set at the end of the referencing procedure.

Be careful when referencing the stage by setting an initial absolute position with POS since the values for MAX_TRAVEL_RANGE_POS and MAX_TRAVEL_RANGE_NEG are not adapted. In the worst case, the soft limits will now be outside of the physical travel range, and the stage will no longer be able to move since the move commands check the soft limit settings.

9.6 Updating PISTages2.dat

To install the latest version of PISTages2.dat from the PI Website proceed as follows:

- 1 On the www.pi.ws front page, click on *Download/Support* in the *Service* section on the left
- 2 On the *Download/Support* page, click on *Manuals and Software*
- 3 Click on *Download* in the navigation bar across the top (no login or password is required)
- 4 Click on the *General Software* category
- 5 Click on *PI Stages*
- 6 Click the download button below *PIStages2.dat*

- 7 In the download window, switch to the ...\\PI\\GcsTranslator directory. The location of the PI directory is that specified upon installation, by default C:\\Documents and Settings\\All Users\\Application Data (Windows XP) or C:\\ProgramData (Windows Vista) (may differ in other-language Windows versions).
Hint: You can identify the path using the *Version Info...* item in the controller menu in PIMikroMove™.
- 8 If desired, rename the existing PISTages2.dat (if present) so as to preserve a copy for safety reasons
- 9 Download the file from the server as PISTages2.dat

10 Error Codes

The error codes listed here are those of the PI General Command Set. As such, some may be not relevant to your controller and will simply never occur.

Controller Errors

0	PI_CNTR_NO_ERROR	No error
1	PI_CNTR_PARAM_SYNTAX	Parameter syntax error
2	PI_CNTR_UNKNOWN_COMMAND	Unknown command
3	PI_CNTR_COMMAND_TOO_LONG	Command length out of limits or command buffer overrun
4	PI_CNTR_SCAN_ERROR	Error while scanning
5	PI_CNTR_MOVE_WITHOUT_REF_OR_NO_SERVO	Unallowable move attempted on unreferenced axis, or move attempted with servo off
6	PI_CNTR_INVALID_SGA_PARAM	Parameter for SGA not valid
7	PI_CNTR_POS_OUT_OF_LIMITS	Position out of limits
8	PI_CNTR_VEL_OUT_OF_LIMITS	Velocity out of limits
9	PI_CNTR_SET_PIVOT_NOT_POSSIBLE	Attempt to set pivot point while U,V and W not all 0
10	PI_CNTR_STOP	Controller was stopped by command
11	PI_CNTR_SST_OR_SCAN_RANGE	Parameter for SST or for one of the embedded scan algorithms out of range
12	PI_CNTR_INVALID_SCAN_AXES	Invalid axis combination for fast scan
13	PI_CNTR_INVALID_NAV_PARAM	Parameter for NAV out of range
14	PI_CNTR_INVALID_ANALOG_INPUT	Invalid analog channel
15	PI_CNTR_INVALID_AXIS_IDENTIFIER	Invalid axis identifier
16	PI_CNTR_INVALID_STAGE_NAME	Unknown stage name
17	PI_CNTR_PARAM_OUT_OF_RANGE	Parameter out of range



18	PI_CNTR_INVALID_MACRO_NAME	Invalid macro name
19	PI_CNTR_MACRO_RECORD	Error while recording macro
20	PI_CNTR_MACRO_NOT_FOUND	Macro not found
21	PI_CNTR_AXIS_HAS_NO_BRAKE	Axis has no brake
22	PI_CNTR_DOUBLE_AXIS	Axis identifier specified more than once
23	PI_CNTR_ILLEGAL_AXIS	Illegal axis
24	PI_CNTR_PARAM_NR	Incorrect number of parameters
25	PI_CNTR_INVALID_REAL_NR	Invalid floating point number
26	PI_CNTR_MISSING_PARAM	Parameter missing
27	PI_CNTR_SOFT_LIMIT_OUT_OF_RANGE	Soft limit out of range
28	PI_CNTR_NO_MANUAL_PAD	No manual pad found
29	PI_CNTR_NO_JUMP	No more step-response values
30	PI_CNTR_INVALID_JUMP	No step-response values recorded
31	PI_CNTR_AXIS_HAS_NO_REFERENCE	Axis has no reference sensor
32	PI_CNTR_STAGE_HAS_NO_LIM_SWITCH	Axis has no limit switch
33	PI_CNTR_NO_RELAY_CARD	No relay card installed
34	PI_CNTR_CMD_NOT_ALLOWED_FOR_STAGE	Command not allowed for selected stage(s)
35	PI_CNTR_NO_DIGITAL_INPUT	No digital input installed
36	PI_CNTR_NO_DIGITAL_OUTPUT	No digital output configured
37	PI_CNTR_NO_MCM	No more MCM responses
38	PI_CNTR_INVALID_MCM	No MCM values recorded

39	PI_CNTR_INVALID_CNTR_NUMBER	Controller number invalid
40	PI_CNTR_NO_JOYSTICK_CONNECTED	No joystick configured
41	PI_CNTR_INVALID_EGE_AXIS	Invalid axis for electronic gearing, axis can not be slave
42	PI_CNTR_SLAVE_POSITION_OUT_OF_RANGE	Position of slave axis is out of range
43	PI_CNTR_COMMAND_EGE_SLAVE	Slave axis cannot be commanded directly when electronic gearing is enabled
44	PI_CNTR_JOYSTICK_CALIBRATION_FAILED	Calibration of joystick failed
45	PI_CNTR_REFERENCING_FAILED	Referencing failed
46	PI_CNTR_OPM_MISSING	OPM (Optical Power Meter) missing
47	PI_CNTR_OPM_NOT_INITIALIZED	OPM (Optical Power Meter) not initialized or cannot be initialized
48	PI_CNTR_OPM_COM_ERROR	OPM (Optical Power Meter) Communication Error
49	PI_CNTR_MOVE_TO_LIMIT_SWITCH_FAILED	Move to limit switch failed
50	PI_CNTR_REF_WITH_REF_DISABLED	Attempt to reference axis with referencing disabled
51	PI_CNTR_AXIS_UNDER_JOYSTICK_CONTROL	Selected axis is controlled by joystick
52	PI_CNTR_COMMUNICATION_ERROR	Controller detected communication error
53	PI_CNTR_DYNAMIC_MOVE_IN_PROCESS	MOV! motion still in progress
54	PI_CNTR_UNKNOWN_PARAMETER	Unknown parameter
55	PI_CNTR_NO_REP_RECORDED	No commands were recorded with REP
56	PI_CNTR_INVALID_PASSWORD	Password invalid
57	PI_CNTR_INVALID_RECORDER_CHAN	Data Record Table does not exist
58	PI_CNTR_INVALID_RECORDER_SRC_OPT	Source does not exist; number too low or too high



59	PI_CNTR_INVALID_RECORDER_SRC_CHAN	Source Record Table number too low or too high
60	PI_CNTR_PARAM_PROTECTION	Protected Param: current Command Level (CCL) too low
61	PI_CNTR_AUTOZERO_RUNNING	Command execution not possible while Autozero is running
62	PI_CNTR_NO_LINEAR_AXIS	Autozero requires at least one linear axis
63	PI_CNTR_INIT_RUNNING	Initialization still in progress
64	PI_CNTR_READ_ONLY_PARAMETER	Parameter is read-only
65	PI_CNTR_PAM_NOT_FOUND	Parameter not found in non-volatile memory
66	PI_CNTR_VOL_OUT_OF_LIMITS	Voltage out of limits
67	PI_CNTR_WAVE_TOO_LARGE	Not enough memory available for requested wave curve
68	PI_CNTR_NOT_ENOUGH_DDL_MEMORY	Not enough memory available for DDL table; DDL can not be started
69	PI_CNTR_DDL_TIME_DELAY_TOO_LARGE	Time delay larger than DDL table; DDL can not be started
70	PI_CNTR_DIFFERENT_ARRAY_LENGTH	The requested arrays have different lengths; query them separately
71	PI_CNTR_GEN_SINGLE_MODE_RESTART	Attempt to restart the generator while it is running in single step mode
72	PI_CNTR_ANALOG_TARGET_ACTIVE	Motion commands and wave generator activation are not allowed when analog target is active
73	PI_CNTR_WAVE_GENERATOR_ACTIVE	Motion commands are not allowed when wave generator is active
74	PI_CNTR_AUTOZERO_DISABLED	No sensor channel or no piezo channel connected to selected axis (sensor and piezo matrix)
75	PI_CNTR_NO_WAVE_SELECTED	Generator started (WGO) without having selected a wave table (WSL).



76	PI_CNTR_IF_BUFFER_OVERRUN	Interface buffer did overrun and command couldn't be received correctly
77	PI_CNTR_NOT_ENOUGH_RECORDED_DATA	Data Record Table does not hold enough recorded data
78	PI_CNTR_TABLE_DEACTIVATED	Data Record Table is not configured for recording
79	PI_CNTR_OPENLOOP_VALUE_SET_WHEN_SERVO_ON	Open-loop commands (SVA, SVR) are not allowed when servo is on
80	PI_CNTR_RAM_ERROR	Hardware error affecting RAM
81	PI_CNTR_MACRO_UNKNOWN_COMMAND	Not macro command
82	PI_CNTR_MACRO_PC_ERROR	Macro counter out of range
83	PI_CNTR_JOYSTICK_ACTIVE	Joystick is active
84	PI_CNTR_MOTOR_IS_OFF	Motor is off
85	PI_CNTR_ONLY_IN_MACRO	Macro-only command
86	PI_CNTR_JOYSTICK_UNKNOWN_AXIS	Invalid joystick axis
87	PI_CNTR_JOYSTICK_UNKNOWN_ID	Joystick unknown
88	PI_CNTR_REF_MODE_IS_ON	Move without referenced stage
89	PI_CNTR_NOT_ALLOWED_IN_CURRENT_MOTION_MODE	Command not allowed in current motion mode
90	PI_CNTR_DIO_AND_TRACING_NOT_POSSIBLE	No tracing possible while digital IOs are used on this HW revision. Reconnect to switch operation mode.
91	PI_CNTR_COLLISION	Move not possible, would cause collision
100	PI_LABVIEW_ERROR	PI LabVIEW driver reports error. See source control for details.
200	PI_CNTR_NO_AXIS	No stage connected to axis
201	PI_CNTR_NO_AXIS_PARAM_FILE	File with axis parameters not found



202	PI_CNTR_INVALID_AXIS_PARAM_FILE	Invalid axis parameter file
203	PI_CNTR_NO_AXIS_PARAM_BACKUP	Backup file with axis parameters not found
204	PI_CNTR_RESERVED_204	PI internal error code 204
205	PI_CNTR_SMO_WITH_SERVO_ON	SMO with servo on
206	PI_CNTR_UUDECODE_INCOMPLETE_HEADER	uudecode: incomplete header
207	PI_CNTR_UUDECODE_NOTHING_TO_DECODE	uudecode: nothing to decode
208	PI_CNTR_UUDECODE_ILLEGAL_FORMAT	uudecode: illegal UUE format
209	PI_CNTR_CRC32_ERROR	CRC32 error
210	PI_CNTR_ILLEGAL_FILENAME	Illegal file name (must be 8-0 format)
211	PI_CNTR_FILE_NOT_FOUND	File not found on controller
212	PI_CNTR_FILE_WRITE_ERROR	Error writing file on controller
213	PI_CNTR_DTR_HINDERS_VELOCITY_CHANGE	VEL command not allowed in DTR Command Mode
214	PI_CNTR_POSITION_UNKNOWN	Position calculations failed
215	PI_CNTR_CONN_POSSIBLY_BROKEN	The connection between controller and stage may be broken
216	PI_CNTR_ON_LIMIT_SWITCH	The connected stage has driven into a limit switch, some controllers need CLR to resume operation
217	PI_CNTR_UNEXPECTED_STRUT_STOP	Strut test command failed because of an unexpected strut stop
218	PI_CNTR_POSITION_BASED_ON_ESTIMATION	While MOV! is running position can only be estimated!
219	PI_CNTR_POSITION_BASED_ON_INTERPOLATION	Position was calculated during MOV motion
230	PI_CNTR_INVALID_HANDLE	Invalid handle



231	PI_CNTR_NO_BIOS_FOUND	No bios found
232	PI_CNTR_SAVE_SYS_CFG_FAILED	Save system configuration failed
233	PI_CNTR_LOAD_SYS_CFG_FAILED	Load system configuration failed
301	PI_CNTR_SEND_BUFFER_OVERFLOW	Send buffer overflow
302	PI_CNTR_VOLTAGE_OUT_OF_LIMITS	Voltage out of limits
303	PI_CNTR_OPEN_LOOP_MOTION_SET_WHEN_SERVO_ON	Open-loop motion attempted when servo ON
304	PI_CNTR_RECEIVING_BUFFER_OVERFLOW	Received command is too long
305	PI_CNTR_EEPROM_ERROR	Error while reading/writing EEPROM
306	PI_CNTR_I2C_ERROR	Error on I2C bus
307	PI_CNTR_RECEIVING_TIMEOUT	Timeout while receiving command
308	PI_CNTR_TIMEOUT	A lengthy operation has not finished in the expected time
309	PI_CNTR_MACRO_OUT_OF_SPACE	Insufficient space to store macro
310	PI_CNTR_EUI_OLDVERSION_CFGDATA	Configuration data has old version number
311	PI_CNTR_EUI_INVALID_CFGDATA	Invalid configuration data
333	PI_CNTR_HARDWARE_ERROR	Internal hardware error
400	PI_CNTR_WAV_INDEX_ERROR	Wave generator index error
401	PI_CNTR_WAV_NOT_DEFINED	Wave table not defined
402	PI_CNTR_WAV_TYPE_NOT_SUPPORTED	Wave type not supported
403	PI_CNTR_WAV_LENGTH_EXCEEDS_LIMIT	Wave length exceeds limit
404	PI_CNTR_WAV_PARAMETER_NR	Wave parameter number error
405	PI_CNTR_WAV_PARAMETER_OUT_OF_LIMIT	Wave parameter out of range

406	PI_CNTR_WGO_BIT_NOT_SUPPORTED	WGO command bit not supported
502	PI_CNTR_REDUNDANCY_LIMIT_EXCEEDED	Position consistency check failed
503	PI_CNTR_COLLISION_SWITCH_ACTIVATED	Hardware collision sensor(s) are activated
504	PI_CNTR_FOLLOWING_ERROR	Strut following error occurred, e.g. caused by overload or encoder failure
555	PI_CNTR_UNKNOWN_ERROR	BasMac: unknown controller error
601	PI_CNTR_NOT_ENOUGH_MEMORY	not enough memory
602	PI_CNTR_HW_VOLTAGE_ERROR	hardware voltage error
603	PI_CNTR_HW_TEMPERATURE_ERROR	hardware temperature out of range
1000	PI_CNTR_TOO_MANY_NESTED_MACROS	Too many nested macros
1001	PI_CNTR_MACRO_ALREADY_DEFINED	Macro already defined
1002	PI_CNTR_NO_MACRO_RECORDING	Macro recording not activated
1003	PI_CNTR_INVALID_MAC_PARAM	Invalid parameter for MAC
1004	PI_CNTR_RESERVED_1004	PI internal error code 1004
1005	PI_CNTR_CONTROLLER_BUSY	Controller is busy with some lengthy operation (e.g. reference move, fast scan algorithm)
2000	PI_CNTR_ALREADY_HAS_SERIAL_NUMBER	Controller already has a serial number
4000	PI_CNTR_SECTOR_ERASE_FAILED	Sector erase failed
4001	PI_CNTR_FLASH_PROGRAM_FAILED	Flash program failed
4002	PI_CNTR_FLASH_READ_FAILED	Flash read failed
4003	PI_CNTR_HW_MATCHCODE_ERROR	HW match code missing/invalid
4004	PI_CNTR_FW_MATCHCODE_ERROR	FW match code missing/invalid



4005	PI_CNTR_HW_VERSION_ERROR	HW version missing/invalid
4006	PI_CNTR_FW_VERSION_ERROR	FW version missing/invalid
4007	PI_CNTR_FW_UPDATE_ERROR	FW update failed
5200	PI_CNTR_AXIS_NOT_CONFIGURED	Axis must be configured for this action

Interface Errors

0	COM_NO_ERROR	No error occurred during function call
-1	COM_ERROR	Error during com operation (could not be specified)
-2	SEND_ERROR	Error while sending data
-3	REC_ERROR	Error while receiving data
-4	NOT_CONNECTED_ERROR	Not connected (no port with given ID open)
-5	COM_BUFFER_OVERFLOW	Buffer overflow
-6	CONNECTION_FAILED	Error while opening port
-7	COM_TIMEOUT	Timeout error
-8	COM_MULTILINE_RESPONSE	There are more lines waiting in buffer
-9	COM_INVALID_ID	There is no interface or DLL handle with the given ID
-10	COM_NOTIFY_EVENT_ERROR	Event/message for notification could not be opened
-11	COM_NOT_IMPLEMENTED	Function not supported by this interface type
-12	COM_ECHO_ERROR	Error while sending "echoed" data
-13	COM_GPIB_EDVR	IEEE488: System error



-14	COM_GPIB_ECIC	IEEE488: Function requires GPIB board to be CIC
-15	COM_GPIB_ENOL	IEEE488: Write function detected no listeners
-16	COM_GPIB_EADR	IEEE488: Interface board not addressed correctly
-17	COM_GPIB_EARG	IEEE488: Invalid argument to function call
-18	COM_GPIB_ESAC	IEEE488: Function requires GPIB board to be SAC
-19	COM_GPIB_EABO	IEEE488: I/O operation aborted
-20	COM_GPIB_ENEB	IEEE488: Interface board not found
-21	COM_GPIB_EDMA	IEEE488: Error performing DMA
-22	COM_GPIB_EOIP	IEEE488: I/O operation started before previous operation completed
-23	COM_GPIB_ECAP	IEEE488: No capability for intended operation
-24	COM_GPIB_EFSO	IEEE488: File system operation error
-25	COM_GPIB_EBUS	IEEE488: Command error during device call
-26	COM_GPIB_ESTB	IEEE488: Serial poll-status byte lost
-27	COM_GPIB_ESRQ	IEEE488: SRQ remains asserted
-28	COM_GPIB_ETAB	IEEE488: Return buffer full
-29	COM_GPIB_ELCK	IEEE488: Address or board locked
-30	COM_RS_INVALID_DATA_BITS	RS-232: 5 data bits with 2 stop bits is an invalid combination, as is 6, 7, or 8 data bits with 1.5 stop bits
-31	COM_ERROR_RS_SETTINGS	RS-232: Error configuring the COM port

-32	COM_INTERNAL_RESOURCES_ERROR	Error dealing with internal system resources (events, threads, ...)
-33	COM_DLL_FUNC_ERROR	A DLL or one of the required functions could not be loaded
-34	COM_FTDIUSB_INVALID_HANDLE	FTDIUSB: invalid handle
-35	COM_FTDIUSB_DEVICE_NOT_FOUND	FTDIUSB: device not found
-36	COM_FTDIUSB_DEVICE_NOT_OPENED	FTDIUSB: device not opened
-37	COM_FTDIUSB_IO_ERROR	FTDIUSB: IO error
-38	COM_FTDIUSB_INSUFFICIENT_RESOURCES	FTDIUSB: insufficient resources
-39	COM_FTDIUSB_INVALID_PARAMETER	FTDIUSB: invalid parameter
-40	COM_FTDIUSB_INVALID_BAUD_RATE	FTDIUSB: invalid baud rate
-41	COM_FTDIUSB_DEVICE_NOT_OPENED_FOR_ERASE	FTDIUSB: device not opened for erase
-42	COM_FTDIUSB_DEVICE_NOT_OPENED_FOR_WRITE	FTDIUSB: device not opened for write
-43	COM_FTDIUSB_FAILED_TO_WRITE_DEVICE	FTDIUSB: failed to write device
-44	COM_FTDIUSB_EEPROM_READ_FAILED	FTDIUSB: EEPROM read failed
-45	COM_FTDIUSB_EEPROM_WRITE_FAILED	FTDIUSB: EEPROM write failed
-46	COM_FTDIUSB_EEPROM_ERASE_FAILED	FTDIUSB: EEPROM erase failed
-47	COM_FTDIUSB_EEPROM_NOT_PRESENT	FTDIUSB: EEPROM not present
-48	COM_FTDIUSB_EEPROM_NOT_PROGRAMMED	FTDIUSB: EEPROM not programmed
-49	COM_FTDIUSB_INVALID_ARGS	FTDIUSB: invalid arguments
-50	COM_FTDIUSB_NOT_SUPPORTED	FTDIUSB: not supported
-51	COM_FTDIUSB_OTHER_ERROR	FTDIUSB: other error



-52	COM_PORT_ALREADY_OPEN	Error while opening the COM port: was already open
-53	COM_PORT_CHECKSUM_ERROR	Checksum error in received data from COM port
-54	COM_SOCKET_NOT_READY	Socket not ready, you should call the function again
-55	COM_SOCKET_PORT_IN_USE	Port is used by another socket
-56	COM_SOCKET_NOT_CONNECTED	Socket not connected (or not valid)
-57	COM_SOCKET_TERMINATED	Connection terminated (by peer)
-58	COM_SOCKET_NO_RESPONSE	Can't connect to peer
-59	COM_SOCKET_INTERRUPTED	Operation was interrupted by a nonblocked signal
-60	COM_PCI_INVALID_ID	No device with this ID is present
-61	COM_PCI_ACCESS_DENIED	Driver could not be opened (on Vista: run as administrator!)

DLL Errors

-1001	PI_UNKNOWN_AXIS_IDENTIFIER	Unknown axis identifier
-1002	PI_NR_NAV_OUT_OF_RANGE	Number for NAV out of range--must be in [1,10000]
-1003	PI_INVALID_SGA	Invalid value for SGA--must be one of 1, 10, 100, 1000
-1004	PI_UNEXPECTED_RESPONSE	Controller sent unexpected response
-1005	PI_NO_MANUAL_PAD	No manual control pad installed, calls to SMA and related commands are not allowed
-1006	PI_INVALID_MANUAL_PAD_KNOB	Invalid number for manual control pad knob
-1007	PI_INVALID_MANUAL_PAD_AXIS	Axis not currently controlled by a manual control pad



-1008	PI_CONTROLLER_BUSY	Controller is busy with some lengthy operation (e.g. reference move, fast scan algorithm)
-1009	PI_THREAD_ERROR	Internal error--could not start thread
-1010	PI_IN_MACRO_MODE	Controller is (already) in macro mode--command not valid in macro mode
-1011	PI_NOT_IN_MACRO_MODE	Controller not in macro mode--command not valid unless macro mode active
-1012	PI_MACRO_FILE_ERROR	Could not open file to write or read macro
-1013	PI_NO_MACRO_OR_EMPTY	No macro with given name on controller, or macro is empty
-1014	PI_MACRO_EDITOR_ERROR	Internal error in macro editor
-1015	PI_INVALID_ARGUMENT	One or more arguments given to function is invalid (empty string, index out of range, ...)
-1016	PI_AXIS_ALREADY_EXISTS	Axis identifier is already in use by a connected stage
-1017	PI_INVALID_AXIS_IDENTIFIER	Invalid axis identifier
-1018	PI_COM_ARRAY_ERROR	Could not access array data in COM server
-1019	PI_COM_ARRAY_RANGE_ERROR	Range of array does not fit the number of parameters
-1020	PI_INVALID_SPA_CMD_ID	Invalid parameter ID given to SPA or SPA?
-1021	PI_NR_AVG_OUT_OF_RANGE	Number for AVG out of range--must be >0
-1022	PI_WAV_SAMPLES_OUT_OF_RANGE	Incorrect number of samples given to WAV
-1023	PI_WAV_FAILED	Generation of wave failed
-1024	PI_MOTION_ERROR	Motion error while axis in motion, call CLR to resume operation
-1025	PI_RUNNING_MACRO	Controller is (already) running a macro



-1026	PI_PZT_CONFIG_FAILED	Configuration of PZT stage or amplifier failed
-1027	PI_PZT_CONFIG_INVALID_PARAMS	Current settings are not valid for desired configuration
-1028	PI_UNKNOWN_CHANNEL_IDENTIFIER	Unknown channel identifier
-1029	PI_WAVE_PARAM_FILE_ERROR	Error while reading/writing wave generator parameter file
-1030	PI_UNKNOWN_WAVE_SET	Could not find description of wave form. Maybe WG.INI is missing?
-1031	PI_WAVE_EDITOR_FUNC_NOT_LOADED	The WGWaveEditor DLL function was not found at startup
-1032	PI_USER_CANCELLED	The user cancelled a dialog
-1033	PI_C844_ERROR	Error from C-844 Controller
-1034	PI_DLL_NOT_LOADED	DLL necessary to call function not loaded, or function not found in DLL
-1035	PI_PARAMETER_FILE_PROTECTED	The open parameter file is protected and cannot be edited
-1036	PI_NO_PARAMETER_FILE_OPENED	There is no parameter file open
-1037	PI_STAGE_DOES_NOT_EXIST	Selected stage does not exist
-1038	PI_PARAMETER_FILE_ALREADY_OPENED	There is already a parameter file open. Close it before opening a new file
-1039	PI_PARAMETER_FILE_OPEN_ERROR	Could not open parameter file
-1040	PI_INVALID_CONTROLLER_VERSION	The version of the connected controller is invalid
-1041	PI_PARAM_SET_ERROR	Parameter could not be set with SPA--parameter not defined for this controller!
-1042	PI_NUMBER_OF_POSSIBLE_WAVES_EXCEEDED	The maximum number of wave definitions has been exceeded
-1043	PI_NUMBER_OF_POSSIBLE_GENERATORS_EXCEEDED	The maximum number of wave generators has been exceeded
-1044	PI_NO_WAVE_FOR_AXIS_DEFINED	No wave defined for specified axis



-1045	PI_CANT_STOP_OR_START_WAV	Wave output to axis already stopped/started
-1046	PI_REFERENCE_ERROR	Not all axes could be referenced
-1047	PI_REQUIRED_WAVE_NOT_FOUND	Could not find parameter set required by frequency relation
-1048	PI_INVALID_SPP_CMD_ID	Command ID given to SPP or SPP? is not valid
-1049	PI_STAGE_NAME_ISNT_UNIQUE	A stage name given to CST is not unique
-1050	PI_FILE_TRANSFER_BEGIN_MISSING	A uuencoded file transferred did not start with "begin" followed by the proper filename
-1051	PI_FILE_TRANSFER_ERROR_TEMP_FILE	Could not create/read file on host PC
-1052	PI_FILE_TRANSFER_CRC_ERROR	Checksum error when transferring a file to/from the controller
-1053	PI_COULDNT_FIND_PISTAGES_DAT	The PiStages2.dat database could not be found. This file is required to connect a stage with the CST command
-1054	PI_NO_WAVE_RUNNING	No wave being output to specified axis
-1055	PI_INVALID_PASSWORD	Invalid password
-1056	PI_OPM_COM_ERROR	Error during communication with OPM (Optical Power Meter), maybe no OPM connected
-1057	PI_WAVE_EDITOR_WRONG_PARAMNUM	WaveEditor: Error during wave creation, incorrect number of parameters
-1058	PI_WAVE_EDITOR_FREQUENCY_OUT_OF_RANGE	WaveEditor: Frequency out of range
-1059	PI_WAVE_EDITOR_WRONG_IP_VALUE	WaveEditor: Error during wave creation, incorrect index for integer parameter
-1060	PI_WAVE_EDITOR_WRONG_DP_VALUE	WaveEditor: Error during wave creation, incorrect index for floating point parameter
-1061	PI_WAVE_EDITOR_WRONG_ITEM_VALUE	WaveEditor: Error during wave creation, could not calculate value



-1062	PI_WAVE_EDITOR_MISSING_GRAPH_COMPONENT	WaveEditor: Graph display component not installed
-1063	PI_EXT_PROFILE_UNALLOWED_CMD	User Profile Mode: Command is not allowed, check for required preparatory commands
-1064	PI_EXT_PROFILE_EXPECTING_MOTION_ERROR	User Profile Mode: First target position in User Profile is too far from current position
-1065	PI_EXT_PROFILE_ACTIVE	Controller is (already) in User Profile Mode
-1066	PI_EXT_PROFILE_INDEX_OUT_OF_RANGE	User Profile Mode: Block or Data Set index out of allowed range
-1067	PI_PROFILE_GENERATOR_NO_PROFILE	ProfileGenerator: No profile has been created yet
-1068	PI_PROFILE_GENERATOR_OUT_OF_LIMITS	ProfileGenerator: Generated profile exceeds limits of one or both axes
-1069	PI_PROFILE_GENERATOR_UNKNOWN_PARAMETER	ProfileGenerator: Unknown parameter ID in Set/Get Parameter command
-1070	PI_PROFILE_GENERATOR_PAR_OUT_OF_RANGE	ProfileGenerator: Parameter out of allowed range
-1071	PI_EXT_PROFILE_OUT_OF_MEMORY	User Profile Mode: Out of memory
-1072	PI_EXT_PROFILE_WRONG_CLUSTER	User Profile Mode: Cluster is not assigned to this axis
-1073	PI_UNKNOWN_CLUSTER_IDENTIFIER	Unknown cluster identifier
-1074	PI_INVALID_DEVICE_DRIVER_VERSION	The installed device driver doesn't match the required version. Please see the documentation to determine the required device driver version.
-1075	PI_INVALID_LIBRARY_VERSION	The library used doesn't match the required version. Please see the documentation to determine the required library version.
-1076	PI_INTERFACE_LOCKED	The interface is currently locked by another function. Please try again later.
-1077	PI_PARAM_DAT_FILE_INVALID_VERSION	Version of parameter DAT file does not match the required version. Current files are available at www.pi.ws .

-1078	PI_CANNOT_WRITE_TO_PARAM_DAT_FILE	Cannot write to parameter DAT file to store user defined stage type.
-1079	PI_CANNOT_CREATE_PARAM_DAT_FILE	Cannot create parameter DAT file to store user defined stage type.
-1080	PI_PARAM_DAT_FILE_INVALID_REVISION	Parameter DAT file does not have correct revision.
-1081	PI_USERSTAGES_DAT_FILE_INVALID_REVISION	User stages DAT file does not have correct revision.

