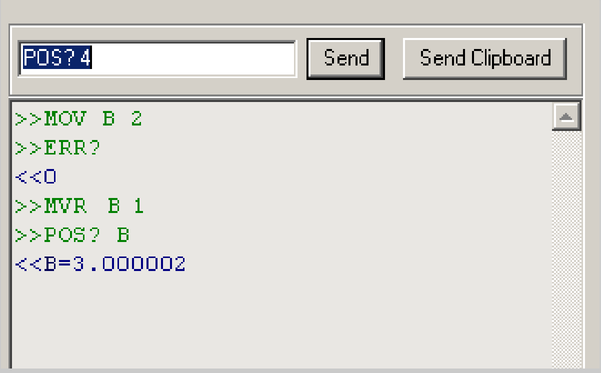
Piezo · Nano · Positioning

MS163E Software Manual

Mercury™ GCS Commands

PI General Command Set

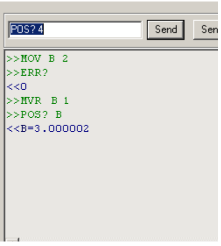
Release: 1.0.2 Date: 2008-05-09



```
>>MOV B 2
>>ERR?
<<0
>>MVR B 1
>>POS? B
<<B=3.000002
```

This document describes software for use with the following products:

- C-663
  Mercury™ Step Networkable Single-Axis Stepper Motor Controller
- C-862
  Mercury™ Networkable Single-Axis DC-Motor Controller
- C-863
  Mercury™ Networkable Single-Axis DC-Motor Controller



```
>>MOV B 2
>>ERR?
<<0
>>MVR B 1
>>POS? B
<<B=3.000002
```

Moving the NanoWorld | www.pi.ws

© Physik Instrumente (PI) GmbH & Co. KG
Auf der Römerstr. 1 · 76228 Karlsruhe, Germany
Tel. +49 721 4846-0 · Fax: +49 721 4846-299
info@pi.ws · www.pi.ws

Document Number MS163E, Release 1.0.2
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 This manual is superseded by any new release.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	User Manual for each hardware component
Mercury™ GCSLabVIEW_MS149E	LabView VIs based on PI GCS command set
Mercury™ GCS DLL_MS154E	Windows DLL Library (GCS commands)
PIMikroMove User Manual SM148E	PIMikroMove® Operating Software (GCS-based)
Mercury™ Commands MS163E	Mercury™ GCS Command descriptions
PIStageEditor _SM144E	Software for managing GCS stage-data database
Mercury™ Native Commands MS176E	Native Mercury™ Commands
MMCRun MS139E	Mercury™ Operating Software (native commands)
Mercury™ Native DLL & LabVIEW MS177E	Windows DLL Library and LabView VIs (native-command-based)

Contents

1 Introduction 3

1.1 Native Command Set 3

1.2 GCS Command Set 3

2 Units and GCS 5

2.1 Hardware, Physical Units and Scaling 5

2.2 Rounding Considerations 5

3 Referencing 6

3.1 Reference Mode 6

3.2 Perform a Reference Move 6

3.3 Set Absolute Position 6

4 Macro Storage on Controller 7

4.1 GCS Macros 7

4.1.1 Features and Restrictions 7

4.1.2 Macro Creation in GCS 7

4.1.3 Listing Stored Macros 10

4.1.4 Macro Translation & Listing Examples 11

5 GCS Command Syntax 13

5.1 Command Format 13

5.2 Identifiers 14

5.2.1 Axes 14

5.2.2 Digital Input/Output 14

5.2.3 Analog Input 15

5.2.4 Joystick Connections 15

6 Command Descriptions 16

6.1 Command List (Alphabetical) 16

6.2 Command Reference (Alphabetical) 17

7 Stage Parameters 44

7.1 Servo-Loop Parameters 44

7.2 Transmission Ratio & Scaling Factors 45

7.3 User-Changeable Parameters at a Glance 45

8 Error Codes 48

1 Introduction

Mercury™ Class controllers include the C-663 Mercury™ Step open-loop, stepper motor controller as well as the C-862 and C-863 Mercury™ DC-motor servo-controllers.

With current firmware, it is possible to operate Mercury™ controllers with two command sets: the native ASCII command set and the PI General Command Set (GCS)* . GCS support is currently provided via a Windows DLL which translates GCS-command-based function calls to the native commands. Either command set can be used to set operating modes, transfer motion parameters and to query system and motion values.

1.1 Native Command Set

The native ASCII command set is understood by the Mercury™ firmware. It can be used with virtually any terminal emulator software and with *MMCRun* on the CD that comes with the controller.

Most native Mercury™ commands begin with a two-letter mnemonic. Because the native networking architecture uses an address selection mechanism, the commands themselves do not include controller or axis designators. The syntax of the native commands and a command reference can be found in the Native Commands Manual, MS176. This manual covers only the GCS command set.

1.2 GCS Command Set

The GCS is the PI standard command set. This command set ensures the compatibility between different controllers. It provides comprehensive access to Mercury™ Class controller functionality.

The GCS command set views networked Mercury™ Class controllers as a single multi-axis controller. Most GCS commands begin with a three-letter mnemonic. Because they address the network as a whole, the commands contain unique identifiers for individual axes (controllers) and for the I/O channels on the controllers (see Identifiers p. 14 for details).

* With current Mercury™ firmware, GCS support is provided via a Windows DLL which translates GCS-command-based function calls to the native commands (for details see the Mercury™ GCS DLL manual). *PIMikroMove®* converts the GCS ASCII commands described here to the corresponding function calls. Check www.pi.ws for availability of the planned Mercury™ GCS firmware, and the operating software manual for the firmware update procedure.

NOTE

Do not mix the GCS and the native commands! GCS move commands, for example, do not work properly after the position has been changed by a native command.

You can type GCS commands in the *Command entry* window of *PIMikroMove*® (see the *PIMikroMove*® manual for details) or using the PITerminal program in GCS DLL mode.

2 Units and GCS

2.1 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 14 and 15) are found in the PIstages.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 7.2 on p. 45.

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

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

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

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. 34) 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. 34).

3.2 Perform a Reference Move

When the reference mode is set to "1" (value in PIStages.DAT, usually “1”), referencing is done by performing a reference move with REF (p. 33), MPL (p. 30), or MNL (p. 28).

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. 33), 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. 32) or by a referencing move.

4 Macro Storage on Controller

4.1 GCS Macros

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.

4.1.1 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.

4.1.2 Macro Creation in GCS

he 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) 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> = <b>
    - MEX JBS? <joystk> 1 = <b>
    - MVR
    - REF (generates a large number of native commands in the macro, see below)

[*] 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),

- SPA
  Access to the following SPA parameters by macros is permitted: all others will be ignored:
  1: P-Term
  2: I-Term
  3: D-Term
  4: I-Limit
  8: Max.Position Error
  10: Max. Velocity
  11: Max Acceleration (muss >200 sein)
  24: Limit Switch Mode
  50: No Limit Switch
  64: Stepper motor hold current (HC native command) in mA
  65: Stepper motor drive current (DC native command) in mA
  66: Stepper motor hold time (HT native command) in ms
- STP
- SVO
- VEL
- WAC ONT? <axis> = 1
- WAC DIO? <ch> = <b>

4.1.3 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>”

4.1.4 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.

5 GCS Command Syntax

5.1 Command Format

GCS ASCII Commands have the format below. Exceptions are the single-character binary commands on p. 42 *ff*.

CMDSPXSPsV.V[{SPXSPsV.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. 14),
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: MOVSPA10.0SPB5.0
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.VLF
where:
X item identifier (see p. 14)
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? SPABLF
Report: A=10.0000SP LF
B=5.0000LF
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.

5.2 Identifiers

5.2.1 Axes

If multiple Mercury™ controllers are connected together in a network, a unique axis identifier is assigned to each controller by the PI_Mercury™_GCS_DLL. The defaults depend on the controller addresses. The address of a controller (0 to 15) is set in DIP switches on the front panel and is one less than the device number (1-16). The corresponding default axis identifiers are A, B, C, D, etc., starting with address 0, device 1. Letters for missing addresses are skipped.

The default identifiers can be changed using SAI (p. 34). 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.

5.2.2 Digital Input/Output

Each controller provides four digital output channels and four channels that can be read as either digital or analog inputs (C-862 has only 3 analog inputs). The digital I/O commands (DIO, DIO?) identify these channels with the single-character identifiers as follows: “A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 1 2 3 4 5 6 7 8 9 0 @ ? > = < ; : ` _ ^] \ [/ . - , + *) (' & % $ # " !” (four for each controller) with addresses 0 through 15. Identifiers associated with missing addresses are skipped.

5.2.3 Analog Input

The same input lines can also be read as analog inputs of 0 to 5 V. The analog input command TAV? identifies the input lines with A1 to A64, again depending on the controller’s address setting, and skipping values associated with any missing addresses. The fourth line on C-862 DC motor controllers is digital only and cannot be read in analog mode.

5.2.4 Joystick Connections

Each axis associated with a controller having a joystick port, can be associated with one axis of motion of a joystick. That axis, and the associated joystick button, is identified in the network by the controller device number, which is one greater than the controller address. Note that the included joystick Y-cable permits connecting one axis and one button of one joystick to one controller and the other axis and other button to another controller.

6 Command Descriptions

6.1 Command List (Alphabetical)

*IDN? (Get Identity), p. 17
BRA (Set brake on or off), p. 17
BRA? (Ask if axis has brakes), p. 18
CST (Change Stage), p. 18
CST? (get stagename), p. 19
DEL (Delay), p. 19
DFF (DeFine Factor), p. 19
DFF? (get factor), p. 20
DFH (DeFine Home), p. 20
DFH? (get home positions), p. 20
DIO (set Digital Output), p. 21
DIO? (get Digital Output), p. 21
ERR? (get ERRor), p. 21
GOH (GO Home), p. 22
HLP? (HeLP), p. 23
HLT (HaLT), p. 23
INI (INItialization), p. 23
JAX? (List joystick to motion-axis assignments)
JON (activate/deactivate joystick)
JON?(get joystick enable status)
LIM? (Indicates whether axes have limit switches), p. 25
MAC (macro), p. 25
MAC? (list macro), p. 27
MEX (Stop macro execution if specified condition true), p. 27
MNL (Move to Negative Limit), p. 28
MOV (MOVe absolute), p. 29
MOV? (read target position), p. 30
MPL (Move to Positive Limit), p. 30
MVR (MoVe relatiVe), p. 31
ONT? (axis ON Target), p. 31
POS (set real position), p. 32
POS? (read real POSition), p. 32
REF (move to REFerence position), p. 33
REF? (Lists axes which have a reference sensor), p. 33
RON (set Reference mode ON | off), p. 34
RON? (get Reference mode), p. 34
SAI (Set Axis Identifier), p. 34
SAI? (get axis identifier), p. 35
SPA (Set Parameter), p. 35
SPA? (Get Parameter), p. 36
SRG? (Read register), p. 36
STP (Stop Motion), p. 36
SVO (set SerVO on or off), p. 37
SVO? (Get servo status), p. 37

TAC? (Tell Analog Channels), p. 38
TAV? (Get Analog Input), p. 38
TIO? (Tell Digital I/Os), p. 38
TMN? (Tell Minimum Travel Value), p. 39
TMX? (Tell Maximum Travel Value), p. 39
TVI? (Tell Valid axis Identifiers) p. 39
VEL (Set Velocity), p. 40
VEL? (Get Velocity), p. 40
VER? (Get Version), p. 41
VST? (Get available Stages), p. 41
WAC (Wait for condition), p. 41
#5, (Poll Motion Status), p. 42
#7, (Controller Ready?), p. 42
#8, (Macro running?), p. 43
#24, (Stop), p. 43

6.2 Command Reference (Alphabetical)

*IDN? (Get Identity Number)

Description:	Reports an identity string
Format:	*IDN?
Arguments:	none
Response:	One-line string terminated by line feed, e.g.: Physik Instrumente,Mercury™ GCS Network,,0.9.3.6

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.: 13
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. 41). 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:	GOH
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 **ST**age for queried axes. CST? will always return all axes, i.e. the answer also includes the axes set to NOSTAGE (see CST, p. 18). In contrast to this, SAI? (p. 35) 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".

DEL (DELay)

Description:	**DEL**ays <uint> milliseconds. During delay controller does not answer on any queries. DEL is used within macros primarily. Do not mistake MAC DEL which deletes macros for DEL which delays. This command can be interrupted with #24.
Format:	DEL <uint>
Arguments:	<uint> is the delay value in milliseconds.
Response:	none

DFF (DeFine Factor)

Description:	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. See Section 7.2 on p. 45 and Section 2.1 on p. 5 for more information.
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. 33), MNL (p. 28) and MPL (p. 30).

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. 38) to get the number of installed digital I/O lines. If the controllers on the network have addresses in order beginning with 0, then the output line designators will begin with A, B, C, D,..., four for each controller (see Identifiers p. 14 for more details).
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?. If the controllers on the network have addresses in order beginning with 0, then the designators to be used when reading the inputs digitally will begin with A, B, C, D,..., four for each controller (see Identifiers p. 14 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 the Mercury™ GCS DLL Manual MS 154E.

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: HaLT the motion of given axes smoothly. Only commands causing non-complex motion (e.g. MOV, GOH) can be interrupted with HLT.

Error code 10 is set.

Use #24 instead to stop complex motions like referencing, etc.

HLT stops motion with given system deceleration with regard to system inertia.

STP (p. 36) in contrast aborts current motion as fast as possible for the controller without taking care of systems inertia or oscillations.

After the axis was stopped, the target position is set to the current position.

Format: HLT [{ <AxisID>}]

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

Response: none

Troubleshooting: Illegal axis identifier

INI (INItialization)

Description: Initializes the axis: sets reference state to "not referenced," 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

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: {<DeviceNumber> 1= <axisID>}
where
<DeviceNumber> is one greater than the device address of the connected Mercury™ Class motion-axis controller
<axisID> is the ID of the associated motion axis.
"1" indicates that each device can connect to only 1 joystick axis.
Troubleshooting:

JDT (load Joystick response Table)

Description: Load pre-defined joystick response table. Table types are either linear or cubic response curve.
The cubic curve offers more sensitive control around the middle position and less sensitivity close to the maximum velocity.
Format: JDT [{ <JoystickAxisNumber> <TableType>}]
Arguments: <JoystickAxisNumber>: is the device number of the individual Mercury™ Class device to which the joystick axis is directly connected
<TableType>: 0 for linear, 1 for cubic
Response: none
Troubleshooting: Illegal joystick axis number, unsupported table type

JON (Activate/deactivate joystick control)

Description:	Enable/disable given joystick axes. Do not enable axes 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
Format:	JON [{ <JoystickAxisNumber> <State>}]
Arguments:	<JoystickAxisNumber>: is the device number of the individual Mercury™ Class device to which the joystick axis is directly connected <Mode>: 0 for disable, 1 for enable
Response:	none
Troubleshooting:	Illegal joystick axis number, unsupported mode

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. 7 for details).
Format:	MAC <keyword> {<parameter>} in particular: MAC BEG <macroname> MAC DEL <macroname> MAC END MAC NSTART <macroname> <uint> MAC START <macroname>
Arguments:	<keyword> determines which macro function

is called. The following keywords and parameters are used:

MAC BEG <macroname>

Start recording a macro on the controller to be named *macroname*, which must be of the form *a*MC0*nn* where *a* is the axis designation of the axis controlled by the controller on which the macro is to be stored and *nn* 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.

MAC END

Stop macro recording (cannot become part of a macro); any error during macro recording can be seen with ERR? after MAC END

MAC DEL <macroname>

Deletes specified macro

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.

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.

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. 41.

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 first 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: 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. 6 for more information

regarding referencing).
If <AxisID> is omitted, moves all axes. If multiple axes are affected by MNL, one axis after another is moved to its limit switch.

This command can be interrupted by #24.

Note that MNL resets the home position set with DFH, p. 20.

Format: MNL [{ <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

MOV (MOVe absolute)

Description: Set new absolute target position for given axis. Axes will start **MOV**ing to the new positions only if ALL given targets are within the allowed ranges and ALL given axes can move.

All given axes start moving simultaneously.

This command can be interrupted by #24, STP and HLT.

Servo-control must be enabled for all commanded axes prior to using this command.

See also Section "Units and GCS" (on p. 5).
Format: MOV <AxisID> <float>[{ <AxisID> <float>}]
Arguments <AxisID> is a valid axis identifier

<float> is the target position in physical units.
Response: none
Troubleshooting: Parameter out of limits, Illegal axis identifier, joystick 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. 32) to get the current positions. See also Section "Units and GCS" (on p. 5).
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. 6 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. 20.
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. 34)

MVR (MoVe Relative)

Description: **M**o**V**e given axes **R**elative to their current position.

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.

This command can be interrupted by #24, HLT and STP.

Servo must be enabled prior to using this command.
See also Section "Units and GCS" (on p. 5).

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

Arguments: <AxisID> is a valid axis identifier.

<float> added to the last commanded target position is set as new target position in physical units.

Response: none

Example:

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

Send:	MVR A 1 B 2
Note:	This is a relative motion.
Send:	POS? A B

Receive:	A=0.500000
	B=14.300000

Send:	MVR A 1 B 2000
Note:	Target position of axis B exceeds travel range. Command is not executed
Send:	POS? A B

Receive:	A=0.500000
	B=14.300000

ONT? (axis ON Target)

Description: Get **ON-T**arget 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. 34).

An axis is considered as "referenced" when the position has been set with POS (see Section 3 on p. 6 for more information).

The minimum and maximum commandable positions (TMN?, p. 39, TMX?, p. 39) 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: Returns the current **POS**ition.

If <AxisID> is omitted, all axes are queried.

Response depends on the factor set by DFF (p. 19).
See also Section "Units and GCS" (on p. 5).

Format: POS? [{ <AxisID>}]

Arguments: <AxisID> is a valid axis identifier.
Response: {<axis>"="<float> LF}
where
<float> is the current position in physical units
Troubleshooting: Illegal axis identifier

REF (move to REFerence position)

Description: 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. 6 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. If <AxisID> is omitted, moves all axes. If multiple axes are affected by REF, one axis after another is moved to its switch.

The REF command always approaches the reference switch from the same side, no matter where the axis is when it is issued. This command can be interrupted by #24.

Format: REF [{ <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. 34)

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

Troubleshooting: reference switch (=1) or not (=0).
Illegal axis identifier

RON (set reference mode)

Description: Set reference mode of given axes.

Mode = 0: referencing moves with REF (p. 33), MNL (p. 28) and MPL (p. 30) are not possible, absolute position must be set with POS (p. 32) 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. 6 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: **S**et old **A**xis **I**dentifier to new identifier. TVI? (p. 39) 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. 18). In contrast to this, SAI? ALL and CST? (p. 19) 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: **S**et a given **PA**rameter of the given axis to given value in volatile memory.
Parameter changes will be lost when the controller is powered off or rebooted. See the *PI Stage Editor* or *PIMikroMove*® for ways to save parameter sets as user stages.
CAUTION! The SPA command is for setting hardware-specific parameters. Incorrect values may lead to improper operation or damage of your hardware!

Format: SPA <AxisID> <ParamNumber> <ParamValue>[{ <AxisID> <ParamNumber> <ParamValue>}]

Arguments <AxisID>: is a valid axis identifier
<ParamNumber> is the parameter ID. Valid parameter IDs are given in the list on p. 45.

<ParamValue> is the value to which the parameter <ParamNumber> of <AxisID> is to be set.

Response: none

Troubleshooting: Illegal axis identifier, incorrect parameter ID

Example: Send: SPA A 10 0.05 B 10 0.08

Note: Set the maximum velocity for axis A to 0.05 mm/s and for axis B to 0.08 mm/s

SPA? (Get Parameter)

Description: Get the value of specified parameters of specified axes

Format: SPA? [{ <AxisID> <ParamNumber>}]

Arguments: <AxisID>: is a valid axis identifier
<ParamNumber> is the parameter ID. Valid parameter IDs are given in the list on p. 45.

Response: {<AxisID>
<ParamNumber>"="<ParamValue> LF}
where
<ParamValue> is the value to which the parameter <ParamNumber> of <AxisID> is set.

Troubleshooting: Illegal axis identifier, incorrect parameter ID

Example:	Send:	SPA? A 10 A 11 A 12
	Receive:	A10=0.1
		A11=10
		A12=10

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 4 for the C-862)

STP (Stop Motion)

Description: **ST**o**P**s 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. 23) instead to stop individual axes smoothly.
Format:	STP
Arguments:	none
Troubleshooting:	Communication breakdown

SVO (set SerVO on or off)

Description: Sets **SerVO**-control mode on or off for given axes.

When servo-control is off for an axis:

- All positioning commands (e.g. MOV, MVR) are ignored.
- With C-86x 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.

When servo is switched off while stage is moving, the stage stops.

Format:	SVO <AxisID> <status>[{ <AxisID> <status>}]
Arguments:	<AxisID>: is a valid axis identifier <status>= 0 set servo off, 1 set servo on
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.
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 Identifiers p. 14 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:	Get the minimum commandable position in physical units The minimum commandable position is defined by the travel range limit of the connected stage type. When the physical unit of an axis is scaled with DFF (p. 19), or the zero-point shifted with DFH, the minimum commandable position is automatically adjusted to the appropriate new value.
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:	Get the maximum commandable position in physical units. The maximum commandable position is defined by the travel range limit of the connected stage type. When the physical unit of an axis is scaled with DFF (p. 19), or the zero-point shifted with DFH, the maximum commandable position is automatically adjusted to the appropriate new value.
Format:	TMX? [{ <AxisID>}]
Arguments:	<AxisID>: is a valid axis identifier
Response	{<AxisID>"="<float> LF} where <float> is the maximum commandable position in physical units

TVI? (Tell Valid axis Identifiers)

Description:	**T**ell **V**alid set of characters which can be used as axis **I**dentifiers. Note: Use SAI (p. 34) to set axis identifiers and SAI? ALL (p. 35) to get the axis identifiers which are currently used.

Format: TVI?

Arguments: none

Response: <string> is a list of characters

Troubleshooting:

VEL (Set Velocity)

Description: Set **VEL**ocity to use for moves of specified axes.

Notes:

The maximum velocity of an axis is given by SPA parameter 10 (see SPA, p. 35, and parameter list on p. 45). VEL does not change this maximum but sets only the currently applied velocity (which must be lower than the maximum velocity).

When the physical unit of an axis is scaled with DFF (p. 19), 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 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.

Format: VER?
Arguments: none
Response: <string> is the version information of the firmware, e.g.
PI_Mercury™_GCS_DLL: 1.0.0.11
A:(c)2006 Physik Instrumente(PI) Karlsruhe, C-663, Ver. 1.06, 2006-08-04
D:(C)2000-4 DIVA Automation/PI GmbH Karlsruhe, Ver. 8.40, 13 Jan, 2004

VST? (Get available Stages)

Description: List the names of all stages which can be connected to the controller (with CST, p. 18).
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. 27.
Format: WAC <CMD?> <OP> <value>
Arguments: <CMD?> is a query command in its usual syntax. The answer must consist of a single value. Supported commands are ONT? and 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 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. 19), 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: Stops all motion abruptly.

This includes motion of all axes (MOV, MVR) and referencing motion (MNL, MPL, REF).

Sets error code to 10.

After all the axes are stopped, their target positions are set to the current positions.

This command is identical in function to STP (p. 36), but only one character must be sent via the interface. Therefore #24 can also be used while the controller is performing time-consuming tasks.

#24 always does a hard stop. Use HLT (p. 23) to stop a single axis or to stop axes smoothly.

Format: #24 (ASCII character 24)

Arguments: none

Response: none

7 Stage Parameters

!

CAUTION

Changing stage parameters may damage your stage!

Most of the parameters (which are loaded from the PiStages.dat or *Controller*UserStages.dat database) describe the physical properties and limits of a stage. They can be changed by several commands, but modifying them imprudently could cause damage to the stage. So be sure to handle these parameters with care and change them only if you want to connect a stage which is not found in the PiStages.dat or *Controller*UserStages.dat database (you get all stages from these DAT files using VST?, p. 41), or in very special cases.
The relevant parameters are listed in the following subsections.

7.1 Servo-Loop Parameters

NOTE

Servo-loop parameters are not relevant for stepper motor controllers because there is no servo-control algorithm implemented in the current C-663 Mercury™ Step controller firmware.

Name	Number*	Description	Range
E_n	-	Accumulated error terms	
K_p	1	P-Term	0 to 32767
K_i	2	I-Term	0 to 32767
K_d	3	D-Term	0 to 32767
-	4	I-Limit	0 to 32767
K_{vff}	5	Kvff (Velocity feed forward)	0 to 32767
K_{aff}	59	Kaff (Acceleration feed forward)	0 to 32767
K_{out}	6	Kout (output scale factor)	0 to 65536
Bias	7	Bias (motor bias)	0 to 32767

See **SPA** (p. 35) and **SPA?** (p. 36).

*Number refers to the parameter ID used with SPA.

7.2 Transmission Ratio & Scaling Factors

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

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

Name	Number *	Description	Range
PU	-	Physical Unit	-
Cnt	-	Counts (with C-86x) or steps (with C-663)	-
CpuN	14	Numerator of the counts / steps per physical unit factor	1 to 2147483647
CpuD	15	Denominator of the counts / steps per physical unit factor	1 to 2147483647
SF	18	Scale factor	>0 and ≤ 1.79769313486231E308

See also **DFF** (p. 19).

*Number means the parameter ID.

7.3 User-Changeable Parameters at a Glance

Parameter numbers in italics apply to C-663, those in bold to C-862 (and, of course, those in bold italics to both)

1	P-Term	0 to 32767	-	
2	I-Term	0 to 32767	-	
3	D-Term	0 to 32767	-	
4	I-Limit	0 to 32767	-	
8	Maximum position error	0 to 32767	Counts	
10	Maximum allowed velocity	> 0	Physical units per second	This parameter is a maximum limit and not the current velocity. By default the current velocity is half the maximum allowed velocity. To change the current velocity use the VEL() command.

11	Maximum allowed acceleration		Physical units per second squared	
14	Numerator of the counts per physical unit factor	1 to 2147483647	-	factor = num./denom. This factor includes the physical transmission ratio and the resolution of the stage. Note: To customize your physical unit use parameter 18 instead.
15	Denominator of the counts per physical unit factor	1 to 2147483647	-	factor = num./denom. This factor includes the physical transmission ratio and the resolution of the stage. Note: To customize your physical unit use parameter 18 instead.
17	Invert the direction	-1 to invert the direction, else 1	-	
18	Scaling factor	-	-	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 the **DFF**() command to change this factor.
19	Rotary stage	1 = rotary stage, else 0	-	
20	Stage has a reference	1 = the stage has a reference, else 0	-	
21	Maximum travel range in positive direction	0 to 2147483647	Physical units	
22	Value at reference position	-2147483647 to 2147483647	Physical units	
23	Distance from the negative limit to the reference position	-2147483647 to 2147483647	Physiccal units	
24	Axis limit mode (must agree with hardware interlock setting, see HW User Manual)	0, 1, 2, 3 -	Code	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
25	Stage type	0 = DC motor 1 = Piezo 2 = Voice coil 4 = Piezomotor 6 = Stepper		Incorrect stage type may cause malfunction.

48	Maximum travel range in negative direction	-2147483647 to 2147483647	Physiccal unit	
49	Invert the reference	1 = invert the reference, else 0	-	
60	Stage name	maximum 15 characters	-	
64	Hold Current (HC native command)		mA	
65	Drive Current (DC native command)		mA	
66	Hold Time (HT native command)		ms	
67	max current, max. value that DC and HC can have,		mA	

* ”Number” refers to the parameter ID used by Mercury_SPA().

8 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 wav 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	GCS-array doesn't support different length; request arrays which have different length 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	MOV, MVR, SVA, SVR, STE, IMP and WGO blocked when analog target is active
73	PI_CNTR_WAVE_GENERATOR_ACTIVE	MOV, MVR, SVA, SVR, STE and IMP blocked when wave generator is active
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, call 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	Position can be estimated only while MOV! is running
219	PI_CNTR_POSITION_BASED_ON_INTERPOLATION	Position was calculated while MOV is running
301	PI_CNTR_SEND_BUFFER_OVERFLOW	Send buffer overflow
302	PI_CNTR_VOLTAGE_OUT_OF_LIMITS	Voltage out of limits
303	PI_CNTR_VOLTAGE_SET_WHEN_SERVO_ON	Attempt to set voltage 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
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
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

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 non-blocked signal

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 PiStages.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