



PZ120E Software Manual E-816 GCS DLL & COM Server Windows Library Reference

Release: 1.1.0 Date: 2007-06-22



This document describes software for use with the following product(s):

E-816

Computer Interface and Command Interpreter Submodule (firmware version 2.11) for PZT Controllers



© 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

Physik Instrumente (PI) GmbH & Co. KG is the owner of the following company names and trademarks: Pl® Hyperbit™

Hyperbit[™] is protected by the following patents: US Patent 6,950,050

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

Copyright by 1999–2007 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.

First printing 2007-06-22 Document Number PZ120E, ECo, Release 1.1.0 E-816_DLL_PZ120E.doc

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 This manual is superseded by any new release.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

Disclaimer

This software is provided "as is". PI does not guarantee that this software is free of errors and will not be responsible for any damage arising from the use of this software. The user agrees to use this software on his own responsibility.

Table of Contents

1.	Introduction to E-816 DLL	3
1.1.	Threads	3
1.2.	Overview	3
1.3.	Master Unit	3
2.	DLL Handling	5
2.1.	Using a Static Import Library	5
2.2.	Using a Module Definition File	5
2.3.	Using Windows API Functions	3
3.	Function Calls	7
3.1.	Controller ID	7
3.2.	Axes Parameter	7
3.3.	Special Axis Identifier A	7
4.	Types Used in PI Software	9
4.1.	Boolean Values	9
4.2.	NULL Pointers	9
4.3.	C-Strings	9
5.	Communication Initialization10)
5.1.	Functions10	C
5.2.	Function Documentation10	0
6.	Interface Settings13	3
7.	E-816 Commands14	4
7.1.	Functions14	4
7.2.	Function Documentation14	4
8.	GCS COM Server2	5
8.1.	No Need for Controller IDs25	5
8.2.	No Need for Buffer Sizes	3
9.	Error Codes27	7
10.	Index 39	3

1. Introduction to E-816 DLL

This library allows controlling one or more PI E-816 Computer Interface and Command Interpreter Submodules connected to a host PC. The library insulates the user from the interface-specific interaction. Connection to each E-816 can be made with RS-232.

For programmers working in languages with better COM (Component Object Model) support, an E-816 COM Object is also provided. See "GCS COM Server" in Section 8 for a brief explanation of the syntax differences. The descriptions in the rest of this manual will use the DLL syntax.

1.1. Threads

This DLL is not "thread-safe". The function calls of the DLL are not synchronized and can be safely used by only one thread at a time.

1.2. Overview

- DLL Handling (p.5) explains how to load the library and how to access the functions provided by the E-816 DLL.
- Function Calls (p.7) and Types Used in PI Software (p.9) provides some general information about the syntax of most commands of the DLL.
- Communication Initialization (*p.10*) shows how to initiate communication with an E-816 controller (see also Interface Settings (p.13)).
- E-816 Commands (p. 14) describes the functions encapsulating the embedded commands of the E-816.
- > **Error Codes** (p.27) has a description of the possible errors.

1.3. Master Unit

Several E-816s can be connected together in a network using the I²C bus lines provided. One device of the network is connected to the host PC with an RS-232 cable. This E-816 is the master and all other E-816s in the network are slaves. With the commands of the E-816 DLL you can control all networked E-816s via the single master. The network behaves like a single multi-axis controller. The DLL does not care which E-816 in the network is the master, and the RS-232 cable can be moved while the library is active (all E-816s must, however, be power-cycled if the master is changed).The term "E-816 network" as used in this manual should be understood to refer to a single (non-networked) E-816 as well.

Although the E-816 itself will not accept more than one axis identifier per command, most of the DLL functions will; they split multi-axis commands into single-axis commands before sending them over the interface. This means that when you call **one** DLL function addressing three axes, three commands will be sent and cannot be exactly synchronized.

There are some commands (almost all for configuration) which may only be addressed to the master. If you want to change the configuration of a slave unit, you must reconnect the RS-232 cable making it the master and power-cycle the E-816s. The master-only commands are:

- ► E816_qERR() (p.17)
- E816_qIDN() (p.17)
 E816_qI2C() (p.17)
- E816_BDR() (p. 15)
- E816_qBDR() (p. 16)
- ➢ E816_AVG() (p.14)
- E816_qAVG() (p. 16)
- E816_SCH() (p.21)
- E816_qSCH() (p.19)
- E816_SPA() (p.21)
- E816_qSPA() (p. 19)
- E816_WPA() (p.23)
- ► E816_RST() (p.21)

Some of the commands require an axis identifier for compatibility reasons; you must use an axis identifier that will be recognized as the master. The master has whatever axis identifier has been assigned to it and, in addition, always executes commands addressed to the special axis identifier "A". If you do not know which axis identifier the master has been assigned, simply use "A" (or call **E816_qSCH**() (*p.19*)).

2. DLL Handling

To get access to and use the DLL functions, the library must be included in your software project. There are a number of techniques supported by the Windows operating system and supplied by the different development systems. The following sections describe the methods which are most commonly used. For detailed information, consult the relevant documentation of the development environment being used. (It is possible to use the E816_DLL.DLL in Delphi projects. Please see http://www.drbob42.com/delphi/headconv.htm for a detailed description of the steps necessary.)

2.1. Using a Static Import Library

The E816_DLL.DLL module is accompanied by the E816_DLL.LIB file. This is the static import library which can be used by the Microsoft Visual C++ system for 32-bit applications. In addition, other systems, like the National Instruments LabWindows CVI or Watcom C++ can handle (i.e. understand) the binary format of a VC++ static library. When the static library is used, the programmer must:

- Use a header or source file in which the DLL functions are declared, as needed for the compiler. The declaration should take into account that these functions come from a "C-Language" Interface. When building a C++ program, the functions have to be declared with the attribute specifying that they are coming from a C environment. The VC++ compiler needs an extern "C" modifier. The declarations must also specify that these functions are to be called like standard Win-API functions. That means the VC++ compiler needs a WINAPI or __stdcall modifier in the declaration.
- Add the static import library to the program project. This is needed by the linker and tells it that the functions are located in a DLL and that they are to be linked dynamically during program startup.

2.2. Using a Module Definition File

The module definition file is a standard element/resource of a 16- or 32-bit Windows application. Most IDEs (integrated development environments) support the use of module definition files. Besides specification of the module type and other parameters like stack size, function imports from DLLs can be declared. In some cases the IDE supports static import libraries. If that is the case, the IDE might not support the ability to declare DLL-imported functions in the module definition file. When a module definition file is used, the programmer must:

- Use a header or source file where the DLL functions must be declared, which is needed for the compiler. The declaration should take into account that these functions come from a "C-Language" Interface. When building a C++ program, the functions have to be declared with the attribute indicating that they are coming from a C environment. The VC++ compiler needs an extern "C" modifier. The declarations must also specify that these functions are to be called like standard Win-API functions. Therefore, the VC++ compiler needs a WINAPI or __stdcall modifier in the declaration.
- Modify the module definition file with an IMPORTS section. In this section, all functions used in the program must be named. Follow the syntax of the IMPORTS statement. Example:

IMPORTS E816_DLL.E816_IsConnected

2.3. Using Windows API Functions

If the library is not to be loaded during program startup, it can sometimes be loaded during program execution using Windows API functions. The entry point for each desired function has to be obtained. The DLL linking/loading with API functions during program execution is always possible, independent of the development system or files which have to be added to the project. When the DLL is loaded dynamically during program execution, the programmer has to:

Use a header or source file in which local or global pointers of a type appropriate for pointing to a function entry point are defined. This type could be defined in a typedef expression. In the following example, the type FP_E816_IsConnected is defined as a pointer to a function which has an int as argument and returns a BOOL value. Afterwards a variable of that type is defined.

```
typedef BOOL (WINAPI *FP_E816_IsConnected)( int );
    FP_E816_IsConnected pE816_IsConnected;
```

Call the Win32-API LoadLibrary() function. The DLL must be loaded into the process address space of the application before access to the library functions is possible. This is why the LoadLibrary() function has to be called. The instance handle obtained has to be saved for use by the GetProcAddress() function. Example:

```
HINSTANCE hPI_Dll = LoadLibrary("E816_DLL.DLL\0");
```

Call the Win32-API GetProcAddress() function for each desired DLL function. To call a library function, the entry point in the loaded module must be known. This address can be assigned to the appropriate function pointer using the GetProcAddress() function. Afterwards the pointer can be used to call the function. Example:

```
pE816_IsConnected =
(FP_E816_IsConnected)GetProcAddress(hPI_Dll,"E816_IsConnected\0");
if (pE816_IsConnected == NULL)
{
    // do something, for example
    return FALSE;
}
BOOL bResult = (*pE816_IsConnected)(1); // call E816_IsConnected(1)
```

3. Function Calls

Almost all functions will return a boolean value of type BOOL (see "Types Used in PI Software," p.9. If the function succeeds the return value is TRUE, otherwise it is FALSE. To find out what went wrong, call **E816_GetError**() (*p.11*) and look up the value returned in "Error Codes," *p.27*.

3.1. Controller ID

The first argument to most function calls is the ID of the selected controller. To allow the handling of multiple controllers, the user will be returned a non-negative "ID" when he or she opens a connection to a controller (see "Communication Initialization" p.10) This is a kind of index to an internal array storing the information for the different controllers. All other calls addressing the same controller have this ID as first parameter.

3.2. Axes Parameter

The E-816 will only accept one axis per command sent over the interface. The DLL functions will accept more than one axis, splitting one function call into several single-axis commands. So although you call only one function, the resulting actions on the E-816s are not executed simultaneously.

The parameters for the axes are stored in an array passed to the function. The parameter for the first axis is stored in array[0], for the second axis in array[1], and so on. So if you call **E816_qPOS**(ID, "ABC", double pos[3]), the position for 'A' is in pos[0], for 'B' in pos[1] and for 'C' in pos[2]. If you call **E816_MOV**(ID, "AC", double pos[2]) the target position for 'A' is in pos[0] and for 'C' in pos[1].

Axes: szAxes = "ABC"	Positions: pos = {1.0, 2.0, 3.0}
<pre>szAxes[0] = 'A'</pre>	pos[0] = 1.0
szAxes[1] = 'B'	pos[1] = 2.0
szAxes[2] = 'C'	pos[2] = 3.0

If you call **E816_MOV**(ID, "AC", double pos[2]) the target position for 'A' is in pos[0] and for 'C' in pos[1].

Each axis identifier is sent only once. Only the **last** occurrence of an axis identifier is actually sent to the controller with its argument. Thus if you call **E816_MOV**(ID, "AAB", pos[3]) with pos[3] = { 1.0, 2.0, 3.0 }, 'A' will move to 2.0 and 'B' to 3.0. If you then call **E816_qPOS**(ID, "AAB", pos[3]), pos[0] and pos[1] will both contain 2.0 as the position of 'A'.

(See E816_MOV() (*p.16*) and E816_qPOS() (*p.18*))

See "Types Used in PI Software," p.9 for a description of types used for parameters.

3.3. Special Axis Identifier A

The E-816 which is directly linked to the host PC with the serial cable is the master. This master has an axis identifier just like all the other controllers. In addition, the master will execute all commands addressed to the special

axis identifier "A". If you do not know the name that has been assigned to the master unit, you can always use "A" to address it. Assigning a unit the axis name "A" with **E816_SCH**() (p.21) erases any previous axis name assignment it might have and makes the unit unreachable as a slave.

4. Types Used in PI Software

4.1. Boolean Values

The library uses the convention used in Microsoft's C++ for boolean values. If your compiler does not support this directly, it can be easily set up: Just add the following lines to a central header file of your project:

```
typedef int BOOL;
#define TRUE 1
#define FALSE 0
```

4.2. NULL Pointers

In the library and the documentation, "null pointers" (pointers pointing nowhere) have the value **NULL**. This is defined in the Windows environment. If your compiler does not know this, simply use:

#define NULL 0

4.3. C-Strings

The library uses the C convention to handle strings. Strings are stored as char arrays with '\0' as terminating delimiter. Thus, the "type" of a c-string is $char^*$. Do not forget to provide enough memory for the final '\0'. If you declare:

char* szText = "HELLO";

it will occupy 6 bytes in memory. To remind you of the zero at the end, the names of the corresponding variables start with "sz".

5. Communication Initialization

To use the DLL and communicate with a E-816 controller the user must initialize the DLL with one of the "open" functions **E816_InterfaceSetupDIg**() (*p.11*) or **E816_ConnectRS232**() (*p.10*). To allow the handling of more than one master controller (i.e. multiple, separate networks), the user will be returned a non-negative "ID" when he calls one of these functions. This is a kind of index to an internal array storing the information for the different controller network. All other calls addressing the same controller network have this ID as first parameter. Moving the RS-232 cable from one E-816 to another on the same I2C network does not affect this mechanism (the E-816s need to be power-cycled when the master is changed, but the host/software need not be reset). E816_CloseConnection() (*p.10*) will close the connection to the specified controller network and free the respective system resources.

5.1. Functions

void E816_CloseConnection (int ID) int E816_ConnectRS232 (const int nPortNr, const long BaudRate) int E816_FindOnRS (int *pnStartPort, int *pnStartBaud) int E816_GetError (int ID) int E816_InterfaceSetupDIg (const char* szRegKeyName) BOOL E816_IsConnected (int ID) BOOL E816_TranslateError (int errNr, char *szBuffer, const int maxlen)

5.2. Function Documentation

void E816_CloseConnection (int ID)

Close connection to E-816 controller network associated with *ID*. *ID* will not be valid any longer.

Parameters:

ID ID of controller network, if **ID** is not valid nothing will happen.

int E816_ConnectRS232 (const int *nPortNr*, const long BaudRate)

Open an RS-232 ("COM") interface to an E-816. All future calls to control this E-816 network need the ID returned by this call.

Parameters:

nPortNr COM port to use (e.g. 1 for "COM1")

BaudRate to use

Returns:

ID of new object, -1 if interface could not be opened or no E-816 is responding.

int E816_FindOnRS (int * pnStartPort, int * pnStartBaud)

Scan available RS-232 ports (up to "COM24") and search for a connected E-816. The scan will open the ports at different baudrates and check

whether an E-816 responds. The baudrates used are 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000 and 256000 baud. The search will start with the value pointed to by *pnStartPort* and go up to port 24. With the value pointed to by *pnBaudRate* you can narrow the range of baudrates to be used. Only baudrates in the above list that are greater than the value specified will be used. If the scan was successful, the values pointed to by *pnStartPort* and *pnStartBaud* contain the values used to open the connection to the E-816. This connection is still active when the function returns, and the E-816 found can be addressed with the returned ID. All future calls to control this E-816 need the ID returned by this call.

Note:

This call may take some time to finish. It will take several 100 milliseconds for each configuration. So if your E-816 is connected on COM4 with 115200 baud and you start from COM1 and search with all baudrates it will take quite long.

Parameters:

pnStartPort pointer to int with the start value for the scan; upon return, if successful, the current port number setting

pnStartBaud pointer to int with the start value for the scan; upon return, if successful, the current baud rate setting in baud.

Returns:

ID of new object or -1 if no E-816 is responding.

int **E816_GetError** (int ID)

Get error status of E816 library and/or the master E-816. This call will also clear the internal error. If there is no internal error the function will call **E816_qERR**() (p.17).

Returns:

error ID, see Error Codes (p.27) for the meaning of the codes.

int **E816_InterfaceSetupDig** (const char* *szRegKeyName*)

Open dialog to let user select the interface and create a new E816 object. All future calls to control this E-816 network need the ID returned by this call. See Interface Settings (p.13) for a detailed description of the dialogs shown.

Parameters:

szRegKeyName key in the window registry to store the settings, the key
used is "HKEY_LOCAL_MACHINE\SOFTWARE\<your keyname>" if
keyname is NULL or "" the default key
"HKEY_LOCAL_MACHINE\SOFTWARE\PI\E816_DLL" is used.

Note:

Use '\\' if you want to create a key and a subkey at once. To create "MyCompany\E816_DLL" you must call

E816_InterfaceSetupDlg("MyCompany\\E816_DLL")

Returns:

ID of new object, -1 if user pressed "CANCEL", the interface could not be opened, or no E-816 is responding.

BOOL E816_IsConnected (int ID)

Check whether there is an E-816 controller network with an ID of ID.

Returns:

TRUE if *ID* points to an existing controller network, **FALSE** otherwise.

BOOL E816_TranslateError (int errNr, char * szBuffer, const int maxlen)

Translate error number to error message.

Parameters:

errNr number of error, as returned from E816_GetError() (p.11).

szBuffer pointer to buffer for the message

maxlen size of the buffer

Returns:

TRUE if successful, **FALSE**, if the buffer was too small to store the message

6. Interface Settings



- COM Port: Select the desired COM port of the PC, something like "COM1" or "COM2. The user will see only the ports available on the system.
- Baud Rate: The baud rate of the interface. Please read the documentation of the connected device to determine the values it supports and how to set them. The settings here and at the device must match. If you move the RS-232 cable to a new master in an E-816 network, make sure it has the proper baud rate.

7. E-816 Commands

These functions encapsulate the embedded commands of the E-816 and provide some shortcuts to make working with the E-816 easier. See **Function Calls** (p.7) for some general notes about the parameter syntax. **Types Used in PI Software** (p.9) will give you some general information about the syntax of most commands.

7.1. Functions

BOOL E816 AVG (int ID, int nAverage) BOOL E816 BDR (int ID, int nBaudRate) BOOL E816_DCO (int ID, char *szAxes, BOOL *pbValarray) BOOL E816 MOV (int ID, const char* szAxes, double *pdValarray) BOOL E816 MVR (int ID, const char* szAxes, double *pdValarray) BOOL E816 gAVG (int ID, int *pnAverage) BOOL E816 gBDR (int ID, int *pnBaudRate) BOOL E816 qDCO (int ID, char *szAxes, BOOL *pbValarray) BOOL E816_gERR (int ID, int *pnError) BOOL E816_ql2C (int ID, int *pnErrorCode, char *pcChannel) BOOL E816_qIDN (int ID, char *buffer, int maxlen) BOOL E816_qMOV (int ID, const char* szAxes, double *pdValarray) BOOL **E816_qONT** (int ID, const char* szAxes, BOOL *pbOnTarget) BOOL **E816_qOVF** (int ID, const char* szAxes, BOOL *pbOverflow) BOOL E816_gPOS (int ID, const char* szAxes, double *pdValarray) BOOL E816 gSAI (int ID, char *axes, const int maxlen) BOOL E816_qSCH (int ID, char *pcChannelName) BOOL E816_qSPA (int ID, const char* szAxes, int *iCmdarray, double *dValarray) BOOL E816 **gSSN** (int ID, char *szAxes, int *piValarray) BOOL E816_qSWT (int ID, char cAxis, int nIndex, double *pdValue) BOOL E816_qSVA (int ID, const char* szAxes, double *pdValarray) BOOL **E816 gSVO** (int ID, char *szAxes, BOOL *pbValarray) BOOL E816_qVOL (int ID, const char* szAxes, double *pdValarray) BOOL E816 RST (int ID) BOOL E816 SCH (int ID, const char cChannelName) BOOL E816 SPA (int ID, const char* szAxes, int *iCmdarray, double *dValarray) BOOL E816 SVA (int ID, const char* szAxes, double *pdValarray) BOOL E816 SVO (int ID, char *szAxes, BOOL *pbValarray) BOOL E816 SVR (int ID, const char* szAxes, double *pdValarray) BOOL E816_SWT (int ID, const char cAxis, const int nIndex, const double dValue) BOOL E816_WPA (int ID, const char* szPassword) BOOL E816_WTO (int ID, const char cAxis, const int nNumber) BOOL E816_WTOTimer (int ID, const char cAxis, const int nNumber, int timer)

7.2. Function Documentation

BOOL E816_AVG (int ID, int *nAverage*)

Corresponding command: AVG

Sets the number of samples to be used when calculating averages on the master unit. Larger values mean more stable output, but slower measurement speed. Must be one of following values: 1, 2, 4, 8, 16, 32 or 64.

Note:

This command will only affect the master E-816.

Parameters:

ID ID of controller network

nAverage number of samples used for average

Returns:

TRUE if successful, FALSE otherwise

BOOL E816_BDR (int ID, int nBaudRate)

Corresponding command: BDR

Set the baud rate of the master. The baud rate can be set to 9600, 19200, 38400, 57600 or 115200 baud. This will only change the setting in the RAM. To store it in the EEPROM call **E816_WPA**() (*p.23*) afterwards. After the next start of the controller the new setting will be used. If you want to change it immediately, call **E816_RST**() (*p.21*) after **E816_WPA**() (*p.23*).

To change the baud rate on the fly use the following code:

```
int ID = E816_ConnectRS232(1, 9600);
if (ID<0) return;
// change settings
if (!E816_BDR(ID, 115200)) return;
// write it to the EEPROM
if (!E816_WPA(ID, szYourPassword)) return;
// restart E-816 with new settings
if (!E816_RST(ID)) return;
// close old 9600 baud connection
E816_CloseConnection(ID);
// the E-816 will need some time to restart with the new settings..
Sleep(5000);
// open new connection
ID = E816_ConnectRS232(1, 115200);
```

Parameters:

ID ID of controller network *nBaudRate* number of samples used for BaudRate

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_DCO** (int ID, const char * *szAxes*, const BOOL * *pbValarray*)

Corresponding command: DCO

Set D/A converter drift compensation "on" or "off". If *pbValarray[index]* is **FALSE** the mode is "off", if **TRUE** it is set to "on". The ON setting eliminates drift of the D/A converter in the E-816 used to provide the target signal to the separate E-802 servo-control submodule. This setting is recommended for static operation, but should be turned OFF for dynamic operation. The drift compensation setting is ignored during wave table output. See "Drift Compensation" and the DCO command in the E-816 User Manual for more details.

Parameters:

ID of controller network

szAxes string with axes

pbValarray modes for the specified axes, TRUE for "on", FALSE for "off"

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_MOV** (int ID, const char* *szAxes*, double * *pdValarray*)

Corresponding command: MOV

Move *szAxes* to absolute positions.

Parameters:

ID ID of controller network *szAxes* string with axes *pdValarray* target positions for the axes

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_MVR** (int ID, const char* *szAxes*, double * *pdValarray*)

Corresponding command: MVR

Move *szAxes* relative to current position.

Parameters:

ID ID of controller network szAxes string with axes pdValarray target positions for the axes Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qAVG** (int ID, int * *pnAverage*)

Corresponding command: AVG?

Get the number of samples used for average calculations by the master E-816.

Note:

This command will only query master E-816.

Parameters:

ID ID of controller network

pnAverage pointer to int for storing the number of samples used for average

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qBDR** (int ID, int * *pnBaudRate*)

Corresponding command: BDR?

Get current RAM baudrate setting of the master. This is the value that will be saved to ROM by **E816_WPA** and may differ from both the power-up and/or the current operating value. See **E816_BDR**() (p.15) for information on how to change the baudrate.

Parameters:

ID ID of controller network *pnBaudRate* pointer to int for storing the baudrate

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qDCO** (int ID, char * *szAxes*, BOOL * *pbValarray*)

Corresponding command: DCO?

Get the state of the D/A converter drift compensation for szAxes

Note: See "Drift Compensation" and the DCO command in the E-816 User Manual for details

Parameters:

ID ID of controller network
 szAxes string with axes
 pbValarray array to be filled with the servo-mode values for the specified axes,
 TRUE for "on", FALSE for "off"

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qERR** (int ID, int * *pnError*)

Corresponding command: ERR?

Get the error state of the master E-816. It is safer to call **E816_GetError**() (*p.11*) because this will also return the internal error state of the library.

Note:

This command will only query the master E-816.

Parameters:

ID ID of controller network *pnError* variable for storing error code of the master controller

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_ql2C** (int ID, int * *pnErrorCode*, char * *pcChannel*)

Corresponding command: 12C?

Get the state if the I2C bus connecting networked E-816s. The status is returned as a bitmap. The bit definition viewpoint is that of the master.

- bit 0: CHK_SEN0 timeout
- bit 1: CHK_PEN0 timeout
- bit 2: CHK_RSEN0 timeout
- bit 3: CHK_RWO timeout
- bit 4: CHK_BFO timeout
- bit 5: CHK_BF1 timeout
- bit 6: CHK_ACK0 timeout
- bit 7 (LSB): SLAVE_BUSY timeout

Parameters:

ID of controller network

pnErrorCode pointer to int for storing the bitmap with errors **pcChannel** pointer to char for storing the associated channel name

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qIDN** (int ID, char * *buffer*, int *maxlen*)

Corresponding command: *IDN?

Get identification string of the master controller.

Note:

This command will only query the master E-816.

Parameters:

ID ID of controller network *buffer* buffer for storing the string read in from controller *maxlen* size of *buffer*, must be given to avoid a buffer overflow.

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qMOV** (int ID, const char* *szAxes*, double * *pdValarray*)

Corresponding command: MOV?

Read the commanded target positions for *szAxes*.

Parameters:

ID of controller network
 szAxes string with axes
 pdValarray array to be filled with target positions of the axes
 Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qONT** (int ID, const char* *szAxes*, BOOL * *pbOnTarget*)

Corresponding command: ONT?

Check if *szAxes* have reached target position.

Parameters:

ID ID of controller network
 szAxes string with axes
 pbOnTarget array to be filled with current on-target status of the axes: TRUE for on target, FALSE otherwise

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qOVF** (int ID, const char* *szAxes*, BOOL * *pbOverflow*)

Corresponding command: OVF?

Check overflow status of *szAxes*.

Parameters:

ID ID of controller network

szAxes string with axes

pbOverflow array to be filled with current overflow status of the axes. The overflow status is supplied to the E-816 as a voltage level from other modules/submodules in the system. See the respective manuals for details.

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qPOS** (int ID, const char* *szAxes*, double * *pdValarray*)

Corresponding command: POS?

Get the current positions of *szAxes*.

Parameters:

ID of controller network

szAxes string with axes

pdValarray array to be filled with current positions of the axes

Returns:

TRUE if successful, FALSE otherwise

BOOL E816_qSAI (int ID, char * axes, const int maxlen)

Corresponding command: SAI?

Get connected axes. Each character in the returned string is an axis identifier for one connected axis.

Note:

This command will be executed by the master E-816 but provides data about the entire network. If axes turn up missing, try power-cycling the E-816s.

Parameters:

ID ID of controller network*axes* buffer to store the string read in*maxlen* size of *buffer*, must be given to prevent a buffer overflow.

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qSCH** (int ID, char * *pcChannelName*)

Corresponding command: SCH?

Get channel name of master E-816

Note:

In addition to any name reported here, the master E-816 is always addressable with the special axis identifier "A".

Parameters:

ID ID of controller network

pcChannelName pointer to char for storing the channel name of the master

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qSPA** (int ID, const char* *szAxes*, int * *iCmdarray*, double * *dValarray*)

Corresponding command: SPA?

Read current parameter settings for *szAxes*. Since this command may only be addressed to the master, other axis names in *szAxes* will cause an error (**PI_INVALID_AXIS_IDENTIFIER**).

For each parameter you wish to query, you must respecify the axis (must be master) in *szAxes* and a parameter ID in the corresponding element of *iCmdarray*. The parameter ID can have following values:

- 1 for VAD gain
- 2 for VAD offset
- 3 for PAD gain
- 4 for PAD offset
- 5 for DA gain
- 6 for DA offset
- 7 for KSen
- 8 for OSen
- 9 for Kpzt
- 10 for Opzt

Note:

This command queries only the master E-816. For compatibility, however, you must include an axis identifier string. To be sure you address only the master controller, using all A's is recommended.

Parameters:

ID ID of controller network*szAxes* master axis designators (e.g. "AAAA")*iCmdarray* IDs of parameters to query

dValarray array to be filled with the values of the parameters

Returns:

TRUE if successful, FALSE otherwise

Errors:

PI_INVALID_SPA_CMD_ID one of the IDs in *iCmdarray* is not valid, must be in 1-10

BOOL **E816_qSSN** (int ID, char * *szAxes*, int * *piValarray*)

Corresponding command: SSN?

Get serial numbers of the connected controllers.

Parameters:

ID ID of controller network*szAxes* string with axis designator(s)*piValarray* array to be filled with the serial numbers

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qSVA** (int ID, const char* *szAxes*, double * *pdValarray*)

Corresponding command: SVA?

Read the commanded PZT voltages for szAxes. (see also E816_qVOL() (p.21))

Parameters:

ID ID of controller network *szAxes* string with axes

SZAXES SUNY WIT AXES

pdValarray array to be filled with the voltage values for the axes

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qSWT** (int ID, const char *cAxis*, const int *nIndex*, const double **pdValue*)

Corresponding command: SWT?

Get wave table data. Each E-816 has a wave table with 64 entries. With this command you can read a value from the table.

Parameters:

ID ID of controller network*cAxis* channel name of the axis*nIndex* index for table entry, must be in 0-63*pdValue* pointer to value to be filled with wave table entry

Returns:

TRUE if successful, FALSE otherwise

BOOL E816_qSVO (int ID, char * szAxes, BOOL * pbValarray)

Corresponding command: SVO?

Get the servo modes for szAxes

Note:

The E-816 simply provides a servo-on signal to a separate servo-controller. That controller must be so configured as to allow computer-control.

Parameters:

ID ID of controller network
 szAxes string with axes
 pbValarray array to be filled with the servo-mode values for the specified axes,
 TRUE for "on", FALSE for "off"

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_qVOL** (int ID, const char* *szAxes*, double * *pdValarray*)

Corresponding command: VOL?

Get current PZT voltages for *szAxes*. (see also **E816_qSVA**() (*p.20*))

Parameters:

ID of controller network
 szAxes string with axes
 pdValarray array to be filled with the current voltages for the axes
 Returns:

TRUE if successful, FALSE otherwise

BOOL E816_RST (int ID)

Corresponding command: RST

Restart master E-816 controller. Use this command if you want to use settings saved with **E816_WPA**() (*p.23*) without power-cycling the E-816. The master unit remains master after a reset, even if the RS-232 cable is pulled before the unit is ready, so changing masters requires power-cycling.

Note:

Use this command if you want to use settings saved with **E816_WPA**() (p.23) without power-cycling the E-816. The master unit remains master after a reset, even if the RS-232 cable is pulled before the unit is ready, so changing masters requires power-cycling.

The E-816 will need some time to restart (up to several seconds), so subsequent commands may fail with a timeout error if you do not wait.

Parameters:

ID of controller network

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_SCH** (int ID, const char *cChannelName*)

Corresponding command: SCH

Set the channel name for the master controller. This will only change the setting in the RAM. To store it in the EEPROM call **E816_WPA**() (*p.23*) afterwards. After the next controller restart the new setting will be used.

Note:

Giving the master an axis name other than "A" makes it possible to use that E-816 as a slave.

Parameters:

ID ID of controller network

cChannelName new channel name of the master, if "A", the unit will not be reachable as slave

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_SPA** (int ID, const char* *szAxes*, int * *iCmdarray*, double * *dValarray*)

Corresponding command: SPA

Set parameters for *szAxes*. You must include an axis identifier in *szAxes* (master axis names only, e.g. "AAA"!) for each parameter you wish to set. Missing or non-master axis names in *szAxes* will cause an error (**PI_INVALID_AXIS_IDENTIFIER**). The function uses

the parameter IDs in the *iCmdarray* and sets the parameters to the values in the corresponding elements of *dValarray*. The parameter IDs can have following values:

- 7 for KSen
- 8 for OSen
- 9 for Kpzt
- 10 for Opzt

Unlike the other functions, E816_SPA has two arrays as arguments. The first array has the parameters which have to be modified, the second one the values. If you want to set KSen (ID=7) to 1.0 and OSen (ID=8) to 10.0, you must call E816_SPA(id, "AA", $\{7, 8\}, \{1.0, 10.0\}$)

szAxes = "AA"	$cmd = \{7, 8\}$	values = {1.0, 10.0}
<pre>szAxes[0] = 'A'</pre>	$\operatorname{cmd}[0] = 7$	values[0] = 1.0
<pre>szAxes[1] = 'A'</pre>	cmd[1] = 8	values[1] = 10.0

Note:

This command works only with the master E-816. For compatibility, however, you must include an axis identifier string. To be sure you address only the master controller, using all A's in the axis identifier string is recommended. If the same parameter ID appears more than once in *iCmdarray*, the **last** value will be

set. For example E816_SPA(id, "AAA", {7, 7, 9}, {10.0, 20.0, 30.0}) will set the KSen of axis A to 20.0 and the Kpzt to 30.0.

Parameters:

ID ID of controller network

szAxes axes for which the parameter should be set, must consist only of the channel name of the master or the special name "A" (to be sure, use of "AAA..." is recommended)

iCmdarray parameter IDs (see above)

dValarray array with the values for the parameters

Returns:

TRUE if successful, FALSE otherwise

Errors:

PI_INVALID_SPA_CMD_ID one of the IDs in *iCmdarray* is not valid, must each be one of {7,8,9,10}

BOOL **E816_SVA** (int ID, const char* *szAxes*, double * *pdValarray*)

Corresponding command: SVA

Set PZT voltages for *szAxes* to absolute values. **Parameters:**

ID ID of controller network *szAxes* string with axes *pdValarray* voltages for the axes Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_SVO** (int ID, char * *szAxes*, BOOL * *pbValarray*)

Corresponding command: SVO

Set servo-control "on" or "off" (closed-loop / open-loop mode). If *pbValarray[index]* is **FALSE** the mode is "off", if **TRUE** it is set to "on"

Parameters:

ID ID of controller network
 szAxes string with axes
 pbValarray modes for the specified axes, TRUE for "on", FALSE for "off"

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_SVR** (int ID, const char* *szAxes*, double * *pdValarray*)

Corresponding command: SVR

Set PZT voltages for *szAxes* relatively, i.e. increase current voltages by the specified values. **Parameters:**

ID ID of controller network *szAxes* string with axes *pdValarray* values to be added to voltage of the axes

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_SWT** (int ID, const char *cAxis*, const int *nIndex*, const double *dValue*)

Corresponding command: SWT

Set wave table data. Each E-816 has a wave table with 64 entries. With this commandyou can place a value in the table.

Parameters:

ID ID of controller network
 cAxis channel name of the axis
 nIndex index for table entry, must be in 0-63
 dValue new value for wave table entry

Returns:

TRUE if successful, FALSE otherwise

BOOL E816_WPA (int ID, const char* szPassword)

Corresponding command: WPA

Write parameters set with **E816_SPA**() (p.21), **E816_AVG**() (p. 14), **816_SCH**() (p.21) and **E816_BDR**() (p.15) to internal flash ROM in the master E-816, where they become poweron defaults. To protect the data in the flash ROM you must provide a password, see the WPA command in the *E-816 User Manual* for details.

Note:

This command will only affect the master E-816.

Parameters:

ID of controller network

szPassword password needed to store the parameters

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **E816_WTO** (int ID, const char *cAxis*, const int *nNumber*)

Corresponding command: WTO

Start or stop the wave table output.

Parameters:

ID ID of controller network *cAxis* channel name of the axis *nNumber* if 0, the output is stopped, otherwise the output is started from index 0 to (*nNumber-1*). *nNumber* must be in 0-64

Returns:

TRUE if successful, FALSE otherwise

BOOL **E816_WTOTimer** (int ID, const char *cAxis*, const int *nNumber*, int *timer*)

Corresponding command: WTO

Start or stop the wave table output with specified tact.

Parameters:

ID ID of controller network
 cAxis channel name of the axis
 nNumber if 0, the output is stopped, otherwise the output of points with index 0
 through (*nNumber-1*) is started and continues indefinitely; *nNumber* must be in 0-64.
 Stop with an E816_WTOTimer() or E816_WTO () with *nNumber* = 0
 timer time in milliseconds between output of the consecutive points

Returns:

TRUE if successful, FALSE otherwise

8. GCS COM Server

For some programming languages it is much simpler to use a COM (Component Object Model) Server than to link to DLL functions. In particular, Visual Basic and other script languages (e.g. Python, Perl) provide good support for COM objects. See the provided samples for how to integrate the GCS COM into the different languages / development environments. Sample programs and the appropriate source code are to be found in the \Samples directory of the product CD.

The functions are more or less the same as provided by the DLL, so this manual can be used to get to know the basic functionality. There are however fundamental syntax differences:

- No controller ID, since you can create instances of the COM object for every single controller connected (see Section 8.1)
- With COM it is possible to allocate space for strings and arrays by the callee without disturbing the caller, so there is no need to send any buffer sizes or array lengths to the COM functions (see Section 8.2)

8.1. No Need for Controller IDs

You can create instances for every controller connected. So when you need to write in C or C++ $\,$

```
int ID1;
int ID2;
ID1 = E816_ConnectRS232(1, 115200);
ID2 = E816_ConnectRS232(2, 115200);
if (!E816_IsConnected(ID1))
    printf("Could not connect to controller 1";
if (!E816_IsConnected(ID2))
    printf("Could not connect to controller 2";
```

you can write in Visual Basic

```
Dim E8161 As New E816
Dim E8162 As New E816
E8161.ConnectRS232(1, 115200)
E8162.ConnectRS232(2, 115200)
If Not E8161.IsConnected Then
    Me.Caption = "Could not connect to controller 1"
End If
If Not E8162.IsConnected Then
    Me.Caption = "Could not connect to controller 2"
End If
```

8.2. No Need for Buffer Sizes

If you have DLL functions and you want to read a string from the DLL, you need to allocate the neccessary space and tell the DLL how large the buffer is. To the COM server however you hand over a string "object". The COM server can let the string grow and the string object itself holds all the neccessary information about length and needed memory. So when you need to write in C or C++ for the DLL something like

```
char sIDN[1024];
E816_qIDN( ID, sIDN, 1024 );
```

you can simply write (e.g. in Visual Basic)

Dim sIDN As String E816.qIDN(sIDN)

9. Error Codes

The error codes are defined in separate header files shipped with the E-816 GCS_DLL.

The error codes listed here are those of the *PI General Command Set.* As such, some are not relevant to the E-816 and will simply never occur with the systems this manual describes.

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

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
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	While MOV! is running position can only be estimated!
219	PI_CNTR_POSITION_BASED_ON_INTERPOLATION	Position was calculated during MOV motion
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

Controller already has a serial number

Macro recording not activated

Invalid parameter for MAC

PI internal error code 1004

Sector erase failed

Flash program failed

HW match code missing/invalid

FW match code missing/invalid

HW version missing/invalid

FW version missing/invalid

Flash read failed

FW update failed

1002	PI_CNTR_NO_MACRO_RECORDING
1003	PI_CNTR_INVALID_MAC_PARAM
1004	PI_CNTR_RESERVED_1004
2000	PI_CNTR_ALREADY_HAS_SERIAL_NUMBER
4000	PI_CNTR_SECTOR_ERASE_FAILED
4001	PI_CNTR_FLASH_PROGRAM_FAILED
4002	PI_CNTR_FLASH_READ_FAILED
4003	PI_CNTR_HW_MATCHCODE_ERROR
4004	PI_CNTR_FW_MATCHCODE_ERROR
4005	PI_CNTR_HW_VERSION_ERROR
4006	PI_CNTR_FW_VERSION_ERROR

4007 PI_CNTR_FW_UPDATE_ERROR

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		
DLL Er	DLL Errors			

-1001	PI_UNKNOWN_AXIS_IDENTIFIER	Unknown axis identifier
-1002	PI_NR_NAV_OUT_OF_RANGE	Number for NAV out of rangemust be in [1,10000]

www.pi.ws

-1003	PI_INVALID_SGA	Invalid value for SGAmust 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 errorcould 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 rangemust 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
-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.

10. Index

*IDN? 17 AVG 14 AVG? 16 axes parameter 7 **BDR** 15 BDR? 16 BOOL 9 boolean values 9 c strings 9 Communication Initialization 10 DCO 15 DCO? 16 dynamic loading of a DLL 6 E-516 Commands 14 E816_AVG 14 E816_BDR 15 E816_CloseConnection 10 E816 ConnectRS232 10 E816 DCO 15 E816_FindOnRS 11 E816_GetError 11 E816_InterfaceSetupDlg 11 E816_IsConnected 12 E816_MOV 16 E816_MVR 16 E816_qAVG 16 E816 gBDR 16 E816 qDCO 16 E816 gERR 17 E816_qI2C 17 E816_qIDN 17 E816 qMOV 18 E816_qONT 18 E816_qOVF 18 E816_qPOS 18 E816_qSAI 19 E816 qSCH 19 E816_qSPA 19 E816 qSSN 20 E816 gSVA 20 E816_qSVO 21 E816_qSWT 20 E816 qVOL 21 E816 RST 21 E816_SCH 21

E816_SetErrorCheck 12 E816_SPA 22 E816_SVA 23 E816_SVO 23 E816_SVR 23 E816_SWT 23 E816 TranslateError 12 E816_WPA 24 E816_WTO 24 E816_WTOTimer 24 ERR? 17 Error codes 27 FALSE 9 GetProcAddress - Win32 API function 6 I2C? 17 LIB - static import library 5 linking a DLL 5 LoadLibrary - Win32 API function 6 module definition file 5 MOV 16 MOV? 18 **MVR** 16 NULL 9 **ONT**? 18 **OVF**? 18 POS? 18 **RST 21** SAI? 19 **SCH 21** SCH? 19 SPA 22 SPA? 19 SSN? 20 static import library 5 SVA 23 SVA? 20 SVO 23 SVO? 21 **SVR 23 SWT 23** SWT? 20 TRUE 9 VOL? 21 WPA 24 WTO 24