Piezo · Nano · Positioning

PZ120E Software Manual

E-816 Windows GCS DLL

Windows Library Reference

Release: 2.0.0 Date: 2009-03-25

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

- E-816
  Computer Interface and Command Interpreter Submodule (firmware version 3.20 and newer) for Piezo 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

The products described in this manual are in part protected by the following patents:
US-Patent No. 6,950,050

First printing 2009-03-25
Document Number PZ120E Eco, BRo, Release 2.0.0
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 This manual is superseded by any new release.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 www.pi.ws (http://www.pi.ws).

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

The E-816 DLL allows controlling one or more PI E-816 Computer Interface and Command Interpreter Submodules connected to a host PC based on the PI General Command Set (GCS). GCS is the PI standard command set and ensures the compatibility between different PI controllers.

The library is available for the following operating systems:

- **Windows** 2000, XP and Vista: E-816 DLL
  See Sections 3 to 5 for more information about PI DLLs.
- **Linux** operating systems (kernel 2.6, GTK 2.0, glibc 2.4): libpi_e816.so.*x.x.x* and libpi_e816-*x.x.x*.a where *x.x.x* gives the version of the library

NOTES

This manual was originally written for the Windows version of the E-816 library (DLL), and so the terminology used in this document is that common with Windows DLLs. Nevertheless this manual can also be used for the Linux versions of the E-816 library because there is no difference in the functionality of the library functions between the individual operating systems.

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. Master Unit

Several E-816s can be connected together in a network using the I^2C bus lines provided. One device of the network is connected to the host PC with an RS-232 or USB 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.

To change the master, you must reconnect the RS-232 or USB cable and power-cycle the E-816s. With an RS-232 connection, the master can be changed while the library is active. With a USB connection, the connection must be closed before changing the master, and re-opened afterwards.

The term “E-816 network” as used in this manual should be understood to refer to a single (non-networked) E-816 as well. See the E-816 User Manual for more information on networking.

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 which may only be addressed to the master. If you want to send them to a slave unit, e.g. to change its configuration, first make it the master. The master-only commands are:

E816_qERR() (p.28)
E816_qIDN() (p.29)
E816_qI2C() (p.29)
E816_qSSN() (p. 32)
E816_qCCL() (p. **Fehler! Textmarke nicht definiert.**)
E816_BDR() (p.23)
E816_qBDR() (p.27)
E816_AVG() (p.23)
E816_qAVG() (p.27)
E816_qSAI() (p. 31)
E816_SCH() (p.33)
E816_qSCH() (p.31)
E816_SPA() (p.34)*
E816_qSPA() (p.31)*
E816_WPA() (p.37)
E816_RST() (p.33)
E816_DEL() (p. 24)
E816_MAC_BEG() (p. 24)
E816_MAC_DEF() (p. 24)
E816_MAC_DEL() (p. 25)
E816_MAC_END() (p. 25)
E816_MAC_NSTART() (p. 25)
E816_MAC_qDEF() (p. 25)
E816_MAC_qFREE() (p. 26)
E816_MAC_START() (p. 26)
E816_qMAC() (p. 29)
E816_IsRunningMacro() (p. 24)
E816_qHLP() (p. 28)

*With firmware revisions 2.xx, E816_SPA() and E816_qSPA() are only available for the master unit. With firmware revision 3.20 and newer they can also be used with slave units.

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.31*)).

2. Quick Start

2.1. Software Installation

To install the E-816 DLL on your host PC, proceed as follows:

Windows operating systems:

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

Linux operating systems:

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

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

See Sections for more information about PI DLLs.

2.2. Connect the Controller

Physically connect the controller (respective the master unit of an I^2C-network) to the PC. The RS-232 and USB interfaces can be active simultaneously.

With USB connections, communication can not be maintained after the E-816 is power-cycled or reset. The connection must then be closed and reopened.

2.3. Install USB Driver

The first time you connect over the USB interface, be sure you are logged on the PC as a user having administrator rights. After the E-816 is powered on, a message will appear saying that new hardware has been detected. Follow the on-screen instructions and insert the E-816 CD. The required hardware driver is found in the \USB_Driver directory.

2.4. Starting Up

NOTES

The following E-816 factory defaults are valid for the first start-up, unless agreed otherwise before delivery:

- Number of readings to use for an average: 32
  Can be changed using E816_AVG()

- Channel name (= axis identifier): A
  Can be changed using E816_SCH()

- Data rate: 115,200 baud
  Can be changed in the range of 9,600 to 115,200 baud using E816_BDR()

Values set with E816_AVG(), E816_SCH() and E816_BDR() can be saved to non-volatile memory where they become the new power-on defaults. See the E816_WPA() description for details.

When controlling the E-816, timing problems can occur if several functions for GCS commands are run in rapid sequence, resulting in lost commands. To prevent such communication errors, it is recommended that you include a certain wait time between the different programming steps, depending on the command to be executed. This is especially true for command functions that need a certain execution time inside the E-816 module, like E816_MOV(), E816_MVR(), E816_SPA(), E816_SVA(), E816_SVR(), E816_RST(), E816_WPA(), E816_SWT() and E816_WTO().

Using a start-up macro, you can set up the device to start with closed-loop operation. See the E-816 User Manual for details.

The term “E-816 network” as used in this manual should be understood to refer to a single (non-networked) E-816 as well.

Preparation:

1. Set up the units in which the E-816s are installed (e.g. E-621, E-625 or E-665 controllers) as described in the “Starting Operation” or “Quick Start” section of the corresponding User Manual.

   In particular, be sure the units are configured to allow computer-controlled operation (see “Control Modes” in the E-816 User Manual and the User Manual of the piezo control electronics for details). In computer-controlled operation, axis motion can then be caused by move commands (e.g. E816_MOV() or E816_SVA(); received via interface or from a running macro), wave table output (E816_WTO()) and trigger input (E816_MVT()).
2. If you are planning to run networked E-816s, prepare the system for networking:

   A unique axis identifier (also referred to as “channel name”) must be assigned to each of the units. E-816s delivered together installed in the same chassis (e.g. E-621 modules) come preconfigured with unique axis identifiers, but the identifiers of E-816s installed in stand-alone devices such as E-625s or E-665 by default are all set to “A” and need to be changed.

   Follow the instructions in “Setting Channel Names” in the E-816 User Manual to change the channel name of an E-816, and see “Checking Connection and Master Unit” in the E-816 User Manual for how to check the settings.

   If all axis identifiers were adapted, interconnect the units which are to be networked as described in “Interlinking Multiple E-816s” in the E-816 User Manual. After interconnecting all units, power-cycle them.

Write a program that performs the following steps:

1. Open a connection between the host PC and the E-816 network, e.g. by calling E816_ConnectRS232()
2. Call E816_qSAI() to determine which axes are present in the network
3. Call E816_qSCH() to determine the axis identifier of the master unit
4. Call E816_qSVO() to check the current servo mode of the axes (open-loop or closed-loop operation). If you want to change the servo mode, use E816_SVO().
5. Make a few test moves to verify your program’s operation:
   In open-loop operation, use E816_SVA() or E816_SVR().
   In closed-loop operation, use E816_MOV() or E816_MVR().

2.5. Samples

There are various sample programs for different programming languages to be found in the \Sample directory of the E-816 CD. The sample code below shows how to connect to an E-816 over USB.

```
char axes[10];
int ID;

// connect to the E-816 over USB
char szDevices[1000];
int nrDevices = E816_EnumerateUSB(szDevices, 999, NULL);
if (nrDevices<=0)
{
    printf("No devices connected to USB");
    goto exit;
}
char* p = strtok(szDevices, "\n");
printf("Found %d devices, connecting to first: \"%s\"\n", nrDevices, szDevices);
ID = E816_ConnectUSB(szDevices);
if (ID<0)
    return FALSE;

if (!E816_qSAI(ID, axes, 9))
    return FALSE;

printf("qSAI() returned \"%s\"", axes);
```

3. 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.)

3.1. 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.

3.2. 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
```

3.3. 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)
```

4. Function Calls

Almost all functions will return a boolean value of type BOOL (see "Types Used in PI Software," p.14. If the function succeeds the return value is TRUE, otherwise it is FALSE. To find out what went wrong, call **E816_GetError**() (*p.17*) and look up the value returned in "Error Codes," *p.39*.

4.1. 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 “Functions for Communication Initialization” p.15) 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.

4.2. 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}`
`szAxes[0] = 'A'`	`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.**26***) and **E816_qPOS**() (*p.30*))

See "Types Used in PI Software,” p.14 for a description of types used for parameters.

4.3. Special Axis Identifier A

The E-816 which is directly linked to the host PC with the serial or USB 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.33*) erases any previous axis name assignment it might have and makes the unit unreachable as a slave.

5. Types Used in PI Software

5.1. 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
```

5.2. 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
```

5.3. 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`".

6. Functions for 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_InterfaceSetupDlg**() (*p.17*), **E816_ConnectRS232**() (*p.16*) or **E816_ConnectUSB**(). Before connecting a device using the E816_ConnectUSB() function, its description string should be queried by **E816_EnumerateUSB**().

Notes

The first time you connect over the USB interface, be sure you are logged on the PC as a user having administrator rights. After the E-816 is powered on, a message will appear saying that new hardware has been detected. Follow the on-screen instructions and insert the E-816 CD. The required hardware driver is found in the \USB_Driver directory.

With USB connections, communication can not be maintained after the E-816 is power-cycled or reset. The connection must then be closed and reopened.

When controlling the E-816, timing problems can occur if several functions for GCS commands are run in rapid sequence, resulting in lost commands. To prevent such communication errors, it is recommended that you include a certain wait time between the different programming steps, depending on the command to be executed. This is especially true for command functions that need a certain execution time inside the E-816 module, like E816_MOV(), E816_MVR(), E816_SPA(), E816_SVA(), E816_SVR(), E816_RST(), E816_WPA(), E816_SWT() and E816_WTO().

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 the “open” functions. This is a kind of index to an internal array storing the information for the different controller networks. All other calls addressing the same controller network have this ID as first parameter. **E816_CloseConnection**() (*p.16*) will close the connection to the specified controller network and free the respective system resources.

To change the master unit of a network, you must reconnect the RS-232 or USB cable and power-cycle the E-816s. With an RS-232 connection, the master can be changed while the library is active. With a USB connection, the connection must be closed before changing the master, and re-opened afterwards.

6.1. Overview

void **E816_CloseConnection** (int ID)
int **E816_ConnectRS232** (const int nPortNr, const long BaudRate)
int **E816_ConnectUSB** (const char* szDescription)
int **E816_EnumerateUSB** (char* szBuffer, int iBufferSize, const char* szFilter)
int **E816_FindOnRS** (int *pnStartPort, int *pnStartBaud)
int **E816_GetError** (int ID)
int **E816_InterfaceSetupDlg** (const char* szRegKeyName)
BOOL **E816_IsConnected** (int ID)
BOOL **E816_TranslateError** (int errNr, char *szBuffer, const int maxlen)

6.2. Function Description

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_ConnectUSB** (const char* *szDescription*)

Open a USB connection to a controller using one of the identification strings listed by E816_EnumerateUSB(). All future calls to control this controller need the ID returned by this call. Will fail if there is already a connection.

Parameters:

szDescription the description of the controller returned by E816_EnumerateUSB()

Returns:

ID of new object, -1 if interface could not be opened or no controller is responding, or the controller responds that it is already connected via USB.

int **E816_EnumerateUSB** (char* *szBuffer*, int *iBufferSize*, const char* *szFilter*)

Lists the identification strings of all controllers available via USB interfaces. Using the mask, you can filter the results for certain text.

Parameters:

szBuffer buffer for the USB devices description.

iBufferSize size of the buffer

szFilter only controllers whose descriptions match the filter are returned in the buffer (e.g. a filter of “E-816” will only return the E-816 controllers, and not all PI controllers).

Returns:

>= 0: the number of controllers in the list
<0: Error code

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.28*).

Returns:

error ID, see Error Codes (*p.39*) for the meaning of the codes.

int **E816_InterfaceSetupDlg** (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. 18) 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.17*).

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

6.3. Interface Settings

With **E816_InterfaceSetupDlg** (), p. 17, the *Connect* dialog is called. This dialog offers interface tab cards where you can configure and establish the connection (see descriptions below). Note that not all of the interfaces shown via the tab cards may be present on your controller.

RS-232

- `COM Port`: Select the desired COM port of the PC, something like "COM1" or "COM2”. Only the ports available on the system are displayed.
- `Baud Rate`: The baud rate of the interface. The baud rate chosen will be set on both the host PC and the controller side of the interface.

  If you move the RS-232 cable to a new master in an E-816 network, make sure it has the proper baud rate.

USB

- All controllers available via USB are listed. Click on the controller to which you want to connect, and click the “Connect” button.

7. Functions for Sending and Reading Strings

CAUTION

Do not mix up the usage of E816_GcsCommandset, E816_GcsGetAnswer and E816_GcsGetAnswerSize with the usage of the PI library functions for GCS commands which are described in Section “Functions for E-816 Commands” on p. 20.

7.1. Overview

BOOL **E816_GcsCommandset** (int ID, const char* szCommand)
BOOL **E816_GcsGetAnswer** (int ID, char* szAnswer, int iBufferSize)
BOOL **E816_GcsGetAnswerSize** (int ID, int* piAnswerSize)

7.2. Function Description

BOOL **E816_GcsCommandset** (int *ID*, const char* *szCommand*)

Sends a GCS command to the controller. Any GCS command can be sent, but this command is intended to allow use of commands not having a function in the current version of the library.

See the User Manual of the controller for a description of the GCS commands which are understood by the controller firmware, for a command reference and for any limitations regarding the arguments of the commands.

Parameters:

ID ID of controller
szCommand the GCS command as string

Returns:

TRUE if no error, FALSE otherwise

BOOL **E816_GcsGetAnswer** (int *ID*, char* *szAnswer*, int *iBufferSize*)

Gets the answer to a GCS command, provided its length does not exceed *bufsize*. The answers to a GCS command are stored inside the DLL, where as much space as necessary is obtained. Each call to this function returns and deletes the oldest answer in the DLL.

See the User Manual of the controller for a description of the GCS commands which are understood by the controller firmware, for a command reference and for any limitations regarding the arguments of the commands.

Parameters:

ID ID of controller
szAwnser the buffer to receive the answer.
iBufferSize the size of *szAnswer*.

Returns:

TRUE if no error, FALSE otherwise

BOOL **E816_GcsGetAnswerSize** (int *ID*, int* *piAnswerSize*)

Gets the size of an answer of a GCS command.

Parameters:

ID ID of controller
piAnswerSize pointer to integer to receive the size of the oldest answer waiting in the DLL.

Returns:

TRUE if no error, FALSE otherwise

8. Functions for 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.12) for some general notes about the parameter syntax. **Types Used in PI Software** (p.14) will give you some general information about the syntax of most commands.

NOTE

When controlling the E-816, timing problems can occur if several functions for GCS commands are run in rapid sequence, resulting in lost commands. To prevent such communication errors, it is recommended that you include a certain wait time between the different programming steps, depending on the command to be executed. This is especially true for command functions that need a certain execution time inside the E-816 module, like E816_MOV(), E816_MVR(), E816_SPA(), E816_SVA(), E816_SVR(), E816_RST(), E816_WPA(), E816_SWT() and E816_WTO().

8.1. Overview

Function	Short Description	Page
BOOL **E816_AVG** (int ID, int nAverage)	Set number of samples to use for averages on master unit	23
BOOL **E816_BDR** (int ID, int nBaudRate)	Set master unit baudrate	23
BOOL **E816_DCO** (int ID, char *szAxes, BOOL *pbValarray)	Set D/A converter drift compensation on or off	23
BOOL **E816_DEL** (int ID, double *dmSeconds*)	Delay the controller for n milliseconds, recommended for usage in macro operation	24
BOOL **E816_IsRunningMacro** (int ID, BOOL * *pbRunningMacro*)	Test if a macro is running on master unit	24
BOOL **E816_MAC_BEG** (int ID, const char * *szName*)	Calls macro function on master unit: Start recording macro	24
BOOL **E816_MAC_DEF** (int ID, const char * *szMacroName*)	Calls macro function on master unit: Set the specified macros as start-up macro	24
BOOL **E816_MAC_DEL** (int ID, const char * *szMacroName*)	Calls macro function on master unit: Delete macro	25
BOOL **E816_MAC_END** (int ID)	Calls macro function on master unit: End macro recording	25
BOOL **E816_MAC_NSTART** (int *ID*, const char * *szName*, int *nrRuns*)	Calls macro function on master unit: Execute macro n times, n should be in the range from 1 to 65535	25
BOOL **E816_MAC_qDEF** (int *ID*, char * *szBuffer*, int *iBufferSize*)	Calls macro function on master unit: Ask name of start-up macro	25
BOOL **E816_MAC_qFREE** (int *ID*, int* pNumberChars)	Calls macro function on master unit: Get free memory to store additional macros	26
BOOL **E816_MAC_START** (int *ID*, const char * *szName*)	Calls macro function on master unit: Start macro (single run)	26

Function	Short Description	Page
BOOL **E816_MOV** (int ID, const char* szAxes, const double *pdValarray)	Move the given axis to absolute position	26
BOOL **E816_MVR** (int ID, const char* szAxes, const double *pdValarray)	Move the given axis relative to current position	26
BOOL **E816_MVT** (int ID, const char *szAxes, const BOOL *pbValarray)	Set “move triggered” mode on/off for the given axis	27
BOOL **E816_qAVG** (int ID, int *pnAverage)	Get number of samples being used for averages on master unit	27
BOOL **E816_qBDR** (int ID, int *pnBaudRate)	Get master unit baudrate	27
BOOL **E816_qDCO** (int ID, char *szAxes, BOOL *pbValarray)	Get D/A converter drift compensation setting	28
BOOL **E816_qDIP** (int ID, const char *szAxes, BOOL *pbValarray)	Ask if a digital pulse was detected since the last call of DIP? for the given axis	28
BOOL **E816_qERR** (int ID, int *pnError)	Get master unit error code	28
BOOL **E816_qHLP** (int *ID*, char* *szBuffer*, int *iBufferSize*)	Get online help list; currently no information is available	28
BOOL **E816_qI2C** (int ID, int *pnErrorCode, char *pcChannel)	Get status of I^2C bus	29
BOOL **E816_qIDN** (int ID, char *buffer, int maxlen)	Get master unit version information	29
BOOL **E816_qMAC** (int ID, char * *szName*, char * *szBuffer*, const int *maxlen*)	List the content of one macro or the names of all macros on the master E-816	29
BOOL **E816_qMOV** (int ID, const char* szAxes, double *pdValarray)	Read the last commanded position of the given axis	30
BOOL **E816_qMVT** (int ID, const char *szAxes, BOOL *pbValarray)	Get current “move triggered” mode of the given axis	30
BOOL **E816_qONT** (int ID, const char* szAxes, BOOL *pbOnTarget)	Get on-target status of the given axis	30
BOOL **E816_qOVF** (int ID, const char* szAxes, BOOL *pbOverflow)	Get overflow status of the given axis	30
BOOL **E816_qPOS** (int ID, const char* szAxes, double *pdValarray)	Read the actual position of the given axis	30
BOOL **E816_qSAI** (int ID, char *axes, const int maxlen)	Get names assigned to all connected (networked) axes	31
BOOL **E816_qSCH** (int ID, char *pcChannelName)	Get channel name (= axis identifier) of master unit	31
BOOL **E816_qSPA** (int ID, const char* szAxes, int *iCmdarray, double *dValarray)	Get specified parameter of the specified axis	31
BOOL **E816_qSSN** (int ID, char *szAxes, int *piValarray)	Get master unit serial number	32

Function	Short Description	Page
BOOL **E816_qSWT** (int ID, char cAxis, int nIndex, double *pdValue)	Get wave table data	32
BOOL **E816_qSVA** (int ID, const char* szAxes, double *pdValarray)	Read the last commanded piezo voltage of the given axis	32
BOOL **E816_qSVO** (int ID, char *szAxes, BOOL *pbValarray)	Get servo-ON/OFF status of the given axis	33
BOOL **E816_qVOL** (int ID, const char* szAxes, double *pdValarray)	Read the actual piezo voltage of the given axis	33
BOOL **E816_RST** (int ID)	Reset the master unit	33
BOOL **E816_SCH** (int ID, const char cChannelName)	Set channel name (= axis identifier) of master unit	33
BOOL **E816_SPA** (int ID, const char* szAxes, int *iCmdarray, double *dValarray)	Set specified parameter of the specified axis	34
BOOL **E816_SVA** (int ID, const char* szAxes, double *pdValarray)	Set the given axis to absolute piezo voltage	35
BOOL **E816_SVO** (int ID, char *szAxes, BOOL *pbValarray)	Set servo-ON/OFF status of the given axis	35
BOOL **E816_SVR** (int ID, const char* szAxes, double *pdValarray)	Change the given axis piezo voltage relative to current value	36
BOOL **E816_SWT** (int ID, const char cAxis, const int nIndex, const double dValue)	Set wave table data	36
BOOL **E816_WPA** (int ID, const char* szPassword)	Write all master unit parameters to master unit flash ROM	37
BOOL **E816_WTO** (int ID, const char cAxis, const int nNumber)	Set wave table output	37
BOOL **E816_WTOTimer** (int ID, const char cAxis, const int nNumber, int timer)	Set wave table output	38

8.2. Function Description

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.

This command only changes the setting in RAM; the new setting will be lost when the unit is powered down or reset (E816_RST()) unless the RAM settings are written to EEPROM with E816_WPA().

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 RS-232 communications 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.37*) 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.33*) after **E816_WPA**() (*p.37*).

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.

This command only changes the setting in RAM; the new setting will be lost when the unit is powered down or reset (E816_RST()) unless the RAM settings are written to EEPROM with E816_WPA().

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_DEL** (int *ID*, double *dmSeconds*)

Corresponding command: `DEL`

Delays the master unit for *dmSeconds* milliseconds, recommended for usage in macro operation (see E816_MAC functions).
Available with firmware revision 3.20 and newer.

Parameters:

ID ID of controller network
dmSeconds delay value in milliseconds

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **E816_IsRunningMacro** (int *ID*, BOOL * *pbRunningMacro*)

Corresponding command: `#8` (ASCII 8)

Check if a macro is currently running on the master unit.
Available with firmware revision 3.20 and newer.

Parameters:

ID ID of controller network
pbRunningMacro pointer to boolean to receive answer: **TRUE** if a macro is running, **FALSE** otherwise

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **E816_MAC_BEG** (int *ID*, const char * *szName*)

Corresponding command: `MAC BEG`

Put the DLL in macro recording mode (macro is recorded on the master unit). This function sets a flag in the library and effects the operation of other functions. Function will fail if already in recording mode. If successful, the commands that follow become part of the macro, so do not check error state unless FALSE is returned. End the recording with E816_MAC_END().

See "Working with E-816 Macros" and the MAC command description in the E-816 User manual for details.

Parameters:

ID ID of controller network
szName name under which macro will be stored in the controller

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_IN_MACRO_MODE if a macro is already being recorded

BOOL **E816_MAC_DEF** (int *ID*, const char * *szMacroName*)

Corresponding command: `MAC DEF`

Set macro with name *szName* as start-up macro on the master unit. This macro will be automatically executed with the next power-on or reboot of the controller. If *szName* is omitted, the current start-up macro selection is canceled. To find out what macros are available call E816_qMAC().

See "Working with E-816 Macros" and the MAC command description in the E-816 User manual for details.

Parameters:

ID ID of controller network
szMacroName name of the macro to be the start-up macro

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **E816_MAC_DEL** (int *ID*, const char * *szMacroName*)

Corresponding command: `MAC DEL`

Delete macro with name *szName* on the master unit. To find out what macros are available on the master unit call E816_qMAC().

See "Working with E-816 Macros" and the MAC command description in the E-816 User manual for details.

Parameters:

ID ID of controller network
szMacroName name of the macro to delete

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **E816_MAC_END** (int *ID*)

Corresponding command: `MAC END`

Take the DLL out of macro recording mode (macros was recorded on the master unit). This function resets a flag in the library and effects the operation of certain other functions. Function will fail if the DLL is not in recording mode.

See "Working with E-816 Macros" and the MAC command description in the E-816 User manual for details.

Parameters:

ID ID of controller network

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_NOT_IN_MACRO_MODE the controller was not recording a macro

BOOL **E816_MAC_NSTART** (int *ID*, const char * *szName*, int *nrRuns*)

Corresponding command: `MAC START`

Start macro with name *szName* on the master unit. The macro is repeated *nrRuns* times. To find out what macros are available on the master unit call E816_qMAC().

See "Working with E-816 Macros" and the MAC command description in the E-816 User manual for details.

Parameters:

ID ID of controller network
szName string with name of the macro to start
nrRuns nr of runs

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **E816_MAC_qDEF** (int *ID*, char * *szBuffer*, const int *iBufferSize*)

Corresponding command: `MAC DEF?`

Ask for the start-up macro on the master unit.

See "Working with E-816 Macros" and the MAC command description in the E-816 User manual for details.

Parameters:

ID ID of controller network
szBuffer buffer to receive the string read in from controller, contains the name of the start-up macro. If no start-up macro is defined, the response is an empty string with the terminating character.
iBufferSize size of *buffer*, must be given to avoid buffer overflow.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **E816_MAC_qFREE** (int *ID*, int* *pNrChars*)

Corresponding command: `MAC FREE?`

Ask for free space on the master unit.

See "Working with E-816 Macros" and the MAC command description in the E-816 User manual for details.

Parameters:

ID ID of controller network
pNrChars pointer to int for storing the number of free characters

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **E816_MAC_START** (int *ID*, const char * *szMacroName*)

Corresponding command: `MAC START`

Start macro with name *szName* on the master unit. To find out what macros are available on the master unit call E816_qMAC().

See "Working with E-816 Macros" and the MAC command description in the E-816 User manual for details.

Parameters:

ID ID of controller network
szMacroName string with name of the macro to start

Returns:

TRUE if successful, **FALSE** otherwise

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

Corresponding command: `MOV`

Move *szAxes* to absolute positions.

Before using this command, please make sure the E-816 is in computer-controlled mode and has servo-control mode set ON (with E816_SVO()). See “Modes of Operation“ in the E-816 User Manual for more information.

Move commands like E816_MOV() are not accepted when the wave table output is running (E816_WTO()) or when triggered motion is enabled (E816_MVT()).

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.

Before using this command, please make sure the E-816 is in computer-controlled mode and has servo-control mode set ON (with E816_SVO()). See “Modes of Operation“ in the E-816 User Manual for more information.

Move commands like E816_MVR() are not accepted when the wave table output is running (E816_WTO()) or when triggered motion is enabled (E816_MVT()).

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **E816_MVT** (int ID, const char * *szAxes*, const BOOL * *pbValarray*)

Corresponding command: `MVT`

Sets the “move triggered” mode ON or OFF. If this mode is enabled for an axis, every trigger pulse received causes a relative step. The step size is given by parameter 11 which can be set with E816_SPA().
Available with firmware revision 3.20 and newer.

Before you enable the “move triggered” mode, make sure the E-816 is in computer-controlled mode (see “Control Modes” in the E-816 User Manual). Furthermore, the piezo control electronics must be configured to accept trigger input, and a suitable trigger signal must be available (min. trigger pulse width = 5 µs; max. trigger frequency = 400 Hz). For details, see the MVT description in the E-816 User Manual and the manual of the piezo control electronics in which the E-816 is installed.

Triggered motion can not be enabled as long as the wave table output is running. When triggered motion is enabled, move commands (e.g. E816_SVA(), E816_MOV()) are not accepted and wave table output (E816_WTO()) can not be started.

The setting made with E816_MVT() is lost upon reset or when the device is powered down. Default setting is 0.

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_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.23*) 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_qDIP** (int ID, char * *szAxes*, BOOL * *pbValarray*)

Corresponding command: `DIP?`

Ask if a digital pulse was detected for the given axis since the last call of E816_qDIP().
After a call of this function, the E-816 resets the pulse flag.
Available with firmware revision 3.20 and newer.

For meaningful results, make sure that the piezo control electronics in which the E-816 is installed is configured properly. For details, see the DIP description in the E-816 User Manual and the manual of the piezo control electronics in which the E-816 is installed.

Parameters:

ID ID of controller network
szAxes string with axes
pbValarray array to be filled with the answer values for the specified axes, **TRUE** for "trigger detected", **FALSE** for "no trigger detected"

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*

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.17*) because this will also return the internal error state of the library.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **E816_qHLP** (int *ID*, char* *szBuffer*, *int iBufferSize*)

Corresponding **command:** `HLP?`

Reports device’s online help list for the master unit. This command is provided for compatibility reasons since the current E-816 firmware is not yet able to generate a help string.
Available with firmware revision 3.20 and newer.

Parameters:

ID ID of controller network
szBuffer buffer to receive the string read in from controller, lines are separated by '\n' ("line-feed")
iBufferSize size of *szBuffer*, must be given to avoid buffer overflow.

Returns:

TRUE if no error, FALSE otherwise

BOOL **E816_qI2C** (int ID, int * *pnErrorCode*, char * *pcChannel*)

Corresponding command: `I2C?`

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 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.

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_qMAC** (int ID, char * *szName*, char * *szBuffer*, const int *maxlen*)

Corresponding command: `MAC?`

Get macros available on the master unit, or list contents of a specific macro on the master unit. If *szName* is empty or **NULL**, all available macros are listed in *szBuffer*, separated with line-feed characters. Otherwise the content of the macro with name *szName* is listed, the single lines separated with by line-feed characters. If there are no macros stored or the requested macro is empty the answer will be "".

See "Working with E-816 Macros" and the MAC command description in the E-816 User manual for details.

Parameters:

ID ID of controller network
szName string with name of the macro to list
szBuffer buffer to receive the string read in from controller, lines are separated by line-feed characters
maxlen size of *buffer*, must be given to avoid 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 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_qMVT** (int ID, const char * *szAxes*, BOOL * *pbValarray*)

Corresponding command: `MVT?`

Reports the current “move triggered” mode setting of the given axis.
Available with firmware revision 3.20 and newer.

Parameters:

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

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 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.
Can only be sent to the master unit 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 (= axis identifier) of the master E-816. 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*. With firmware revisions 2.xx, E816_qSPA() is only available for the master unit, i.e. the values in *szAxes* must be the axis identifier of the master unit or “A”. With firmware revision 3.20 and newer, E816_qSPA() can also be used with slave units.

For each parameter you wish to query, you must respecify the axis in *szAxes* and a parameter ID in the corresponding element of *iCmdarray*. The parameter ID can have following values (see the E-816 User Manual for details):

- 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

- 11 for the step size used for triggered motion (see E816_MVT()), the value is interpreted as µm in closed-loop operation or as volts in open-loop operation (with firmware rev. 3.20 and newer)
- 12 for configuration of wave table operation, the value is bit-coded as follows (with firmware rev. 3.20 and newer):

	Bit 1 “TrigOnce”	Bit 0 “En”
Description	If set, every trigger pulse received starts one wave table output cycle, i.e. all points set by the last call of E816_WTO() are output once.	If set, the last saved status of E816_WTO() and E816_SVO() is recovered after power-on or reset
Default setting	0 = Normal wave trigger mode, i.e. one point is output per trigger pulse	0 = Status recovery disabled

Parameters:

ID ID of controller network
szAxes 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 1-12

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

Corresponding command: `SSN?`

Get serial number of the master unit.

Parameters:

ID ID of controller network
szAxes string with axis designator
piValarray array to be filled with the serial number of the master unit

Returns:

TRUE if successful, **FALSE** otherwise

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

Corresponding command: `SVA?`

Read the commanded piezo voltages for *szAxes*. (see also **E816_qVOL**() (*p.33*))

Parameters:

ID ID of controller network
szAxes string with 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 (256 points with firmware revision 3.20 and newer). 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. (0 to 255)
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*
Reports the last sent E816_SVO() settings of the given axis. Even if the E-816 is in analog mode, E816_qSVO() does not report the hardware settings for the servo mode. See E-816 User Manual for more information.

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 piezo voltages for *szAxes*. (see also **E816_qSVA**() (*p.32*))

Parameters:

ID 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.37*) without power-cycling the E-816. The master unit remains master after a reset, even if the communications 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 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 (axis identifier) of the E-816. E816_SCH() can only be used with the master unit, but the axis identifier set is to be used to address the unit irrespective of its current master / slave state.

Note that if the name is set to A with this command, the unit cannot be addressed as slave since commands with A as axis identifier will always be executed by the master unit.

This will only change the setting in the RAM. To store it in the EEPROM call **E816_WPA**() (*p.37*) afterwards.

Parameters:

ID ID of controller network
cChannelName the channel name of the device, can be any letter from A to Z; 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*. With firmware revisions 2.xx, E816_SPA() is only available for the master unit. With firmware revision 3.20 and newer it can also be used with slave units.

You must include an axis identifier in *szAxes* for each parameter you wish to set. Missing or illegal 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 (see the E-816 User Manual for details):

- 7 for KSen
- 8 for OSen
- 9 for Kpzt
- 10 for Opzt
- 11 for the step size used for triggered motion (see E816_MVT()), the value is interpreted as µm in closed-loop operation or as volts in open-loop operation (with firmware rev. 3.20 and newer)
- 12 for configuration of wave table operation, the value is bit-coded as follows (with firmware rev. 3.20 and newer):

	Bit 1 “TrigOnce”	Bit 0 “En”
Description	If set, every trigger pulse received starts one wave table output cycle, i.e. all points set by the last call of E816_WTO() are output once.	If set, the last saved status of E816_WTO(), E816_WTOTimer() and E816_SVO() is recovered after power-on or reset
Default setting	0 = Normal wave trigger mode, i.e. one point is output per trigger pulse	0 = Status recovery disabled

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, for example, KSen (ID=7) to 1.0 and OSen (ID=8) to 10.0 for the master axis, you must call `E816_SPA(id, "AA", {7, 8}, {1.0, 10.0})`

`szAxes = "AA"`	`cmd = {7, 8}`	`values = {1.0, 10.0}`
`szAxes[0] = 'A'`	`cmd[0] = 7`	`values[0] = 1.0`
`szAxes[1] = 'A'`	`cmd[1] = 8`	`values[1] = 10.0`

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
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 {7,8,9,10,11,12}

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

Corresponding command: SVA

Set piezo voltages for *szAxes* to absolute values.

Before using this command, please make sure the E-816 is in computer-controlled mode and has servo-control mode set OFF (with E816_SVO()). See “Modes of Operation“ in the E-816 User Manual for more information.

Move commands like E816_SVA() are not accepted when the wave table output is running (E816_WTO()) or when triggered motion is enabled (E816_MVT()).

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: SV0

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".

Only takes effect when the E-816 is in computer-controlled mode. See “Modes of Operation“ in the E-816 User Manual for more information. Note that in computer-controlled mode the servo mode selection made by the DIP switches or SERVO toggle switch on the piezo control electronics is ignored.

E816_WPA() saves the current E816_SVO() settings. But to make them the new power-on defaults, you must set parameter 12 with E816_SPA() to the corresponding value (see p. 34). With the default setting of parameter 12, the last saved E816_SVO() settings are not recovered on power-on or reset.

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 piezo voltages for *szAxes* relatively, i.e. increase current voltages by the specified values.

Before using this command, please make sure the E-816 is in computer-controlled mode and has servo-control mode set OFF (with E816_SVO()). See “Modes of Operation“ in the E-816 User Manual for more information.

Move commands like E816_SVR() are not accepted when the wave table output is running (E816_WTO()) or when triggered motion is enabled (E816_MVT()).

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 (256 points with firmware revision 3.20 and newer). With this command you can place a value in the table. The data is automatically stored in non-volatile memory and can be reused after next power-up.

The first time data is written to the wave table, it is recommended to define all points. Afterwards, it may be sufficient to define certain points.

Parameters:

ID ID of controller network
cAxis channel name of the axis
nIndex index for table entry, must be in 0 to 63 (0 to 255)
dValue new value for wave table entry, the value is interpreted as µm in closed-loop operation or as volts in open-loop operation

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **E816_WPA** (int ID, const char* *szPassword*)

Corresponding command: `WPA`

E816_WPA() only affects the master unit. The current values of parameters settable by E816_SPA(), E816_AVG(), E816_BDR(), E816_DCO() and E816_SCH() are written to nonvolatile memory (EEPROM), where they become the new power-on defaults. Furthermore, E816_WPA() saves macros, and is also required if an existing macro has been changed, was deleted or defined as start-up macro (see the E816_MAC functions).

Note that the volatile (RAM) value of E816_BDR() does not go into effect until after it is written to ROM and the system reset, so the RAM value may differ from the current operating value.

E816_WPA() also saves the current E816_WTO() and E816_SVO() settings. But to make them the new power-on defaults, you must set parameter 12 with E816_SPA() to the corresponding value (see p. 34). With the default setting of parameter 12, the last saved E816_WTO() and E816_SVO() settings are not recovered on power-on or reset.

If the current RAM values are incompatible, the system may malfunction. Be sure that you have entered the correct parameter settings before using E816_WPA().

Parameters:

ID ID of controller network
szPassword password needed to store the parameters. The password is “100”.

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. One wave-table point is output each time an external trigger signal is received.

Before starting wave table output, please make sure the E-816 is in computer-controlled mode. Furthermore, the piezo control electronics must be configured to accept trigger input, and a suitable trigger signal must be available (min. trigger pulse width = 5 µs; max. trigger frequency = 400 Hz). For details, see the E-816 User Manual and the manual of the piezo control electronics in which the E-816 is installed.

Wave table output can not be started as long as triggered motion is enabled (E816_MVT()). When wave table output is running, move commands (e.g. E816_SVA(), E816_MOV()) are not accepted and triggered motion (E816_MVT()) can not be enabled.

During wave-table output drift compensation (see E816_DCO()) is not carried out even if set to 1.

During wave-table output it is possible to change values in the wave table, with changes taking effect when the next point is output.

E816_WPA() saves the current E816_WTO() settings. But to make them the new power-on defaults, you must set parameter 12 with E816_SPA() to the corresponding value (see p. 34). With the default setting of parameter 12, the last saved E816_WTO() settings are not recovered on power-on or reset.

See “Working with the Wave Table” in the E-816 User Manual for more information.

Parameters:

ID ID of controller network
cAxis channel name of the axis
nNumber if 0, the output is stopped. With *nNumber* in 1 to 64 (1 to 256 with firmware rev. 3.20 and newer), the output is started from index 0 to (*nNumber-1)*.

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. Output of the points specified by *nNumber* will be started immediately and each point will be output for the amount of time specified by *timer* in milliseconds. Output will roll over from the point with index *nNumber*-1 to the point with index 0 and continue until stopped by E816_WTOTimer() or E816_WTO() with *nNumber* = 0.

Before starting wave table output, please make sure the E-816 is in computer-controlled mode. For details, see the E-816 User Manual.

Wave table output can not be started as long as triggered motion is enabled (E816_MVT()). When wave table output is running, move commands (e.g. E816_SVA(), E816_MOV()) are not accepted and triggered motion (E816_MVT()) can not be enabled.

During wave-table output drift compensation (see E816_DCO()) is not carried out even if set to 1.

During wave-table output it is possible to change values in the wave table, with changes taking effect when the next point is output.

E816_WPA() saves the current E816_WTOTimer() settings. But to make them the new power-on defaults, you must set parameter 12 with E816_SPA() to the corresponding value (see p. 34). With the default setting of parameter 12, the last saved E816_WTOTimer() settings are not recovered on power-on or reset.

See “Working with the Wave Table” in the E-816 User Manual for more information.

Parameters:

ID ID of controller network
cAxis channel name of the axis
nNumber if 0, the output is stopped. With *nNumber* in 1 to 64 (1 to 256 with firmware rev. 3.20 and newer), the output continues indefinitely. Stop with an **E816_WTOTimer**() or **E816_WTO** () with *nNumber* = 0
timer time in milliseconds, each point will be output for that amount of time

Returns:

TRUE if successful, **FALSE** otherwise

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 or cannot be initialized
48	PI_CNTR_OPM_COM_ERROR	OPM (Optical Power Meter) Communication Error
49	PI_CNTR_MOVE_TO_LIMIT_SWITCH_FAILED	Move to limit switch failed
50	PI_CNTR_REF_WITH_REF_DISABLED	Attempt to reference axis with referencing disabled
51	PI_CNTR_AXIS_UNDER_JOYSTICK_CONTROL	Selected axis is controlled by joystick
52	PI_CNTR_COMMUNICATION_ERROR	Controller detected communication error
53	PI_CNTR_DYNAMIC_MOVE_IN_PROCESS	MOV! motion still in progress
54	PI_CNTR_UNKNOWN_PARAMETER	Unknown parameter
55	PI_CNTR_NO_REP_RECORDED	No commands were recorded with REP
56	PI_CNTR_INVALID_PASSWORD	Password invalid
57	PI_CNTR_INVALID_RECORDER_CHAN	Data Record Table does not exist
58	PI_CNTR_INVALID_RECORDER_SRC_OPT	Source does not exist; number too low or too high
59	PI_CNTR_INVALID_RECORDER_SRC_CHAN	Source Record Table number too low or too high
60	PI_CNTR_PARAM_PROTECTION	Protected Param: current Command Level (CCL) too low
61	PI_CNTR_AUTOZERO_RUNNING	Command execution not possible while Autozero is running
62	PI_CNTR_NO_LINEAR_AXIS	Autozero requires at least one linear axis
63	PI_CNTR_INIT_RUNNING	Initialization still in progress
64	PI_CNTR_READ_ONLY_PARAMETER	Parameter is read-only
65	PI_CNTR_PAM_NOT_FOUND	Parameter not found in non-volatile memory
66	PI_CNTR_VOL_OUT_OF_LIMITS	Voltage out of limits

67	PI_CNTR_WAVE_TOO_LARGE	Not enough memory available for requested wave curve
68	PI_CNTR_NOT_ENOUGH_DDL_MEMORY	Not enough memory available for DDL table; DDL can not be started
69	PI_CNTR_DDL_TIME_DELAY_TOO_LARGE	Time delay larger than DDL table; DDL can not be started
70	PI_CNTR_DIFFERENT_ARRAY_LENGTH	The requested arrays have different lengths; query them separately
71	PI_CNTR_GEN_SINGLE_MODE_RESTART	Attempt to restart the generator while it is running in single step mode
72	PI_CNTR_ANALOG_TARGET_ACTIVE	Motion commands and wave generator activation are not allowed when analog target is active
73	PI_CNTR_WAVE_GENERATOR_ACTIVE	Motion commands are not allowed when wave generator is active
74	PI_CNTR_AUTOZERO_DISABLED	No sensor channel or no piezo channel connected to selected axis (sensor and piezo matrix)
75	PI_CNTR_NO_WAVE_SELECTED	Generator started (WGO) without having selected a wave table (WSL).
76	PI_CNTR_IF_BUFFER_OVERRUN	Interface buffer did overrun and command couldn't be received correctly
77	PI_CNTR_NOT_ENOUGH_RECORDED_DATA	Data Record Table does not hold enough recorded data
78	PI_CNTR_TABLE_DEACTIVATED	Data Record Table is not configured for recording
79	PI_CNTR_OPENLOOP_VALUE_SET_WHEN_SERVO_ON	Open-loop commands (SVA, SVR) are not allowed when servo is on
80	PI_CNTR_RAM_ERROR	Hardware error affecting RAM
81	PI_CNTR_MACRO_UNKNOWN_COMMAND	Not macro command
82	PI_CNTR_MACRO_PC_ERROR	Macro counter out of range
83	PI_CNTR_JOYSTICK_ACTIVE	Joystick is active
84	PI_CNTR_MOTOR_IS_OFF	Motor is off
85	PI_CNTR_ONLY_IN_MACRO	Macro-only command
86	PI_CNTR_JOYSTICK_UNKNOWN_AXIS	Invalid joystick axis
87	PI_CNTR_JOYSTICK_UNKNOWN_ID	Joystick unknown
88	PI_CNTR_REF_MODE_IS_ON	Move without referenced stage

89	PI_CNTR_NOT_ALLOWED_IN_CURRENT_MOTION_MODE	Command not allowed in current motion mode
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
230	PI_CNTR_INVALID_HANDLE	Invalid handle

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

601	PI_CNTR_NOT_ENOUGH_MEMORY	not enough memory
602	PI_CNTR_HW_VOLTAGE_ERROR	hardware voltage error
603	PI_CNTR_HW_TEMPERATURE_ERROR	hardware temperature out of range
1000	PI_CNTR_TOO_MANY_NESTED_MACROS	Too many nested macros
1001	PI_CNTR_MACRO_ALREADY_DEFINED	Macro already defined
1002	PI_CNTR_NO_MACRO_RECORDING	Macro recording not activated
1003	PI_CNTR_INVALID_MAC_PARAM	Invalid parameter for MAC
1004	PI_CNTR_RESERVED_1004	PI internal error code 1004
1005	PI_CNTR_CONTROLLER_BUSY	Controller is busy with some lengthy operation (e.g. reference move, fast scan algorithm)
2000	PI_CNTR_ALREADY_HAS_SERIAL_NUMBER	Controller already has a serial number
4000	PI_CNTR_SECTOR_ERASE_FAILED	Sector erase failed
4001	PI_CNTR_FLASH_PROGRAM_FAILED	Flash program failed
4002	PI_CNTR_FLASH_READ_FAILED	Flash read failed
4003	PI_CNTR_HW_MATCHCODE_ERROR	HW match code missing/invalid
4004	PI_CNTR_FW_MATCHCODE_ERROR	FW match code missing/invalid
4005	PI_CNTR_HW_VERSION_ERROR	HW version missing/invalid
4006	PI_CNTR_FW_VERSION_ERROR	FW version missing/invalid
4007	PI_CNTR_FW_UPDATE_ERROR	FW update failed

Interface Errors

0	COM_NO_ERROR	No error occurred during function call
-1	COM_ERROR	Error during com operation (could not be specified)
-2	SEND_ERROR	Error while sending data

-3	REC_ERROR	Error while receiving data
-4	NOT_CONNECTED_ERROR	Not connected (no port with given ID open)
-5	COM_BUFFER_OVERFLOW	Buffer overflow
-6	CONNECTION_FAILED	Error while opening port
-7	COM_TIMEOUT	Timeout error
-8	COM_MULTILINE_RESPONSE	There are more lines waiting in buffer
-9	COM_INVALID_ID	There is no interface or DLL handle with the given ID
-10	COM_NOTIFY_EVENT_ERROR	Event/message for notification could not be opened
-11	COM_NOT_IMPLEMENTED	Function not supported by this interface type
-12	COM_ECHO_ERROR	Error while sending "echoed" data
-13	COM_GPIB_EDVR	IEEE488: System error
-14	COM_GPIB_ECIC	IEEE488: Function requires GPIB board to be CIC
-15	COM_GPIB_ENOL	IEEE488: Write function detected no listeners
-16	COM_GPIB_EADR	IEEE488: Interface board not addressed correctly
-17	COM_GPIB_EARG	IEEE488: Invalid argument to function call
-18	COM_GPIB_ESAC	IEEE488: Function requires GPIB board to be SAC
-19	COM_GPIB_EABO	IEEE488: I/O operation aborted
-20	COM_GPIB_ENEB	IEEE488: Interface board not found
-21	COM_GPIB_EDMA	IEEE488: Error performing DMA
-22	COM_GPIB_EOIP	IEEE488: I/O operation started before previous operation completed
-23	COM_GPIB_ECAP	IEEE488: No capability for intended operation
-24	COM_GPIB_EFSO	IEEE488: File system operation error
-25	COM_GPIB_EBUS	IEEE488: Command error during device call

-26	COM_GPIB_ESTB	IEEE488: Serial poll-status byte lost
-27	COM_GPIB_ESRQ	IEEE488: SRQ remains asserted
-28	COM_GPIB_ETAB	IEEE488: Return buffer full
-29	COM_GPIB_ELCK	IEEE488: Address or board locked
-30	COM_RS_INVALID_DATA_BITS	RS-232: 5 data bits with 2 stop bits is an invalid combination, as is 6, 7, or 8 data bits with 1.5 stop bits
-31	COM_ERROR_RS_SETTINGS	RS-232: Error configuring the COM port
-32	COM_INTERNAL_RESOURCES_ERROR	Error dealing with internal system resources (events, threads, ...)
-33	COM_DLL_FUNC_ERROR	A DLL or one of the required functions could not be loaded
-34	COM_FTDIUSB_INVALID_HANDLE	FTDIUSB: invalid handle
-35	COM_FTDIUSB_DEVICE_NOT_FOUND	FTDIUSB: device not found
-36	COM_FTDIUSB_DEVICE_NOT_OPENED	FTDIUSB: device not opened
-37	COM_FTDIUSB_IO_ERROR	FTDIUSB: IO error
-38	COM_FTDIUSB_INSUFFICIENT_RESOURCES	FTDIUSB: insufficient resources
-39	COM_FTDIUSB_INVALID_PARAMETER	FTDIUSB: invalid parameter
-40	COM_FTDIUSB_INVALID_BAUD_RATE	FTDIUSB: invalid baud rate
-41	COM_FTDIUSB_DEVICE_NOT_OPENED_FOR_ERASE	FTDIUSB: device not opened for erase
-42	COM_FTDIUSB_DEVICE_NOT_OPENED_FOR_WRITE	FTDIUSB: device not opened for write
-43	COM_FTDIUSB_FAILED_TO_WRITE_DEVICE	FTDIUSB: failed to write device
-44	COM_FTDIUSB_EEPROM_READ_FAILED	FTDIUSB: EEPROM read failed
-45	COM_FTDIUSB_EEPROM_WRITE_FAILED	FTDIUSB: EEPROM write failed
-46	COM_FTDIUSB_EEPROM_ERASE_FAILED	FTDIUSB: EEPROM erase failed
-47	COM_FTDIUSB_EEPROM_NOT_PRESENT	FTDIUSB: EEPROM not present
-48	COM_FTDIUSB_EEPROM_NOT_PROGRAMMED	FTDIUSB: EEPROM not programmed

-49	COM_FTDIUSB_INVALID_ARGS	FTDIUSB: invalid arguments
-50	COM_FTDIUSB_NOT_SUPPORTED	FTDIUSB: not supported
-51	COM_FTDIUSB_OTHER_ERROR	FTDIUSB: other error
-52	COM_PORT_ALREADY_OPEN	Error while opening the COM port: was already open
-53	COM_PORT_CHECKSUM_ERROR	Checksum error in received data from COM port
-54	COM_SOCKET_NOT_READY	Socket not ready, you should call the function again
-55	COM_SOCKET_PORT_IN_USE	Port is used by another socket
-56	COM_SOCKET_NOT_CONNECTED	Socket not connected (or not valid)
-57	COM_SOCKET_TERMINATED	Connection terminated (by peer)
-58	COM_SOCKET_NO_RESPONSE	Can't connect to peer
-59	COM_SOCKET_INTERRUPTED	Operation was interrupted by a nonblocked signal
-60	COM_PCI_INVALID_ID	No device with this ID is present
-61	COM_PCI_ACCESS_DENIED	Driver could not be opened (on Vista: run as administrator!)

DLL Errors

-1001	PI_UNKNOWN_AXIS_IDENTIFIER	Unknown axis identifier
-1002	PI_NR_NAV_OUT_OF_RANGE	Number for NAV out of range--must be in [1,10000]
-1003	PI_INVALID_SGA	Invalid value for SGA--must be one of 1, 10, 100, 1000
-1004	PI_UNEXPECTED_RESPONSE	Controller sent unexpected response
-1005	PI_NO_MANUAL_PAD	No manual control pad installed, calls to SMA and related commands are not allowed
-1006	PI_INVALID_MANUAL_PAD_KNOB	Invalid number for manual control pad knob
-1007	PI_INVALID_MANUAL_PAD_AXIS	Axis not currently controlled by a manual control pad
-1008	PI_CONTROLLER_BUSY	Controller is busy with some lengthy operation (e.g. reference move, fast scan algorithm)

-1009	PI_THREAD_ERROR	Internal error--could not start thread
-1010	PI_IN_MACRO_MODE	Controller is (already) in macro mode--command not valid in macro mode
-1011	PI_NOT_IN_MACRO_MODE	Controller not in macro mode--command not valid unless macro mode active
-1012	PI_MACRO_FILE_ERROR	Could not open file to write or read macro
-1013	PI_NO_MACRO_OR_EMPTY	No macro with given name on controller, or macro is empty
-1014	PI_MACRO_EDITOR_ERROR	Internal error in macro editor
-1015	PI_INVALID_ARGUMENT	One or more arguments given to function is invalid (empty string, index out of range, ...)
-1016	PI_AXIS_ALREADY_EXISTS	Axis identifier is already in use by a connected stage
-1017	PI_INVALID_AXIS_IDENTIFIER	Invalid axis identifier
-1018	PI_COM_ARRAY_ERROR	Could not access array data in COM server
-1019	PI_COM_ARRAY_RANGE_ERROR	Range of array does not fit the number of parameters
-1020	PI_INVALID_SPA_CMD_ID	Invalid parameter ID given to SPA or SPA?
-1021	PI_NR_AVG_OUT_OF_RANGE	Number for AVG out of range--must be >0
-1022	PI_WAV_SAMPLES_OUT_OF_RANGE	Incorrect number of samples given to WAV
-1023	PI_WAV_FAILED	Generation of wave failed
-1024	PI_MOTION_ERROR	Motion error while axis in motion, call CLR to resume operation
-1025	PI_RUNNING_MACRO	Controller is (already) running a macro
-1026	PI_PZT_CONFIG_FAILED	Configuration of PZT stage or amplifier failed
-1027	PI_PZT_CONFIG_INVALID_PARAMS	Current settings are not valid for desired configuration
-1028	PI_UNKNOWN_CHANNEL_IDENTIFIER	Unknown channel identifier
-1029	PI_WAVE_PARAM_FILE_ERROR	Error while reading/writing wave generator parameter file
-1030	PI_UNKNOWN_WAVE_SET	Could not find description of wave form. Maybe WG.INI is missing?

-1031	PI_WAVE_EDITOR_FUNC_NOT_LOADED	The WGWaveEditor DLL function was not found at startup
-1032	PI_USER_CANCELLED	The user cancelled a dialog
-1033	PI_C844_ERROR	Error from C-844 Controller
-1034	PI_DLL_NOT_LOADED	DLL necessary to call function not loaded, or function not found in DLL
-1035	PI_PARAMETER_FILE_PROTECTED	The open parameter file is protected and cannot be edited
-1036	PI_NO_PARAMETER_FILE_OPENED	There is no parameter file open
-1037	PI_STAGE_DOES_NOT_EXIST	Selected stage does not exist
-1038	PI_PARAMETER_FILE_ALREADY_OPENED	There is already a parameter file open. Close it before opening a new file
-1039	PI_PARAMETER_FILE_OPEN_ERROR	Could not open parameter file
-1040	PI_INVALID_CONTROLLER_VERSION	The version of the connected controller is invalid
-1041	PI_PARAM_SET_ERROR	Parameter could not be set with SPA--parameter not defined for this controller!
-1042	PI_NUMBER_OF_POSSIBLE_WAVES_EXCEEDED	The maximum number of wave definitions has been exceeded
-1043	PI_NUMBER_OF_POSSIBLE_GENERATORS_EXCEEDED	The maximum number of wave generators has been exceeded
-1044	PI_NO_WAVE_FOR_AXIS_DEFINED	No wave defined for specified axis
-1045	PI_CANT_STOP_OR_START_WAV	Wave output to axis already stopped/started
-1046	PI_REFERENCE_ERROR	Not all axes could be referenced
-1047	PI_REQUIRED_WAVE_NOT_FOUND	Could not find parameter set required by frequency relation
-1048	PI_INVALID_SPP_CMD_ID	Command ID given to SPP or SPP? is not valid
-1049	PI_STAGE_NAME_ISNT_UNIQUE	A stage name given to CST is not unique
-1050	PI_FILE_TRANSFER_BEGIN_MISSING	A uuencoded file transferred did not start with "begin" followed by the proper filename
-1051	PI_FILE_TRANSFER_ERROR_TEMP_FILE	Could not create/read file on host PC
-1052	PI_FILE_TRANSFER_CRC_ERROR	Checksum error when transferring a file to/from the controller

-1053	PI_COULDNT_FIND_PISTAGES_DAT	The 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