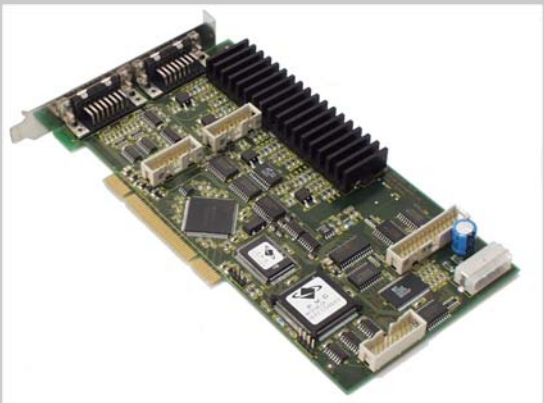
Piezo · Nano · Positioning

MS112E Software Manual

C-843 GCS DLL

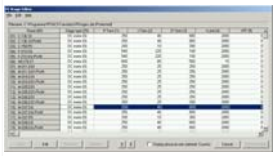
Windows Library Reference

Release: 4.0.0 Date: 2009-11-20



This document describes software for use with the following products:

- C-843.21
  Motor Controller Card (PCI), 2 motor axes
- C-843.41
  Motor Controller Card (PCI), 4 motor axes



Auf der Römerstr. 1 · 76228 Karlsruhe, Germany
Tel. +49-721-4846-0 · Fax: +49-721-4846-299
info@pi.ws · www.pi.ws

Moving the NanoWorld | www.pi.ws

First printing 2009-11-20
Document Number MS112E, Bro, Release 4.0.0
C-843_GCS_DLL_MS112E.doc

Table of Contents

0. Manufacturer Declarations

0.1. Scope of This Manual

This manual describes the function calls in the C-843 GCS DLL.

The contents of the manual is furnished for informational use only, is subject to change without notice and should not be construed as a commitment by Physik Instrumente (PI).

Physik Instrumente (PI) assumes no responsibility or liability for any errors or inaccuracies that may appear in this manual.

0.2. Disclaimer

The software described in this manual is distributed “as is”. Physik Instrumente (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.

Physik Instrumente (PI) expressly disclaims any and all warranties including any implied warranty of merchantability and fitness for a particular purpose. Physik Instrumente (PI) does not warrant, guarantee, or make any obligations regarding the use or the results of the use of this software, in terms of correctness, accuracy, reliability or otherwise, and does not warrant that operation of the software will be uninterrupted or error-free. The user agrees to use this software on his own responsibility.

1. Introduction

This manual describes the C-843 GCS DLL which makes all relevant commands of the PI General Command Set available for the C-843. The PI General Command Set (GCS) is supported by a wide range of PI systems. This command set is well-suited for positioning tasks with one or more axes and is independent of the specific hardware (controller or attached stages).

The commands of the motion processors command set (also referred to as "QMC" command set) can be sent via the C-843 GCS DLL as well so that its complete functionality is available to the programmer. The complete documentation provided by the motion processors manufacturer is included on the C-843 CD. Note that because of their complexity, successful use of the motion processor commands requires extensive experience.

The hardware and software installation procedures for the C-843 are described in the C-843 User Manual (MS77E). Software tools which might be mentioned in this document are described in their own manuals. All documents are available as PDF files on the C-843 CD. Updated releases are available for download at www.pi.ws or via e-mail: contact your Physik Instrumente Sales Engineer or write info@pi.ws.

List of manuals related to this document:

C-843_User_MS77E	User Manual for C-843 DC-Servo-Motor Controller board
C-843_GCS_Commands_SM149E	Manual for GCS Library (Windows and Linux Versions)
C843_GCS_LabVIEW_MS89E	Manual for GCS LabVIEW Drivers
PIMikroMoveUserManual_SM148E	Manual for PIMikroMove™ Host Software
PiStageEditor_SM144E	Manual for PIStageEditor Tool for Stage Database Handling
GCSData_User_SM146E	GCS Data Format Description
A000T0014_100_UserProfileModeSoftware	Technical Note for User Profile Mode and Profile Generator DLL

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

1.1. Conversion of Units

The GCS system uses physical units of measure. It provides a conversion factor to convert hardware-dependent units (encoder counts) into mm or degrees, as appropriate (see **C843_SPA** () and **C843_qSPA** (), parameters 0xE and 0xF). Defaults for the values are generally taken from a database of stages that can be connected. An additional scale factor can be applied, making a second physical unit (working unit) available without overwriting the conversion factor for the first (see the **C843_DFF**(), **C843_qDFF**() function calls).

1.2. Rounding Considerations

When converting move commands in (working) units to the hardware-dependent units required by the motion control layers, rounding errors can occur. The GCS software is so designed, that a relative move of x working units will always result in a relative move of the same number of hardware units. Because of rounding errors, this means, for example, that 2 relative

moves of x working units may differ slightly from one relative move of 2x. When making large numbers of relative moves, especially when moving back and forth, either intersperse absolute moves, or make sure that each relative move in one direction is matched by a relative move of the same size in the other direction.

Examples (assuming 5 hardware units = 33 x 10-6 physical units):
Relative moves smaller than 0.000003 physical units cause move of 0 hardware units.
Relative moves of 0.000004 to 0.000009 physical units cause move of 1 hardware unit.
Relative moves of 0.000010 to 0.000016 physical units cause move of 2 hardware units.
Relative moves of 0.000017 to 0.000023 physical units cause move of 3 hardware units.
Relative moves of 0.000024 to 0.000029 physical units cause move of 4 hardware units.

Hence:
2 moves of 10 x 10-6 physical units followed by 1 move of 20 x 10-6 in the other direction cause a net motion of 1 hardware unit forward.

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

5000 moves of 2 x 10-6 result in no motion.

1.3. Items to be Commanded

The identifiers listed below are used to address the appropriate items with GCS commands:

- **Axes**:

  The identifiers are 1 and 2 (with C-843.21) or 1 to 4 (with C-843.41) by default.
  The default identifiers can be changed using **C843_SAI**(). The new identifiers must then be used with all szAxes arguments, but are lost when the C-843 board is disconnected in the software.

- **Digital output lines**:

  A to H for the lines concerned by **C843_DIO**(). These output lines can be brought out of the PC housing using the sub-D 25f connector of the adapter bracket included with C-843. The corresponding ribbon cable connects to the 26-pin IDC connector (J5) on the C-843 (see C-843 User Manual for details).

  1 and 2 (with C-843.21) or 1 to 4 (with C-843.41) for the lines concerned by **C843_CTO**() and **C843_TRO**(). These output lines can be brought out of the PC housing using the sub-D 15m connector of the adapter bracket included with C-843. The corresponding ribbon cable connects to the All-axes 16-pin IDC connector of the C-843 board (J8; see C-843 User Manual for details). Note that older revisions of the adapter bracket do not provide the sub-D 15m connector. If required, contact your PI sales engineer or write info@pi.ws to obtain a new version.

  The identifiers of the digital output lines can not be changed.

- **Digital input lines**:

  A to H for the lines concerned by **C843_qDIO**(). These input lines are located on the same 26-pin IDC connector (J5) like the digital output

lines A to H, see above for how to bring them out of the PC housing using a bracket.

1 and 2 (with C-843.21) or 1 to 4 (with C-843.41) for the lines which can be used by **C843_FED**(). These input lines are located on the same 16-pin IDC connector (J8) like the digital output lines 1 to 2 (or 4), see above for how to bring them out of the PC housing using a bracket.

The identifiers of the digital input lines can not be changed.

- **Data recorder tables** (memory tables for recorded data):

  1 to 4; the identifiers can not be changed. See "Data Recording" in the C-843 GCS Commands manual (SM149E) for more information.

- **Joystick**:

  Each joystick connected to the host PC is identified by a joystick device ID, and each of its axes is identified by a joystick axis ID. Both joystick device IDs and joystick axis IDs start with 1 and can not be changed. See "Joystick Control" in the C-843 GCS Commands manual (SM149E) for more information.

- **Clusters and Blocks for User Profile Mode**:

  Using the User Profile Mode commands (U*xx*), you can perform motion by processing Datasets in the specified Cluster(s). A Cluster consists of space for a specified number of Datasets (a Dataset specifies a point on a 1-D trajectory; the values it contains are used for trajectory interpolation—time and position are always required, while velocity, acceleration and jerk are optionally required). Data can only be introduced into a Cluster using **C843_UPA**(), which swaps Datasets into a Cluster from a Block, which in turn was filled using **C843_UPD**().

  Possible cluster IDs are A to G, the maximum number of Blocks that can be assigned to a Cluster is 32.

1.4. Program Sequence

After all required files have been installed (the hardware and software installation procedures for the C-843 are described in the C-843 User Manual (MS77E)), write a program that performs the following steps:

1. Call C843_Connect() (p.19) to open a connection to the board
2. Call C843_CST() (p.19) to determine which stages are connected to the four/two axes of the C-843 board.
3. Call C843_INI() (p.30) to initialize the motion processor for the axes (switches also the servo on).
4. Run one of the referencing functions—C843_REF() (p.51), C843_MNL() (p.33), C843_MPL() (p.34), C843_FNL(), C843_FPL(), C843_FRF(), depending on what stage types are connected—to be able to perform absolute moves or access abolute position information with functions like C843_MOV() (p.34) or C843_qPOS() (p. 44).

5. Make a few test moves with C843_MOV() (p.34) so that you can verify your program's operation.

2. General Information About PI DLLs

The information below is valid for the DLL described in this manual as well as for the DLLs for many other PI products.

2.1. Threads

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

2.2. DLL Handling

To get access to and use the DLL functions, the library must be included in your software project. There are a number of techniques supported by the windows operating system and supplied by the different development systems. The following sections describe the methods which are most commonly used. For detailed information, consult the relevant documentation of the development environment being used. (It is possible to use the C843_GCS_DLL.DLL in Delphi projects. Please see `http://www.drbob42.com/delphi/headconv.htm` for a detailed description of the steps necessary.)

2.2.1. Using a Static Import Library

The `C843_GCS_DLL.DLL` module is accompanied by the `C843_GCS_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 function 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 declaration also specify that these functions are be called like standard Win-API functions. That means the VC++ compiler needs to see a WINAPI or __stdcall modifier in the declaration.
- Add the static import library to the program project. This is needed for the linker and tells it that the functions are located in a DLL and that they are to be linked dynamically during program startup.

2.2.2. 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 in which the DLL functions must be 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 indicating that they are coming from a C environment. The VC++ compiler needs an extern "C" modifier. The declaration also specify that these functions

are be called like standard Win-API functions. That means the VC++ compiler needs to see 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
   C843_GCS_DLL.C843_IsConnected
```

2.2.3. Using Windows API Functions

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

- Use a header or source file in which local or global pointers of a type appropriate for pointing to a function entry point are defined. This type could be defined in a typedef expression. In the following example, the type FP_C843_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_C843_IsConnected)( int );
FP_C843_IsConnected pC843_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("C843_GCS_DLL.DLL");
```

- Call the Win32-API GetProcAddress() Win32-API 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:

```
pC843_IsConnected  = (FP_C843_IsConnected)GetProcAddress(hPI_Dll,"C843_IsConnected");
if (pC843_IsConnected == NULL)
{
    // do something, for example
    return FALSE;
}
BOOL bResult = (*pC843_IsConnected)(1); // call C843_IsConnected(1)
```

2.3. Function Calls

Almost all functions will return a boolean value (type `BOOL`, see "**Types Used in PI Software,** p.11)"). If the function succeeded the return value is **TRUE**, otherwise it is **FALSE**. To find out what went wrong, call **C843_GetError**() (p.27) and look up the value returned in "**Error Codes** (p.61)". The first argument to most function calls is the ID of the selected controller.

2.3.1. Item Identifiers

Many commands accept one ore more axis identifiers. If no axes are specified (either by giving an empty string or a **NULL** pointer) some commands will address all connected axes. The same model is used for the other items listed in Section 1.3 on p. 7.

2.3.2. Arguments for the Items

The arguments for the axes (or other items to be commanded) are stored in an array passed to the function. The argument for the first axis in the axis

string is stored in `array[0]`, for the second axis in `array[1]`, and so on. So if you call `C843_qPOS("123", double pos[3])`, the position for '1' is in `pos[0]`, for '2' in `pos[1]` and for '3' in `pos[2]`. If you call `C843_MOV("12", double pos[2])` the target position for '1' is in `pos[0]` and for '2' in `pos[1]`.
Each axis identifier is sent only once. Only the **last** occurrence of an axis identifier is actually sent with its argument to the controller. So if you call `C843_MOV("112", pos[3])` with `pos[3] = { 1.0, 2.0, 3.0 }`, '1' will move to 2.0 and '2' to 3.0. If you then call `C843_qPOS("112", pos[3])`, `pos[0]` and `pos[1]` will contain 2.0 as the position of '1'.
(See **C843_MOV**() (p.34) and **C843_qPOS**() (p.44))
See **Types Used in PI Software** (p.11) for a description of used types for arguments.

2.4. Types Used in PI Software

2.4.1. Boolean Values

The library will use the convention used in Microsoft's C++ for boolean values. If your compiler does not support this, 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
```

2.4.2. NULL Pointers

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

```
#define NULL 0
```

2.4.3. C-Strings

The library uses the C convention to handle strings t. Strings are stored as arrays of `char` 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`".

3. C-843 GCS DLL Function Groups

In the following sections the GCS DLL functions are grouped by type. Detailed descriptions are in the Function Reference Section, starting on p. 18.

- "Communication Functions" (p. 12) shows how to initiate communication with a C-843 controller.
- "Functions for Initialization of the C-843 GCS DLL" (p. 12) describes the steps necessary at startup of the library.
- Functions for GCS Commands**,** (p. 13) describes the functions giving access to the C-843 GCS commands (p. 13).
- Functions for Accessing QMC Commands (p. 16) describes the functions giving access to the C-843 QMC commands (command set of the motion processor).
- Functions for User-Defined Stages (p. 17) describes the functions to add, edit and remove user-defined stages (parameter sets).

3.1. Communication Functions

To use the DLL and communicate with a C-843 controller the program must first initialize the controller with the "open" function **C843_Connect**() (p.19). To allow the handling of multiple controllers, the open function returns a non-negative ID. 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 their first argument. **C843_CloseConnection**() (p.19) will close the connection to the specified controller and free its system resources. The communications functions are listed below:

BOOL C843_FUNC_DECL **C843_ListPCI**(char* *szIDList*,long *maxlen*)
long C843_FUNC_DECL **C843_Connect**(long *iBoardNumber*)
BOOL C843_FUNC_DECL **C843_IsConnected**(long *iID*)
void C843_FUNC_DECL **C843_CloseConnection**(long *iID*)
long C843_FUNC_DECL **C843_GetError**(long *iID*)
BOOL C843_FUNC_DECL **C843_SetErrorCheck**(long *iID*, BOOL *bErrorCheck*)
BOOL C843_FUNC_DECL **C843_TranslateError**(long *errNr*, char* *szBuffer*, long *maxlen*)

3.2. Functions for Initialization of the C-843 GCS DLL

The C-843 GCS library cannot determine which stages are connected and must be configured at startup.
Call **C843_CST**() (p.19) to setup the connected stages. Afterwards call **C843_INI()** (p.30) to initialize the motion processor for the axes (switches also the servo on). With **C843_qCST**() (p.37) you can find out which stages are currently configured. **C843_qCST**() (p.37) will always return 2 or 4 axes, depending on the hardware version (2- or 4-axis version), axes to which no stage was connected are assigned to "NOSTAGE". **C843_qSAI**() (p.45) will always return only the identifiers of axes which are not set to "NOSTAGE".
Call **C843_qVST**() (p.51) to find out which stages the library already knows about (the content of the stage databases).

BOOL C843_FUNC_DECL **C843_CST** (long *iID*, const char* *szAxes*, const char * *names*)
BOOL C843_FUNC_DECL **C843_qCST** (long *iID*, const char* *szAxes*, char * *names*, long *maxlen*)
BOOL C843_FUNC_DECL **C843_INI** (long *iID*, const char* *szAxes*)
BOOL C843_FUNC_DECL **C843_SAI** (long *iID*, const char* *szOldAxes*, const char* *szNewAxes*)
BOOL C843_FUNC_DECL **C843_qSAI** (long *iID*, char **axes*, long *maxlen*)
BOOL C843_FUNC_DECL **C843_qVST** (long *iID*, char **buffer*, long *maxlen*)

The following example shows how to configure a C-843 and, except for the calls to printf(), is a typical initialization of the C-843 library.

```
char stages[1024];
char axes[10];
int ID;
// connect to the C-843
ID = C843_Connect(1);
if (ID<0)
    return FALSE;

// nothing is configured
if (!C843_qCST(ID, "1234", stages, 1023))
    return FALSE;
// the output should be "1=NOSTAGE\n2=NOSTAGE\n3=NOSTAGE\n4=NOSTAGE\n"
printf("qCST() returned \"%s\"", stages);

if (!C843_qSAI(ID, axes, 9))
    return FALSE;
// the output should be "" - no configured axes
printf("qSAI() returned \"%s\"", axes);

// we want to connect two M-111.1DG to channels 1 and 2
// and a M-112.2DG to channel four
sprintf(stages, "M-111.1DG\nM-111.1DG\nM-111.1DG");
if (!C843_CST(ID, "124", stages))
    return FALSE;

if (!C843_qSAI(ID, axes, 9))
    return FALSE;
// the output should be "124" - the new configured axes
printf("qSAI() returned \"%s\"", axes);

if (!C843_qCST(ID, "1234", stages, 1023))
    return FALSE;
// the output should be "1=M-111.1DG\n2=M-111.1DG\n3=NOSTAGE\n4=M-112.2DG\n"
printf("qCST() returned \"%s\"", stages);

// call INI for all axes
// "" as axes string will address all configured axes
if (!C843_INI(ID, ""))
    return FALSE;
```

3.3. Functions for GCS Commands

The functions listed here provide access to the embedded commands of the C-843 and provide some shortcuts to make the work with C-843 easier. For the GCS commands belonging to the appropriate functions see the alphabetical function reference beginning on p. 18. A list of GCS commands is also available in the C-843 GCS Commands manual, which is included on the C-843 CD.
The “Function Calls” section (see p. 10) will give you some general information about the syntax of most commands. Read the ”Types Used in PI Software” section (p.11) for some general notes about the argument syntax.
In the list below, the functions are arranged according to their functionality.

System Information

BOOL C843_FUNC_DECL **C843_qHLP**(const long iID, char* buffer, long maxlen)
BOOL C843_FUNC_DECL **C843_qHPA**(const long iID, char* buffer, long maxlen)
BOOL C843_FUNC_DECL **C843_qHDR** (const long iID, char* Buffer, long maxlen)
BOOL C843_FUNC_DECL **C843_qERR**(long iID, long* pnError)
BOOL C843_FUNC_DECL **C843_qIDN**(long iID, char* buffer, long maxlen)
BOOL C843_FUNC_DECL **C843_qVER** (long iID, char* szVersion, int iBufferSize)
BOOL C843_FUNC_DECL **C843_qSRG**(long iID, const char* szAxes, const long* iCmdarray, long* iValarray)

Initialization, Parameter Settings

BOOL C843_FUNC_DECL **C843_CST**(long iID, const char* szAxes, const char* names)

BOOL C843_FUNC_DECL **C843_qCST**(long iID, const char* szAxes, char* names, long maxlen)
BOOL C843_FUNC_DECL **C843_INI**(long iID, const char* szAxes)
BOOL C843_FUNC_DECL **C843_qVST**(long iID, char* buffer, long maxlen)
BOOL C843_FUNC_DECL **C843_qTVI**(long iID, char* axes, long maxlen)
BOOL C843_FUNC_DECL **C843_SAI**(long iID, const char* szOldAxes, const char* szNewAxes)
BOOL C843_FUNC_DECL **C843_qSAI**(long iID, char* axes, long maxlen)
BOOL C843_FUNC_DECL **C843_qSAI_**ALL(long iID, char* axes, long maxlen)
BOOL C843_FUNC_DECL **C843_SPA**(long iID, const char* szAxes, const long* iCmdarray, const double* dValarray, const char* szStageNames)
BOOL C843_FUNC_DECL **C843_qSPA**(long iID, const char* szAxes, const long* iCmdarray, double* dValarray, char* szStageNames, long iMaxNameSize)
BOOL C843_FUNC_DECL **C843_DFF**(long iID, const char* szAxes, const double* pdValarray)
BOOL C843_FUNC_DECL **C843_qDFF**(long iID, const char* szAxes, double* pdValarray)
BOOL C843_FUNC_DECL **C843_CLR**(long iID, const char* szAxes)

Servo Activation

BOOL C843_FUNC_DECL **C843_SVO**(long iID, const char* szAxes, const BOOL* pbValarray)
BOOL C843_FUNC_DECL **C843_qSVO**(long iID, const char* szAxes, BOOL* pbValarray)

Referencing

BOOL C843_FUNC_DECL **C843_MNL**(long iID, const char* szAxes)
BOOL C843_FUNC_DECL **C843_MPL**(long iID, const char* szAxes)
BOOL C843_FUNC_DECL **C843_REF**(long iID, const char* szAxes)
BOOL C843_FUNC_DECL **C843_FRF**(long iID, const char* szAxes)
BOOL C843_FUNC_DECL **C843_FPL**(long iID, const char* szAxes)
BOOL C843_FUNC_DECL **C843_FNL**(long iID, const char* szAxes)
BOOL C843_FUNC_DECL **C843_FED**(long iID, const char* szAxes, const long* iEdgeIDArray, const long* iParArray)
BOOL C843_FUNC_DECL **C843_qFED**(long iID, const char* szAxes, long* iEdgeIDArray, long* iParArray)
BOOL C843_FUNC_DECL **C843_qFES**(long iID, const char* szAxes, BOOL* pbValarray)
BOOL C843_FUNC_DECL **C843_qREF**(long iID, const char* szAxes, BOOL* pbValarray)
BOOL C843_FUNC_DECL **C843_qFRF**(long iID, const char* szAxes, BOOL* pbValarray)
BOOL C843_FUNC_DECL **C843_qLIM**(long iID, const char* szAxes, BOOL* pbValarray)
BOOL C843_FUNC_DECL **C843_IsReferencing**(long iID, const char* szAxes, BOOL* pbIsReferencing)
BOOL C843_FUNC_DECL **C843_IsControllerReady**(const long ID, long* piControllerReady)
BOOL C843_FUNC_DECL **C843_GetRefResult**(long iID, const char* szAxes, BOOL* pnResult)
BOOL C843_FUNC_DECL **C843_IsReferenceOK**(long iID, const char* szAxes, BOOL* pbValarray)
BOOL C843_FUNC_DECL **C843_RON**(long iID, const char* szAxes, const BOOL* pbValarray)
BOOL C843_FUNC_DECL **C843_qRON**(long iID, const char* szAxes, BOOL* pbValarray)

Motion and Positioning

BOOL C843_FUNC_DECL **C843_VEL**(long iID, const char* szAxes, const double* pdValarray)
BOOL C843_FUNC_DECL **C843_qVEL**(long iID, const char* szAxes, double* pdValarray)
BOOL C843_FUNC_DECL **C843_ACC**(long iID, const char* szAxes, const double* pdValarray)
BOOL C843_FUNC_DECL **C843_qACC**(long iID, const char* szAxes, double* pdValarray)
BOOL C843_FUNC_DECL **C843_DEC**(long iID, const char* szAxes, const double* pdValarray)
BOOL C843_FUNC_DECL **C843_qDEC**(long iID, const char* szAxes, double* pdValarray)
BOOL C843_FUNC_DECL **C843_qTMN**(long iID, const char* szAxes, double* pdValarray)
BOOL C843_FUNC_DECL **C843_qTMX**(long iID, const char* szAxes, double* pdValarray)
BOOL C843_FUNC_DECL **C843_MOV**(long iID, const char* szAxes, const double* pdValarray)
BOOL C843_FUNC_DECL **C843_qMOV**(long iID, const char* szAxes, double* pdValarray)
BOOL C843_FUNC_DECL **C843_MVR**(long iID, const char* szAxes, const double* pdValarray)
BOOL C843_FUNC_DECL **C843_MVE**(long iID, const char* szAxes, const double* pdValarray)
BOOL C843_FUNC_DECL **C843_IsMoving**(long iID, const char* szAxes, BOOL* pbValarray)
BOOL C843_FUNC_DECL **C843_qONT**(long iID, const char* szAxes, BOOL* pbValarray)
BOOL C843_FUNC_DECL **C843_qPOS**(long iID, const char* szAxes, double* pdValarray)
BOOL C843_FUNC_DECL **C843_POS**(long iID, const char* szAxes, const double* pdValarray)

BOOL C843_FUNC_DECL **C843_DFH**(long ilD, const char* szAxes)
BOOL C843_FUNC_DECL **C843_qDFH**(long ilD, const char* szAxes, double* pdValarray)
BOOL C843_FUNC_DECL **C843_GOH**(long ilD, const char* szAxes)
BOOL C843_FUNC_DECL **C843_HLT**(long ilD, const char* szAxes)
BOOL C843_FUNC_DECL **C843_STP**(long ilD)
BOOL C843_FUNC_DECL **C843_SMO**(long ilD, const char* szAxes, const long* pnValarray)
BOOL C843_FUNC_DECL **C843_qSMO**(long ilD, const char* szAxes, long* pnValarray)
BOOL C843_FUNC_DECL **C843_BRA**(long ilD, const char* szAxes, const BOOL* pbValarray)
BOOL C843_FUNC_DECL **C843_qBRA**(long ilD, char* szBuffer, const long maxlen)

Digital I/O, Triggering

BOOL C843_FUNC_DECL **C843_CTO**(long ID, const long* iTriggerLines, const long* iParamID, const char* szValues, long iArraySize)
BOOL C843_FUNC_DECL **C843_qCTO**(long ID, const long* iTriggerLines, const long* pParamID, char* szBuffer, long iArraySize, long iBufferMaxlen)
BOOL C843_FUNC_DECL **C843_TRO**(long ID, const long* iTriggerLines, const BOOL* pbValarray, long iArraySize)
BOOL C843_FUNC_DECL **C843_qTRO**(long ID, const long* iTriggerLines, BOOL* pbValarray, long iArraySize)
BOOL C843_FUNC_DECL **C843_GetInputChannelNames**(long ilD, char* szBuffer, const long maxlen)
BOOL C843_FUNC_DECL **C843_GetOutputChannelNames**(long ilD, char* szBuffer, const long maxlen)
BOOL C843_FUNC_DECL **C843_DIO**(long ilD, const char* szChannels, const BOOL* pbValarray)
BOOL C843_FUNC_DECL **C843_qDIO**(long ilD, const char* szChannels, BOOL* pbValarray)
BOOL C843_FUNC_DECL **C843_qTIO**(long ilD, long* pINr, long* pONr)

Sending GCS Commands Directly

BOOL C843_FUNC_DECL **C843_GcsCommandset**(long ilD, const char* szCommand)
BOOL C843_FUNC_DECL **C843_GcsGetAnswer**(long ID, char* szAnswer, long bufsize)
BOOL C843_FUNC_DECL **C843_GcsGetAnswerSize**(long ID, long* iAnswerSize)

User Profile Mode

BOOL C843_FUNC_DECL **C843_IsUserProfileActive**(long ilD, const char* szAxes, BOOL* pbValarray)
BOOL C843_FUNC_DECL **C843_UPB**(long ilD, const char* szClusters, const long* iCmdarray, const long* iPararray, const long* iValarray)
BOOL C843_FUNC_DECL **C843_UPD**(long ilD, const char* szClusters, const long* iCmdarray, const long* iPararray, const double* dValarray)
BOOL C843_FUNC_DECL **C843_UPC**(long ilD, const char* szAxes, const char* szClusters, const long* iCmdarray, const long* iPararray)
BOOL C843_FUNC_DECL **C843_UPA**(long ilD, const char* szClusters, const long* iCmdarray)
BOOL C843_FUNC_DECL **C843_UPR**(long ilD, const char* szAxes, const char* szClusters, const long* iCmdarray)
BOOL C843_FUNC_DECL **C843_qUPB**(long ilD,const char* szClusters, const long* iCmdarray, const long* iPararray, long* iValarray)
BOOL C843_FUNC_DECL **C843_qUPD**(long ilD,const char* szClusters, const long* iCmdarray, const long* iPararray, double* dValarray)
BOOL C843_FUNC_DECL **C843_qUPC**(long ilD,char* szClusters,char* szAxes, long* iCmdarray, long* iPararray)
BOOL C843_FUNC_DECL **C843_qUPA**(long ilD,const char* szClusters, const long* iCmdarray, long* iPararray)

Data Recording, Step Response

BOOL C843_FUNC_DECL **C843_DRC**(long ilD, const long* iRecTableId, const char* sRecSourceId, const long* iRecOption, const long* TriggerOption)
BOOL C843_FUNC_DECL **C843_qDRC**(long ilD, const long* iRecTableId, char* sRecSourceId, long* iRecOption, long* TriggerOption, long iArraySize)
BOOL C843_FUNC_DECL **C843_DRT**(long ilD, const long* iRecTableId, const long*

TriggerOption, const char* sValue, long iArrayLength)
BOOL C843_FUNC_DECL **C843_qDRT**(long iID, const long* iRecTableId, long* TriggerOption, char* sValue, long iArraySize, long iValueBufferLength)
BOOL C843_FUNC_DECL **C843_qDRR_**SYNC(long iID, long iRecTableId, long iOffset, long nrValues, double* pdValArray)
BOOL C843_FUNC_DECL **C843_qDRR**(long iID, const long* piRecTableIds, long iNumberOfRecChannels, long iOffset, long nrValues, double** pdValArray, char* szGcsArrayHeader, long iGcsArrayHeaderMaxSize)
BOOL C843_FUNC_DECL **C843_GetAsyncBuffer** (long iID, double **pnValArray)
long C843_FUNC_DECL **C843_GetAsyncBufferIndex**(long iID)
BOOL C843_FUNC_DECL **C843_qTNR**(long iID, long* iNrOfTables)
BOOL C843_FUNC_DECL **C843_RTR**(long iID, long iRecordTableRate)
BOOL C843_FUNC_DECL **C843_qRTR**(long iID, long* iRecordTableRate)
BOOL C843_FUNC_DECL **C843_STE**(long iID, char cAxis, double dOffset)
BOOL C843_FUNC_DECL **C843_qSTE**(long iID, char cAxis, long iOffset, long nrValues, double* pdValarray)

Joystick Control

BOOL C843_FUNC_DECL **C843_JON**(long iId,const long* iJoystickIDsArray, const BOOL* pbValueArray, long iArraySize)
BOOL C843_FUNC_DECL **C843_qJON**(long ID, const long* iJoystickIDsArray, BOOL* pbValueArray, long iArraySize)
BOOL C843_FUNC_DECL **C843_qJAX**(long iId, const long* iJoystickIDsArray, const long* iAxesIDsArray, long iArraySize, char* szAxesBuffer, long iBufferSize)
BOOL C843_FUNC_DECL **C843_JAX**(long ID, long iJoystickID, long iAxesID, const char* szAxesBuffer)

Electronic Gearing

BOOL C843_FUNC_DECL **C843_SRA**(long ID, const char* szAxes, double* dValArray)
BOOL C843_FUNC_DECL **C843_qSRA**(long ID, const char* szAxes, double* dValArray)
BOOL C843_FUNC_DECL **C843_EGE**(long ID, const char* szAxes, BOOL* bValArray)
BOOL C843_FUNC_DECL **C843_qEGE**(long ID, const char* szAxes, BOOL* bValArray)
BOOL C843_FUNC_DECL **C843_MAS**(long ID, const char* szAxes, const char* szMasters)
BOOL C843_FUNC_DECL **C843_qMAS**(long ID, const char* szAxes, char* szMasters)

3.4. Functions for Accessing QMC Commands

The C-843 GCS DLL includes a command which provides access to all the commands of the motion processors command set (“QMC” command set). The QMC commands are sent directly to the motion processor on the C-843 controller board. This command set is the fastest but also the most difficult to use. You have to understand how the motion processor on the C-843 controller board works if you want to use the QMC command set. The corresponding User's Guide and Programmer's Reference for the PMD Navigator Motion Processor MC2140CP are included on the C-843 CD. The QMC command passing feature is implemented for customers who need very fast applications. You have to be extremely careful with the QMC command set, as you can crash your system if you do not handle it correctly. All necessary settings must be made explicitly. Use QMC commands only if the GCS command set is not fast enough.

BOOL C843_FUNC_DECL **C843_SetQMC** (long *iID*, BYTE *bCmd*, BYTE *bAxis*, int *Param*)
BOOL C843_FUNC_DECL **C843_GetQMC** (long *iID*, BYTE *bCmd*, BYTE *bAxis*, int **pResult*)
BOOL C843_FUNC_DECL **C843_SetQMCA** (long *iID*, BYTE *bCmd*, BYTE *bAxis*, WORD *Param1*, int *lParam2*)
BOOL C843_FUNC_DECL **C843_GetQMCA** (long *iID*, BYTE *bCmd*, BYTE *bAxis*, WORD *lParam*, int **pResult*)

3.5. Functions for User-Defined Stages

The C-843 GCS DLL has functions allowing you to both define and save new stages (parameter sets) to the *C843UserStages2.dat* stage database. Being able to specify the parameters of a stage and then save those parameters as a set under the stage name makes it easier to connect to previously defined stages.
Provided that you have initialized the C-843 GCS DLL by calling C843_CST() and C843_INI(), you can afterwards create a new stage parameter set by changing the stage parameters with C843_SPA(). It is important to set the stage parameters correctly. Note that the parameter which determines whether a stage is “new” or not is the Stage Name parameter (ID 0x3C). If it is not specified, the C843_AddStage command will fail. See “Parameter List” on p. 62 for a complete list and the parameter handling description starting on p. 61 for further details.
To save the new stage and thus make it available for a future connection with C843_CST(), use C843_AddStage() to add its parameter set to *C843UserStages2.dat*. After addition to *C843UserStages2.dat* the stage will also appear in the list returned by C843_qVST().
If you want to remove a stage from *C843UserStages2.dat* call C843_RemoveStage().
It may be more comfortable to set the stage parameters using the PIStageEditor (a GUI dialog). See the separate PI Stage Editor manual (SM144E) for a description of how to operate that graphic interface.
The PIStageEditor can also be started from PIMikroMove™. This program provides several functions which ease creating and editing stage parameter sets. For further information, refer to “Tutorials - Frequently Asked Questions” in the PIMikroMove™ manual.

NOTES

The C843_OpenUserStagesEditDialog() or C843_OpenPiStagesEditDialog() functions are provided for compatibility reasons only and should not be used to open the PIStageEditor. Since the PIStageEditor is not modal, problems can occur when the calling application exits before the PIStageEditor window is closed. Please start the PIStageEditor either from PIMikroMove™ or via its executable.

If an older version of the software was installed an existing *C843UserStages.dat* is automatically converted into *C843UserStages2.dat*. For details see “Parameter Databases” on p. 71.

BOOL C843_FUNC_DECL **C843_AddStage** (long ilD, const char* szAxes)
BOOL C843_FUNC_DECL **C843_RemoveStage** (long ilD, char *szStageName)
BOOL C843_FUNC_DECL **C843_OpenUserStagesEditDialog** (long ilD)
BOOL C843_FUNC_DECL **C843_OpenPiStagesEditDialog** (long ilD)

4. C-843 GCS DLL Function Reference (alphabetical)

BOOL C843_FUNC_DECL **C843_ACC** (long *ilD*, const char* *szAxes*, const double * *pdValarray*)

Corresponding GCS command: ACC

Set the acceleration of *szAxes*.

During vectorial moves started with C843_MVE(), velocities, accelerations and decelerations will be calculated to ensure that all axes follow the path. The current settings for velocity, acceleration and deceleration define the maximum possible values, and the slowest axis determines the resulting velocities.

Arguments:

ilD ID of controller
szAxes string with axes
pdValarray accelerations for the axes

Note: The value in *pdValarray* is limited by the stage parameter 0x4A (maximum allowed acceleration; see Motion Parameters, p. 61).

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_AddStage** (long ilD, const char* *szAxes*)

Adds the stage of the specified *axis* to the file *C843UserStages2.dat* with the user defined stages.

Arguments:

ilD ID of controller
szAxes character of the axis.

Returns:

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

BOOL C843_FUNC_DECL **C843_BRA** (long *ilD*, const char* *szAxes*, const BOOL * *pbValarray*)

Corresponding GCS command: BRA

Set brake state for *szAxes* to on (**TRUE)** or off (**FALSE)**

CAUTION: Setting the brake with C843_BRA() does not affect the servo state of the axis. I.e. if you activate the brake, the servo remains on so that the motor may work against the brake which can cause overheating. In this case, it may be necessary to switch the servo off temporarily. Do not deactivate the brake when the servo is switched off! Otherwise unwanted motion can occur. Unwanted motion could cause irreparable damage to the stage and the application setup.

Note that the brake is activated automatically when the servo is switched off with C843_SVO(), and deactivated when the servo is switched on.

Arguments:

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

Returns:

TRUE if successful, **FALSE** otherwise

void C843_FUNC_DECL **C843_CloseConnection** (long ilD)

Close connection to C-843 controller associated with ID = *ilD*; the ID will no longer be valid.

Arguments:

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

Returns:

none

BOOL C843_FUNC_DECL **C843_CLR** (long *ilD*, const char* *szAxes*)

Corresponding GCS command: CLR

Clear status of *szAxes*.

The following actions are done by C843_CLR():

Switches the servo on.

Resets error to 0.

If the stage has tripped a limit switch, C843_CLR() will move it away from the limit switch until the limit condition is no longer given, and the target position is set to the current position afterwards.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected

Returns:

TRUE if successful, **FALSE** otherwise

int C843_FUNC_DECL **C843_Connect** (long *iBoardNumber*)

Open a C-843 PCI board with the specified board number. Please read the C-843 manual to find out which number to use. If you have installed only one board, use 1. All future calls to control this C-843 need the ID returned by this call. Use C843_ListPCI to get a list of possible board numbers.

Arguments:

iBoardNumber: Board to use.

Returns:

-1 if connection failed, board ID otherwise

BOOL C843_FUNC_DECL **C843_CST** (long *ilD*, const char* *szAxes*, const char * *names*)

Corresponding GCS command: CST

Assigns *szAxes* to stages. This is done by loading stage parameters suitable for the connected hardware from a stage database. Afterwards, the loaded values must be written to the controller by calling C843_INI() to initialize the motion processor on the C-843 board.

Valid stage names can be listed with C843_qVST() which reports the content of the stage databases (PIStages2.dat, C843Userstages2.dat, M-xxx.dat files) used by the C-843 GCS DLL.

If no stage is connected to the corresponding socket, or if motion of the axis is strictly forbidden, the stage name should be “NOSTAGE”. This deactivates the axis which means that this axis is not available for axis-related commands any more (e.g. motion commands, position queries). You can undo axis deactivation at any time by setting a valid stage name with C843_CST().

Note: To connect a stage, always use C843_CST(). Do not set the Stage Name parameter (ID 0x3C) with C843_SPA() for that purpose. Otherwise the stage parameters will not be loaded properly from the stage database.

The stage names are separated by \n (line feed), for example "M-505.1PD\nM-505.2PD". See “Functions for Initialization of the C-843 GCS DLL” (p. 12) for an example of how to setup the C-843 library.

Arguments:

iID ID of controller
szAxes identifiers of the stages, if "" or **NULL** all axes are affected
names the names of the Stages separated by \n (line feed)

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_CTO** (long *iID*, const long* *iTriggerLinesArray*, const long* *iParamIDArray*, const char* *szValues*, long *iArraySize*)

Corresponding GCS command: CTO

Configures the trigger output conditions for the given digital output line.

The trigger output will only become active when enabled with C843_TRO().

Arguments:

iID ID of controller
iTriggerLinesArray identifiers of the digital output lines located on the J8 ("All-axes") connector on the C-843 board (digital output from the motion processor, TTL, max. 5 mA).
with C-843.21: can be 1 and 2
with C-843.41: can be 1 to 4
The lines can be brought out of the PC housing using an adapter bracket with a sub-D 15m connector (included with C-843).
iParamIDArray identifiers of the parameters to be set, available are
2 = Axis
3 = TriggerMode
7 = Polarity
szValues the values to which the parameters are to be set
for parameter identifier = 2:
the axis to connect to the trigger output line

for parameter identifier = 3:
2 = OnTarget
5 = Motion Error
6 = In Motion

for parameter identifier = 7:
0 = Active Low
1 = Active High
iArraySize number of trigger lines to be configured

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_DEC** (long *ilD*, const char* *szAxes*, const double * *pdValarray*)

Corresponding GCS command: DEC

Set the deceleration of *szAxes*.

During vectorial moves started with C843_MVE(), velocities, accelerations and decelerations will be calculated to ensure that all axes follow the path. The current settings for velocity, acceleration and deceleration define the maximum possible values, and the slowest axis determines the resulting velocities.

Arguments:

ilD ID of controller
szAxes string with axes
pdValarray decelerations for the axes

Note: The value in *pdValarray* is limited by the stage parameter 0x4B (maximum allowed deceleration; see Motion Parameters, p. 61).

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_DFF** (long *ilD*, const char* *szAxes*, const double * *pdValarray*)

Corresponding GCS command: DFF

Set the scaling factor for physical units for *szAxes*. This factor is applied to the counts-per-physical-unit value (parameter 0xE / parameter 0xF). For example, a scaling factor of 25.4 sets the working units to inches. Changing the scale factor will change the numerical results of other commands.

Note: To change the sacling factor for an axis, always use C843_DFF(). Do not set the Scaling Factor parameter (ID 0x12) with C843_SPA(). Otherwise the scaling factor will not be applied properly.

Arguments:

ilD ID of controller
szAxes string with axes
pdValarray factors for the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_DFH** (long *ilD*, const char* *szAxes*)

Corresponding GCS command: DFH

Makes current positions of *szAxes* the new home positions.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_DIO** (long *ilD*, const char* *szChannels*, const BOOL * *pbValarray*)

Corresponding GCS command: DIO

Switches the specified output line(s) to specified state(s). If pbValarray[index] is **TRUE** the mode is set to **HIGH**, otherwise it is set to **LOW**. Can be used to trigger external devices.

Note that with some older C-843 hardware models, usage of the output lines and usage of the volatile memory on the C-843 board (data recorder, User Profile mode) are mutually exclusive. This means that after the C-843 board was connected in the software, only the functionality called first is available. The selection is reset any time the C-843 board is reconnected. If, for example, the data recorder configuration was queried with C843_qDRC() before C843_DIO() was called, you can not set the digital output lines until you reconnect the C-843.

Arguments:

ilD ID of controller
szChannels string with the identfiers of the digital output lines of the 26-pin IDC connector (J5) on the C-843 board (A to H). They can be brought out of the PC housing using an adapter bracket with a sub-D 25f connector (included with C-843).
pbValarray array containing the states of specified digital output channels, **TRUE** if HIGH, **FALSE** if LOW

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_DRC** (long *ilD*, const long* *iRecTableId*, const char* *sRecSourceId*, const long* *iRecOption*, const long* *TriggerOption*)

Corresponding GCS command: DRC

Set data recorder configuration: determines the data source and the kind of data (RecOption) used for the given data recorder table.

The record option set with C843_DRC() for data recorder table 1 is automatically changed to "actual position" when a step response measurement is made with C843_STE().

If one data recorder table is deactivated by choosing record option 0 ("nothing is recorded"), all subsequent tables are deactivated too. The points available for recording are in equal shares allocated to the tables with non-zero record options (for the total number of points to allocate ask C843_qSPA()) with parameter 0x16000200, for the maximum number of record tables ask C843_qTNR()). Note that the data recorder shares the 32,256 points of volatile memory provided on the C-843 card (referred to as “external RAM” in the MotionProcessor Users Guide) with the multi-axis motion profiles which can be created by the User Profile Mode commands (C843_UPx functions). It may be necessary to free memory occupied by user-defined motion profiles using C843_UPC() to have enough memory for data recording.

With C843_qHDR() you will obtain a list of available record options and trigger options and information about additional parameters and commands regarding data recording.

Note that with some older C-843 hardware models, you can not use the data recorder if the digital output lines have been set with C843_DIO() before any data-recorder-related command was sent. To use the data recorder, reconnect the C-843 board in the software.

For detailed information see "Data Recording" in the C-843 GCS Commands manual (SM149E).

Arguments:

ilD ID of controller
iRecTableId identifier of the data recorder table
sRecSourceId identifier of the axis of which the performance variable is to be recorded
iRecOption identifier of the performance variable (kind of data to be recorded), can be
0 = nothing is recorded
1 = commanded position of axis
2 = actual position of axis
3 = position error of axis
70 = commanded velocity of axis
71 = commanded acceleration of axis
72 = actual velocity of axis
73 = motor output of axis

74 = chipset time
75 = capture register of axis
76 = integral of axis
77 = derivative of axis
78 = event status register of axis
79 = activity status register of axis
80 = signal status register of axis
82 = PID servo error of axis (input of PID servo filter)

iTriggerOption not used

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_DRT** (long iID, const long* *iRecTableId*, const long* *TriggerOption*, const char* *sValue*, long *iArrayLength*)

Corresponding GCS command: DRT

Defines a trigger source.

By default data recording is triggered when a step response measurement is made with C843_STE().

A trigger option set with C843_DRT() will become valid for all data recorder tables with non-zero record option.

With C843_qHDR() you will obtain a list of available record options and trigger options and information about additional parameters and commands regarding data recording.

Note that with some older C-843 hardware models, you can not use the data recorder if the digital output lines have been set with C843_DIO() before any data-recorder-related command was sent. To use the data recorder, reconnect the C-843.

For detailed information see "Data Recording" in the C-843 GCS Commands manual (SM149E).

Arguments:

iID ID of controller
iRecTableId identifier of the record table ID. Here, only the pseudo ID “0” is valid, and one setting has effect on all data recorder channels
iTriggerOption identifier of the trigger to be used. Available identifiers are:
0 = default setting; data recording is triggered with C843_STE()
1 = any command changing position (e.g. C843_MVR(), C843_MOV(), C843_SMO())
2 = next command, resets trigger after execution
4 = start on InMotion, stop on AxisSettled
5 = start immediately, stop on AxisSettled
sValue not used
iArrayLength number of table IDs to be configured, must be 1

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_EGE** (long *iID*, const char* *szAxes*, BOOL * b*ValArray*)

Corresponding GCS command: EGE

Enable or disable electronic gearing mode for given axis. If bValarray[index] is **TRUE** the mode is set to **ON**, otherwise it is set to **OFF**.

Via electronic gearing a "master" and a "geared" (slave) axis are linked, so that motion of the master automatically entails proportional motion of the slave.

Enabling electronic gearing with C843_EGE() for an axis means that this axis will be linked as slave to the master axis selected with C843_MAS(). The gear ratio to be applied can be set with C843_SRA(). Master selection and ratio setting for an axis are only possible if electronic gearing is disabled for that axis. The ratio setting is checked automatically upon the activation of electronic gearing. If the slave axis is not able to follow the master axis, you have to adapt the ratio value.

Electronic gearing can only be enabled for axes which are referenced. Reference moves are not allowed for axes which are involved in electronic gearing (as master or slave), and their referencing mode can not be changed.

A slave axis can not be commanded directly by move commands. It is only moved when its master axis moves. When motion is commanded for the master axis, the available travel ranges for master and slave are checked.

Joystick operation is possible for master axes. Slave axes connected to a joystick-controlled master will move correspondingly. Slave axes can not be assigned to joystick axes, i.e. they can not be controlled directly by a joystick. If joystick control is enabled for an axis, electronic gearing can not be enabled for that axis.

Arguments:

iID ID of controller
szAxes string with axis identifiers
bValarray array containing the states of the electronic gearing mode of the specified axes, **TRUE** if ON, **FALSE** if OFF

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_FED** (long *iID*, const char* *szAxes,* const long* iEdgeIDArray, const long* iParArray)

Corresponding GCS command: FED

Move given Axes to a given signal edge. Call **C843_IsReferencing**() (p.31) to find out if an axis is still moving and **C843_qFES**() (p. 41) to get the result from the controller. The controller will be "busy" while referencing, so most other commands will cause a **PI_CONTROLLER_BUSY** error. Use **C843_STP**() (p.56*)* to stop it.

Notes:

In contrast to the referencing functions (C843_MNL, C843_MPL, C843_REF, C843_FNL, C843_FPL and C843_FRF), this function does not change the reference state of the axis and does not set a certain position value at the selected edge. It does move out of the limit condition, therefore the axis motion finishes at the same position as with the corresponding referencing functions.

If multiple axes are given, they are moved synchronously.

The C-843 GCS DLL detects the presence or absence of reference switch and limit switches using parameters (ID 0x14 for reference switch; ID 0x32 for limit switches). According to the values of those parameters, the C-843 GCS DLL enables or disables C843_FED motions to the appropriate signal edges. You can adapt the parameter values to your hardware using C843_SPA. See "Motion Parameter" (p. 61) for more information.

C843_FED can be used to measure the physical travel range of a new mechanics and thus to identify the values for the corresponding parameters: the distance from negative to positive limit switch, the distance between the negative limit switch and the reference switch (DISTANCE_REF_TO_N_LIM, parameter ID 0x17), and the distance between reference switch and positive limit switch (DISTANCE_REF_TO_P_LIM, parameter ID 0x2F). See "Travel Range Adjustment" (p. 68) for more information.

Motion commands like C843_FED are not allowed when the joystick is active for the axis.

If the position error of an axis falls out of the window formed by the Maximum position error parameter, the servo is switched off automatically for that axis, motion of all other axes is stopped immediately, and error code -1024 is set.

Arguments:

ilD ID of controller
szAxes axes to move.
iEdgeIDArray identifier of the edge to which the stage is to be moved. The following edge types with their parameter settings are available:
1 = negative limit switch, *iParArray* is 0 when the default setting should be used (e.g. from Pistages2.dat), 1 when active high, -1 when active low
2 = positive limit switch, *iParArray* is 0 when the default setting should be used (e.g. from Pistages2.dat), 1 when active high, -1 when active low
3 = reference switch, *iParArray* is 0 when the default setting should be used (e.g. from Pistages2.dat), 1 when active high, -1 when active low
4 = autofind additional switch changing its state at a certain position (signal must be connected to the digital input line of the motion processor which corresponds to the axis given in C843_FED() (All-axes connector J8 on the C-843 board)), *iParArray* gives the signal state to the left of the edge (high = 1 or low = -1)
iParArray parameter to define the polarity of the signal in which the edge is to be searched for

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_FNL** (long *ilD*, const char* *szAxes*)

Corresponding GCS command: `FNL`

Moves all axes *szAxes* synchronously to their negative limit switch. This can be used for fast referencing of multiple axes which have no reference switch. Call **C843_IsReferencing**() (p.31) to find out if an axis is still moving and **C843_qFRF**() (p. 41) to get the result from the controller. The controller will be "busy" while referencing, so most other commands will cause a **PI_CONTROLLER_BUSY** error. Use **C843_STP**() (p.56) to stop it.

Notes:

Calling the **C843_FNL**() function resets the current positions of the axes. Therefore moving the axes to the same position using **MOV**() (absolute move) before and after a call of this function may move the stage to a different physical position! You should call **C843_FNL**() only once after a call of **C843_INI**().

To reference axes one after the other, use **C843_MNL**() (p.33) instead.

Arguments:

ilD ID of controller
szAxes axes to move.

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_UNKNOWN_AXIS_IDENTIFIER `cAxis` is not a valid axis identifier

BOOL C843_FUNC_DECL **C843_FPL** (long *ilD*, const char* *szAxes*)

Corresponding GCS command: FPL

Moves all axes *szAxes* synchronously to their positive limit switch. This can be used for fast referencing of multiple axes which have no reference switch. Call **C843_IsReferencing**() (p.31) to find out if an axis is still moving and **C843_qFRF**() (p. 41) to get the result from the controller. The controller will be "busy" while referencing, so most other commands will cause a **PI_CONTROLLER_BUSY** error. Use **C843_STP**() (p.56*)* to stop it.

Notes:

Calling the **C843_FPL**() function resets the current positions of the axes. Therefore moving the axes to the same position using **MOV**() (absolute move) before and after a call of this function may move the stage to a different physical position! You should call **C843_FPL**() only once after a call of **C843_INI**().

To reference axes one after the other, use **C843_MPL**() (p.34) instead.

Arguments:

ilD ID of controller
szAxes axes to move.

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_UNKNOWN_AXIS_IDENTIFIER cAxis is not a valid axis identifier

BOOL C843_FUNC_DECL **C843_FRF** (long *ilD*, const char* *szAxes*)

Corresponding GCS command: FRF

Fast (synchronous) reference move of all axes *szAxes*. Call **C843_IsReferencing**() (p.31) to find out if the axes are still moving and **C843_qFRF**() (p. 41) to get the results from the controller. The controller will be "busy" while referencing, so most other commands will cause a **PI_CONTROLLER_BUSY** error. Use **C843_STP**() (p.56*)* to stop it.

Notes:

Calling the **C843_FRF** function resets the current positions. That means moving to the same position using the **MOV**() command (absolute move) before and after a call of this function may move the stage to a different physical position! You should call this function only once after a call of **C843_INI**().

To reference axes one after the other, use **C843_REF**() (p.51) instead.

Arguments:

ilD ID of controller
szAxes string with axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_GcsCommandset** (long *ilD*, const char* *szCommand*)

Sends a GCS command to the C-843. Any GCS command can be sent, but this command is intended for commands not having a function in the current version of the library.

Arguments:

ilD ID of controller
szCommand the GCS command as string.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_GcsGetAnswer** (long *ilD*, const char* *szAnswer*, long *bufsize*)

Gets the answer of a GCS command. The answers of a GCS command are buffered inside the DLL. This is to be compatible to the PI controllers which are connected via an external interface. The buffer simulates the interface. Each call to this function returns the oldest answer in the buffer.

Arguments:

ilD ID of controller
szAwnser the buffer to take the answer.
bufsize the buffer size of the answer.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_GcsGetAnswerSize** (long *ilD*, long * *iAnswerSize*)

Gets the size of an answer of a GCS command.

Arguments:

ilD ID of controller
iAnswerSize pointer to take the size of the next answer.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_GetAsyncBuffer** (long *ilD*, double **pnValArray*)

Get address of internal buffer used for storing data read in by a call to C843_qDRR().

Arguments:

ilD ID of controller
pnValarray pointer to receive address of internal array used to store the data, the DLL will have allocated enough memory to store all data; call C843_GetAsyncBufferIndex() to find out how many data points have been transferred up to that time.

Returns:

TRUE if successful, **FALSE** otherwise

long C843_FUNC_DECL **C843_GetAsyncBufferIndex** (long *ilD*)

Get index used for the internal buffer filled with data read in by a call to C843_qDRR().

Arguments:

ilD ID of controller

Returns:

Index of the data element which was last read in, **-1** otherwise

long C843_FUNC_DECL **C843_GetError** (long *ilD*)

Get error status of C-843. If there is no internal error this function will call **C843_qERR**() (p.40).

Returns:

error ID, see **Error Codes** (p.61) for the meaning of the codes.

BOOL C843_FUNC_DECL **C843_GetInputChannelNames** (long *ilID*, char * *szBuffer*, const long *maxlen*)

Get valid character identifiers for installed digital input channels. Each character in the returned string is the valid channel identifier of an installed digital input channel.

Notes:

C843_GetInputChannelNames reports the identifiers for the digital input lines on the 26-pin IDC connector (J5) of the C-843 board. They can be brought out of the PC housing using an adapter bracket with a sub-D 25f connector (included with C-843). The input lines on the 16-pin IDC connector (J8) are not contained in the response to **C843_GetInputChannelNames**.

Call **C843_qDIO**() (p.47) to query the states of the digital input channels.

Arguments:

ilID ID of controller
szBuffer buffer for storing the identifier string
maxlen size of *szBuffer*, must be given to avoid a buffer overflow.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_GetOutputChannelNames** (long *ilID*, char * *szBuffer*, const long *maxlen*)

Get valid character identifiers for installed digital output channels. Each character in the returned string is the valid channel identifier of an installed digital output channel.

Notes:

C843_GetInputChannelNames reports the identifiers for the digital output lines on the 26-pin IDC connector (J5) of the C-843 board. They can be brought out of the PC housing using an adapter bracket with a sub-D 25f connector (included with C-843). The output lines on the 16-pin IDC connector (J8) are not contained in the response to **C843_GetOutputChannelNames**.

Call **C843_DIO**() (p.22) to set the states of the digital outputs.

Arguments:

ilID ID of controller
szBuffer buffer for storing the identifier string
maxlen size of *szBuffer*, must be given to avoid a buffer overflow.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_GetQMC** (long *ilID*, BYTE *bCmd*, BYTE *bAxis*, int * *pResult*)

Sends a QMC query command to the C-843 controller.

Arguments:

ilID ID of controller
bCmd the QMC command.
bAxis the axis (The first axis is axis 0).
pResult pointer to the result.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_GetQMCA** (long *ilD*, BYTE *bCmd*, BYTE *bAxis*, WORD *lParam*, int * *pResult*)

Sends a QMC query command with one argument to the C-843 controller.

Arguments:

ilD ID of controller
bCmd the QMC command.
bAxis the axis (The first axis is axis 0).
lParam the argument.
pResult pointer to the result.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_GetRefResult** (long *ilD*, const char* *szAxes*, BOOL * *pnResult*)

Get results of last call to **C843_REF**() (p.51), **C843_MNL**() (p.33) or **C843_MPL**() (p.34). If still referencing, or no reference move was started since startup of library, the result is 0. Call **C843_qREF**() (p.44) to see which axes have a reference. To reference an axis call **C843_REF**() (p.51) for axes with reference or **C843_MNL**() (p.33) or **C843_MPL**() (p.34) for axes without reference. For fast (synchronous) referencing of multiple axes call **C843_FRF**() (p. 26) for axes with reference or **C843_FNL**() (p. 25) or **C843_FPL**() (p. 26) for axes without reference. Call **C843_IsReferencing**() to find out if there are axes (still) referencing.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pnResult 1 if successful, 0 if reference move failed, has not finished yet or axis has no reference

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_GOH** (long *ilD*, const char* *szAxes*)

Corresponding GCS command: GOH

Move all axes in *szAxes* to their home positions.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_HLT** (long *ilD*, const char* *szAxes*)

Corresponding GCS command: HLT

Halt motion of *szAxes* smoothly.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_INI** (long *ilD*, const char* *szAxes*)

Corresponding GCS command: `INI`

Initializes motion control chip for *szAxes.*

The following actions are done by C843_INI():

Writes the stage parameters which were loaded with C843_CST() from the stage database to the controller.

Switches the servo on.

Sets reference mode to 1, i.e. C843_REF(), C843_FRF(), C843_MNL(), C843_FNL(), C843_MPL() or C843_FPL() is required to reference the axis, usage of C843_POS() is not allowed.

Sets reference state to "not referenced".

If the stage has tripped a limit switch, C843_INI() will move it away from the limit switch until the limit condition is no longer given, and the target position is set to the current position afterwards.

Sets trigger output mode to default configuration.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_IsConnected** (long *ilD*)

Check if there is a C-843 controller with an ID of *ilD*.

Returns:

TRUE if *ilD* points to an exisiting controller, **FALSE** otherwise.

BOOL C843_FUNC_DECL **C843_IsControllerReady** (const long *ilD*, long* *piControllerReady*)

Corresponding **command:** #7 (ASCII 7)

Asks controller for ready status (tests if controller is ready to perform a new command).

Arguments:

ilD ID of controller
piControllerReady status of the controller:
B1h (ASCII character 177 = "±" in Windows) if controller is ready
B0h (ASCII character 176 = "°" in Windows) if controller is not ready (e.g. performing a referencing command)

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_IsMoving** (long *ilD*, const char* *szAxes*, BOOL * *pbValarray*)

Check if *szAxes* are moving. If an axis is moving the corresponding element of the array will be TRUE, otherwise FALSE. If no axes were specified, only one boolean value is returned and *pbValarray[0]* will contain a generalized state: TRUE if at least one axis is moving, FALSE if no axis is moving.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pbValarray status of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_IsReferenceOK** (long *ilD*, const char* *szAxes*, BOOL * *pbValarray*)

Check the reference status of the given axes. Call **C843_qREF**() (p.44) to find out which axes have a reference. To reference an axis call **C843_REF**() (p.51) for axes with reference, or **C843_MNL**() (p.33) or **C843_MPL**() (p.34) for axes without reference. For fast (synchronous) referencing of multiple axes call **C843_FRF**() (p. 26) for axes with reference or **C843_FNL**() (p. 25) or **C843_FPL**() (p. 26) for axes without reference.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pbValarray **TRUE** if the axis is referenced, **FALSE** if not

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_IsReferencing** (long *ilD*, const char* *szAxes*, BOOL * *pbIsReferencing*)

Check if C-843 is busy with referencing.

Note:

If you do not specify any axis, you will get back only one BOOL. It will be **TRUE** if the controller is referencing any axis.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** overall state is returned.
pbIsReferencing status of axes or controller, **TRUE** if referencing, **FALSE** otherwise

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_IsUserProfileActive** (long *ilD*, const char* *szAxes*, BOOL * *pbValarray*)

Corresponding GCS command: #9

Check if *szAxes* are moving in User Profile Mode. If an axis is moving in UP mode the corresponding element of the array will be TRUE, otherwise FALSE. If no axes were specified, only one boolean value is returned and *pbValarray[0]* will contain a generalized state: TRUE if at least one axis is moving in UP mode, FALSE if no axes are moving in UP mode.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pbValarray array to receive status of the specified axes or of the axes as a group

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_JAX** (long *ilD*, long *iJoystickID*, long *iAxesID*, const char* *szAxesBuffer*)

Corresponding command: JAX

Set axis controlled by a joystick connected to the PC.

Each axis of the controller can only be controlled by one joystick axis.

For joystick control, connect the joystick device to the PC before you start the C-843 GCS DLL (which is called, for example, if you connect to the C-843 in PIMikroMove™ or in PITerminal). Otherwise the joystick will not be recognized by the software. Avoid removing and reconnecting the joystick at run time of the software since this can cause unpredictable results.

Joystick control must be enabled with C843_JON().

See "Joystick Control" in the C-843 GCS Commands manual (SM149E) for details.

Arguments:

iID ID of controller
iJoystickID joystick device connected to the PC
iAxesID ID of the joystick axis
szAxesBuffer name(s) of the axis or axes to be controlled by this joystick axis

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_JON** (long *iId*, const long* *iJoystickIDsArray*, const BOOL* *pbValueArray*, long *iArraySize*)

Corresponding command: JON

Enable or disable a joystick device which is connected to the PC.

For joystick control, connect the joystick device to the PC before you start the C-843 GCS DLL (which is called, for example, if you connect to the C-843 in PIMikroMove™ or in PITerminal). Otherwise the joystick will not be recognized by the software. Avoid removing and reconnecting the joystick at run time of the software since this can cause unpredictable results.

For joystick control of a controller axis, this axis must be assigned to a joystick axis with C843_JAX().

While a joystick connected to the C-843 is enabled with C843_JON(), this joystick controls the axis velocity. In open-loop mode (servo off) no joystick operation is possible. When disabling a joystick, the target position is set to the current position for joystick-controlled axes.

Motion commands like C843_MOV() are not allowed when a joystick is active on the axis.

See "Joystick Control" in the C-843 GCS Commands manual (SM149E) for details.

Arguments:

iID ID of controller
iJoystickIDsArray array with joystick devices connected to the controller
pbValarray pointer to array with joystick enable states (0 for deactivate, 1 for activate)
iArraySize size of arrays

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_ListPCI** (char* *szIDList*, long *maxlen*)

Receive a list of possible board numbers to be used with **C843_Connect**(). The returned string holds lines of numbers. For example, when there are two boards available with the board numbers 1 and 3, szIDList will hold

"1<SP><LF>

3<LF>"

Arguments:

szIDList buffer for storing the string read in from DLL, lines are separated by SP LF (space, line feed)
maxlen size of *buffer*, must be given to avoid a buffer overflow.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_MAS** (long *ilD*, const char* *szAxes*, const char* *szMasters*)

Corresponding GCS command: `MAS`

Set the electronic gearing master axes for *szAxes*. For details see C843_EGE().

Arguments:

ilD ID of controller
szAxes string with "slave" axes
szMasters string with the master axes for the slaves in *szAxes*

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_MNL** (long *ilD*, const char* *szAxes*)

Corresponding GCS command: `MNL`

Moves axis *szAxes* to its negative limit switch. This can be used to reference an axis without a reference switch. For fast (synchronous) referencing of multiple axes call **C843_FNL**() (p. 25) instead. Call **C843_IsReferencing**() (p.31) to find out if an axis is still moving and **C843_GetRefResult**() (p.29) to get the result from the controller. The controller will be "busy" while referencing, so most other commands will cause a **PI_CONTROLLER_BUSY** error. Use **C843_STP**() (p.56*)* to stop it.

Note: Calling the **C843_MNL**() function resets the current position. Therefore moving the axis to the same position using **MOV**() (absolute move) before and after a call of this function may move the stage to a different physical position! You should call **C843_MNL**() only once after a call of the **INI**() command.

Arguments:

ilD ID of controller
szAxes axes to move.

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_UNKNOWN_AXIS_IDENTIFIER `cAxis` is not a valid axis identifier

BOOL C843_FUNC_DECL **C843_MOV** (long *ilD*, const char* *szAxes*, const double * *pdValarray*)

Corresponding GCS command: `MOV`

Move *szAxes* to absolute position.

Arguments:

ilD ID of controller
szAxes string with axes
pdValarray target positions of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_MPL** (long *ilD*, const char* *szAxes*)

Corresponding GCS command: `MPL`

Moves axis *szAxes* to its positive limit switch. This can be used to reference an axis without a reference switch. For fast (synchronous) referencing of multiple axes call **C843_FPL**() (p. 26) instead. Call **C843_IsReferencing**() (p.31) to find out if the axis is still moving and **C843_GetRefResult**() (p.29) to get the result from the controller. The controller will be "busy" while referencing, so most other commands will cause a **PI_CONTROLLER_BUSY** error. Use **C843_STP**() (p.56) to stop it.

Note: Calling the **C843_MPL**() function resets the actual position. Therefore moving the axis to the same position using **MOV**() (absolute move) before and after a call of this function may move the stage to a different physical position! You should call **C843_MPL**() only once after a call of the **INI**() command.

Arguments:

ilD ID of controller
szAxes axes to move.

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_UNKNOWN_AXIS_IDENTIFIER `cAxis` is no valid axis identifier

BOOL C843_FUNC_DECL **C843_MVE** (long *ilD*, const char* *szAxes*, const double * *pdValarray*)

Corresponding GCS command: `MVE`

Move *szAxes* absolutely on linear path. If the affected axes are mounted in a way that they move perpendicular to each other, the combined motion of them will describe a linear path. This is achieved by setting the accelerations, velocities and decelerations temporarily.

Arguments:

ilD ID of controller
szAxes string with axes
pdValarray positions of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_MVR** (long *ilD*, const char* *szAxes*, const double * *pdValarray*)

Corresponding GCS command: MVR

Move *szAxes* relatively.

Arguments:

ilD ID of controller
szAxes string with axes
pdValarray positions of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_OpenPiStagesEditDialog** (long *ilD*)

Opens a dialog to look at the *PiStages2.dat* file, which contains the stages defined by PI. No changes can be made to this file.

CAUTION: This function is provided for compatibility reasons only. It is not recommended to open the *PIStageEditor* this way. Since the *PIStageEditor* is not modal, problems can occur when the calling application exits before the *PIStageEditor* window is closed. Please start the *PIStageEditor* either from *PIMikroMove*™ or via its executable to check stage parameter sets in *PiStages2.dat*.

Arguments:

ilD ID of controller

Returns:

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

BOOL C843_FUNC_DECL **C843_OpenUserStagesEditDialog** (long *ilD*)

Opens a dialog to edit, add and remove stages from the *C843UserStages2.dat* file, which contains the user-defined stages.

CAUTION: This function is provided for compatibility reasons only. It is not recommended to open the *PIStageEditor* this way. Since the *PIStageEditor* is not modal, problems can occur when the calling application exits before the *PIStageEditor* window is closed. Please start the *PIStageEditor* either from *PIMikroMove*™ or via its executable to edit, add or remove stage parameter sets in *C843UserStages2.dat*.

Arguments:

ilD ID of controller

Returns:

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

BOOL C843_FUNC_DECL **C843_POS** (long *iID*, const char* *szAxes*, const double * *pdValarray*)

Corresponding command: POS

Sets absolute position for given axes. Reference mode for the axes must be OFF.

When reference mode is OFF only relative moves can be commanded (**C843_MVR**() (p.36)) until the actual position is set with this command. See **C843_RON**() (p.51) for a detailed description of reference mode and how to turn it on and off. For stages with neither reference nor limit switch, reference mode is automatically OFF.

CAUTION:

The minimum and maximum commandable positions (C843_qTMN(), C843_qTMX()) are not adapted when a position is set with C843_POS(). This may result in target positions which are allowed by the software and cannot be reached by the hardware. Also target positions are possible which can be reached by the hardware but are denied by the software. Furthermore, the home position can be outside of the physical travel range after using C843_POS().

Arguments:

iID ID of controller
szAxes string with axes
pdValarray absolute positions for the specified axes

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_CNTR_CMD_NOT_ALLOWED_FOR_STAGE if the reference mode for any of the given axes is ON

BOOL C843_FUNC_DECL **C843_qACC** (long *iID*, const char* *szAxes*, double * *pdValarray*)

Corresponding GCS command: ACC?

Get the accelerations of *szAxes*.

Arguments:

iID ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pdValarray array to be filled with the accelerations of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qBRA** (long *iID*, char * *szBuffer*, const long *maxlen*)

Corresponding GCS command: BRA?

Get axes with brakes.

Arguments:

iID ID of controller
szBuffer buffer to store the read in string
maxlen size of *buffer*, must be given to avoid a buffer overflow.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qCST** (long *iID*, const char* *szAxes*, char * *names*, long *maxlen*)

Corresponding GCS command: CST?

Get the type names of the stages connected to *szAxes*. The single names begin with the axis identifier+'=' and are separated by \n (line feed). For example "1=M-505.1PD\n2=M-505.2PD". When nothing was configured, the unconfigured axes will be named "NOSTAGE". **C843_qSAI**() (p.45) will only return the axes configured with **C843_CST**() (p.19), **C843_qCST**() (p.37) will always return all axes. See "Functions for Initialization of the C-843 GCS DLL" (p.12) for an example of how to setup the C-843 library.

Arguments:

iID ID of controller
szAxes identifiers of the axes, if "" or **NULL** all axes are affected
names buffer for storing the string read in from controller, lines are separated by \n (line feed)
maxlen size of *name*, must be given to avoid a buffer overflow.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qCTO** (long *iID*, const long* *iTriggerLinesArray*, const long* *pParamIDArray*, char* *szBuffer*, long *iArraySize*, long *iBufferMaxlen*)

Corresponding command: CTO?

Get the Trigger Output configuration for the given trigger output line.

Arguments:

iID ID of controller
iTriggerLinesArray is an array with the trigger output lines of the controller
pParamIDArray is an array with the CTO parameter IDs, see C843_CTO() for details.
szBuffer buffer to receive the values to which the CTO parameters are set, see C843_CTO() for details.
iArraySize is the size of the buffer *iTriggerLinesArray*
iBufferMaxlen is the size of the buffer *szBuffer*

Returns:

TRUE if no error, FALSE otherwise

BOOL C843_FUNC_DECL **C843_qDEC** (long *iID*, const char* *szAxes*, double * *pdValarray*)

Corresponding GCS command: DEC?

Get the decelerations of *szAxes*.

Arguments:

iID ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pdValarray array to be filled with the decelerations of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qDFF** (long *ilD*, const char* *szAxes*, double * *pdValarray*)

Corresponding GCS command: DFF?

Get scale factors for *szAxes* set with **C843_DFF**() (p.21).

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pdValarray factors for the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qDFH** (long *ilD*, const char* *szAxes*, double * *pdValarray*)

Corresponding GCS command: DFH?

Get the home position in working units for *szAxes*.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pdValarray home positions of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qDIO** (long *ilD*, const char* *szChannels*, BOOL * *pbValarray*)

Corresponding GCS command: DIO?

Get the states for *szChannels* digital input channel(s).

Arguments:

ilD ID of controller
szChannels string with the identfiers of the digital input lines of the 26-pin IDC connector (J5) on the C-843 board (A to H). They can be brought out of the PC housing using an adapter bracket with a sub-D 25f connector (included with C-843). If "" or **NULL** all channels are affected.
pbValarray states of digital input channel, **TRUE** if "HIGH", **FALSE** if "LOW"

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qDRC** (long *ilD*, const long* *iRecTableIdArray*, char* *sRecSourceId*, long* *iRecOptionArray*, long* *TriggerOption*, long *iArraySize*)

Corresponding command: DRC?

Returns the data recorder configuration for the queried record table. The configuration can be changed with C843_DRC(). The recorded data can be read with C843_qDRR().

Trigger options for recording can be set with C843_DRT() and read with C843_qDRT().

Arguments:

ilD ID of controller
iRecTableIdArray array of the record table IDs.
sRecSourceId string to receive the record source (axis identifier).
iRecOptionArray array to receive the record option, i.e. the kind of data to be recorded, for the possible options see C843_DRC()
TriggerOption not used
iArraySize the size of the arrays *iRecTableIdArray*, *iRecOptionArray*

Returns:

TRUE if no error, FALSE otherwise

BOOL C843_FUNC_DECL **C843_qDRR** (long *iID*, const long* *piRecTableIdsArray*, long *iNumberOfRecChannels*, long *iOffset*, long *nrValues*, double** *pdValArray*, char* *szGcsArrayHeader*, long *iGcsArrayHeaderMaxSize*)

Corresponding command: DRR?

Reading of the last recorded Data Set.

Reading can take long depending on the number of points to be read!

It is possible to read the data while recording is still in progress.

With *nrValues* = -1 all points of the last record are read.

If *piRecTableIdsArray* is empty, the data from all tables with non-zero record option (see C843_DRC()) is read.

Step response measurements done with C843_STE() can also be read with C843_qSTE().

With C843_qHDR() you will obtain a list of available record options and trigger options and information about additional parameters and commands regarding data recording.

Note that the data recorder shares the 32,256 points of volatile memory provided on the C-843 card (referred to as “external RAM” in the MotionProcessor Users Guide) with the multi-axis motion profiles which can be created by the User Profile Mode commands (C843_Upx() functions). It may be necessary to free memory occupied by user-defined motion profiles using C843_UPC() to have enough memory for data recording.

Note that with some older C-843 hardware models, you can not use the data recorder if the digital output lines have been set with C843_DIO() before any data-recorder-related function was called. To use the data recorder, reconnect the C-843.

For detailed information see "Data Recording" in the C-843 GCS Commands manual (SM149E).

Arguments:

iID ID of controller
piRecTableIdsArray IDs of data recorder tables
iNumberOfRecChannels number of data recorder tables to read
iOffset index of first value to be read (starts with index 1)
nrValues number of values to read
pdValarray pointer to internal array to store the data; data from all tables read will be placed in the same array with the values interspersed; the DLL will allocate enough memory to store all data, call C843_GetAsyncBufferIndex() to find out how many data points have already been transferred
szGcsArrayHeader buffer to store the GCS array header
iGcsArrayHeaderMaxSize size of the buffer to store the GCS Array header, must be given to prevent buffer overflow

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qDRR_SYNC** (long *iID*, long *iRecTableIdArray*, long *iOffset*, long *nrValues*, double* *pdValArray*)

Corresponding command: DRR?

Returns the data points of the last recorded data set.

For detailed information regarding data recording see the notes in C843_qDRR().

Arguments:

iID ID of controller
iRecordTableIdArray Id of the record table.
iOffset The start point in the specified record table (starts with index 1)
nrValues The number of values to read.
pdValArray array to receive the values

Returns:

TRUE if no error, FALSE otherwise

BOOL C843_FUNC_DECL **C843_qDRT** (long *iID*, const long* *iRecTableIdArray*, long* *TriggerOptionArray*, char* *sValueArray*, long *iArraySize*, long *iValueBufferLength*)

Corresponding command: DRT?

Returns the current trigger source setting for the given data recorder table.

Arguments:

iID ID of controller
iRecTableIdArray array of the record table IDs
TriggerOptionArray array to receive the trigger source, see C843_DRT() for details.
sValueArray array to receive the trigger-source-dependent values
iArraySize the size of the arrays *iRecTableIdArray* and *TriggerOptionArray*
iValueBufferLength is the size of *sValueArray*

Returns:

TRUE if no error, FALSE otherwise

BOOL C843_FUNC_DECL **C843_qEGE** (long *iID*, const char* *szAxes*, BOOL* *valarray*

Corresponding GCS command: EGE?

Gets electronic gearing enable status for *szAxes.*

Arguments:

iID ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pbValarray modes of the specified axes, **TRUE** for "on", **FALSE** for "off"

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qERR** (long *iID*, long* *pnError*)

Corresponding GCS command: ERR?

Get the error state of the controller. It is safer to call **C843_GetError**() (p.27) because this will also return the internal error state of the library.

Arguments:

iID ID of controller
pnError error code of the controller

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qFED** (long *iID*, const char* *szAxes*, long* *iEdgeIDArray*, long* *iParArray*)

Corresponding GCS command: FED?

Get settings of the last commanded search for a signal edge.

Arguments:

iID ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
iEdgeIDArray identifier of edge to be searched
iParArray polarity definition of edge

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qFES** (long *ilD*, const char* *szAxes*, BOOL* *pbValarray*)

Corresponding GCS command: FES?

Get status of search for a signal edge.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pbValarray result of search (1 = success, 0 = failure)

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qFRF** (long *ilD*, const char* *szAxes*, BOOL* *pbValarray*)

Corresponding GCS command: FRF?

Get results of last call of the fast referencing functions **C843_FRF**() (p. 26), **C843_FNL**() (p. 25) or **C843_FPL**() (p. 26. If still referencing, or no fast reference move was started since startup of library, the result is 0. Call **C843_qREF**() (p. 44) to see which axes have a reference. For fast referencing of multiple axes call **C843_FRF**() (p. 26) for axes with reference or **C843_FNL**() (p. 25) or **C843_FPL**() (p. 26) for axes without reference. Call **C843_IsReferencing**() to find out if there are axes (still) referencing.

Note: To reference axes one after the other, use **C843_REF**() (p.51), **C843_MNL**() (p.33) or **C843_MPL**() (p.34).

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pbValarray 1 if successful, 0 if reference move failed, has not finished yet or axis has no reference

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qHDR** (const long *ilD*, char* *Buffer*, long *maxlen*)

Corresponding **command:** HDR?

Lists a help string which contains all information available for data recording (record options and trigger options, information about additional parameters and commands regarding data recording).

For more information regarding data recording see the notes in C843_qDRR().

Arguments:

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

Returns:

TRUE if no error, FALSE otherwise

BOOL C843_FUNC_DECL **C843_qHLP** (const long *ilD*, char* *buffer*, long *maxlen*)

Corresponding command: HLP?

Read in the help string provided by the C-843 GCS DLL. The answer is quite long (up to 3000 characters) so be sure to provide enough space!.

Arguments:

ilD ID of controller
buffer buffer to receive the string read in from the C-843 GCS DLL, 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 C843_FUNC_DECL **C843_qHPA** (const long *ilD*, char* *buffer*, long *maxlen*)

Corresponding **command:** HPA?

Lists a help string which contains all parameters provided by the C-843 GCS DLL with short descriptions. See “Motion Parameters” beginning on p. 61 for parameter handling and for an appropriate list of all parameters available for C-843 controller cards.

Arguments:

ilD ID of controller
szBuffer buffer to receive the string read in from the C-843 GCS DLL, lines are separated by '\n' ("line-feed")
iBufferSize size of *szBuffer*, must be given to avoid buffer overflow.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qIDN** (long *ilD*, char * *buffer*, long *maxlen*)

Corresponding GCS command: *IDN?

Get identification string of the controller.

Arguments:

ilD ID of controller
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 C843_FUNC_DECL **C843_qJAX** (long *iId*, const long* *iJoystickIDsArray*, const long* *iAxesIDsArray*, long *iArraySize*, char* *szAxesBuffer*, long *iBufferSize*)

Corresponding command: JAX?

Get axis controlled by a joystick which is connected to the PC.

See "Joystick Control" in the C-843 GCS Commands manual (SM149E) for details.

Arguments:

ilD ID of controller
iJoystickIDsArray array with joystick devices connected to the PC
iAxesIDsArray array with IDs of the joystick axes
iArraySize size of arrays
szAxesBuffer buffer to receive the string read in from controller; will contain axis IDs of axes associated with corresponding joystick axis
iBufferSize size of *buffer*, must be given to avoid buffer overflow.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qJON** (long *ilD*, const long* *iJoystickIDsArray*, BOOL* *pbValueArray*, long *iArraySize*)

Corresponding command: JON?

Get activation state of the given joystick which is connected to the PC. See also C843_JON().

See "Joystick Control" in the C-843 GCS Commands manual (SM149E) for details.

Arguments:

ilD ID of controller

iJoystickIDsArray array with joystick devices connected to the PC
pbValueArray pointer to array to receive the joystick enable states (0 for deactivate, 1 for activate)
iArraySize size of arrays

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qLIM** (long *ilD*, const char* *szAxes*, BOOL * *pbValarray*)

Corresponding command: `LIM?`

Check if the given axes have limit switches

Arguments:

ID ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pbValarray array for limit switch info: **TRUE** if axis has limit switches, **FALSE** if not

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qMAS** (long *ilD*, const char* *szAxes*, char* *szMasters*)

Corresponding command: `MAS?`

Get the electronic gearing master axes for *szAxes*. The second string is filled with the corresponding master axes. e.g. *szMasters* [1] is the master for *szAxes* [1].

See C843_EGE(9 for further detailon electronic gearing.

Arguments:

ilD ID of controller
szAxes string with “slave” axes
szMasters string to be filled with the master axes for the slaves in *szAxes*

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qMOV** (long *ilD*, const char* *szAxes*, double * *pdValarray*)

Corresponding GCS command: `MOV?`

Read the commanded target positions for *szAxes*.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pdValarray array to be filled with target positions of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qONT** (long *ilD*, const char* *szAxes*, BOOL * *pbValarray*)

Corresponding GCS command: ONT?

Check if *szAxes* have reached target position.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pbValarray array to be filled with current on-target status of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qPOS** (long *ilD*, const char* *szAxes*, double * *pdValarray*)

Corresponding GCS command: POS?

Get the positions of *szAxes*.

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pdValarray positions of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qREF** (long *ilD*, const char* *szAxes*, BOOL * *pbValarray*)

Corresponding GCS command: REF?

Check if the given axes have a reference

Arguments:

ilD ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pbValarray **TRUE** if axis has a reference, **FALSE** if not

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qRON** (long *ID*, const char* *szAxes*, BOOL * *pbValarray*)

Corresponding command: RON?

Gets reference mode for given axes. See **C843_RON**() (p.51) for a detailed description of reference mode.

Arguments:

ID ID of controller
szAxes string with axes
pbValarray array to receive reference modes for the specified axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qRTR** (long *ilD*, long* *iRecordTableRate*)

Corresponding command: RTR?

Gets the current record table rate, i.e. the number of servo-loop cycles used in data recording operations.

Arguments:

ilD ID of controller
iRecordTableRate variable to be filled with the record table rate

Returns:

TRUE if no error, FALSE otherwise

BOOL C843_FUNC_DECL **C843_qSAI** (long *ilD*, char * *axes*, long *maxlen*)

Corresponding GCS command: `SAI?`

Get connected axes. Each character in the returned string is an axis identifier for one connected and configured axis. At startup of the library **C843_qSAI**() (p.45) will return an empty string. You must configure the stages with calls to **C843_CST**() (p.19) and **C843_INI**() (p. 30) before you can use them. See "Functions for Initialization of the C-843 GCS DLL" (p. 12) for an example of how to setup the C-843 library.

Arguments:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qSAI_ALL** (long *ilD*, char * *axes*, long *maxlen*)

Corresponding GCS command: `SAI? ALL`

Get all possible axes, and not only all connected and configured axes as returned by C843_qSAI(). Each character in the returned string is an axis identifier for one possible axis.

Arguments:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qSMO** (long *ilD*, const char* *szAxes*, long* *pnValarray*)

Corresponding GCS command: `SMO?`

Get the motor output.

Arguments:

ilD ID of controller
szAxes string with axes, if **""** or **NULL** all axes are affected.
pnValarray motor output for the specified axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qSPA** (long *iID*, const char* *szAxes*, const long * *iCmdarray*, double * *dValarray*, char * *szStageNames*, long *iMaxNameSize*)

Corresponding GCS command: SPA?

Read parameters for *szAxes*. For each desired parameter you must specify an axis in *szAxes* and a parameter ID in the corresponding element of *iCmdarray*. The most important parameter IDs are the servo loop parameters listed below. For a complete list, see “Parameter List”, p. 62.

- 0x1 for P-Term
- 0x2 for I-Term
- 0x3 for D-Term
- 0x4 for I-Limit (integration limit)
- 0x5 for VFF (velocity feed forward)
- 0x6 for Kout (output scale factor)
- 0x7 for Bias (motor bias)
- 0x8 for the maximum position error

Arguments:

iID ID of controller
szAxes axis for which the parameter should be read
iCmdarray IDs of parameter
dValarray array to be filled with the values for the parameters
szStageNames string when needed, set to **NULL** if a numeric values is used
iMaxNameSize size of *szStageNames*

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_INVALID_SPA_CMD_ID *one* of the IDs in *iCmdarray* is not valid, must be one of {1,2,3}

BOOL C843_FUNC_DECL **C843_qSRA** (long *iID*, const char* *szAxes*, double* *pdValarray*)

Corresponding command: SRA?

Gets the electronic gear ratio for *szAxes*. See C843_EGE() for further details regarding electronic gearing.

Parameters:

iID ID of controller
szAxes string with “slave” axes
pdValarray array to be filled with ratios for the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qSRG** (long *iID*, const char* *szAxes*, const long* *iCmdarray*, long* *iValarray*)

Corresponding GCS command: SRG?

Get the content of the status registers for *szAxes*.

Arguments:

iID ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
iCmdarray register to be queried, if "" or **NULL** all registers are affected:
1 = Event Status register
2 = Activity Status register
3 = Signal Status register
4 = Signal Sense mask
For detailed descriptions of the registers see the Motion Processors User Guide on the C-843 CD. Note that the states of the digital input and output lines located on the J8 (“All-axes”) connector on the C-843 board are provided in the Signal Status register (AxisIn and AxisOut bits).

iValarray register content of the specified axes for the given register; see MotionProcessor_UsersGuide.pdf on the C-843 CD for details.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qSTE** (long *iID*, char *cAxis*, long *iOffset*, long *nrValues*, double * *pdValarray*)

Corresponding GCS command: STE?

Get the recorded positions of a step response. The controller will move the given axis to the target position and record 32256 position values from start. Call **C843_STE**() (p.56) to start the step response measurement.

Arguments:

iID ID of controller
cAxis axis for which the step response was recorded
iOffset index of first value to be read, the first stored value has index 0
nrValues number of values to be read. At most 32256 positions are stored.
pdValarray Array to store the position values. Caller is responsible for providing enough space for *nrValues* doubles

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_INVALID_ARGUMENT *the* combination of *iOffset* and *nrValues* specifies values out of range

BOOL C843_FUNC_DECL **C843_qSVO** (long *iID*, const char* *szAxes*, BOOL * *pbValarray*)

Corresponding GCS command: SV0?

Get the servo mode for *szAxes*

Arguments:

iID ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pbValarray modes of the specified axes, **TRUE** for "on", **FALSE** for "off"

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qTIO** (long *iID*, long* *pINr*, long* *pONr*)

Corresponding GCS command: TIO?

Get the number of digital input and output channels installed. Call **C843_GetInputChannelNames**() (p.27) and **C843_GetOutputChannelNames**() (p.28) to find out how to address them

C843_qTIO() queries the number of digital IO lines on the 26-pin IDC connector (J5). The IO lines on the 16-pin IDC connector (J8) are not contained in the C843_qTIO() response. See “External Triggering / Signaling” in the C-843 GCS Commands manual for more information.

Arguments:

iID ID of controller
pINr pointer for storing the number of digital input channels installed
pONr pointer for storing the number of digital output channels installed

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qTMN** (long *ilID*, const char* *szAxes*, double * *pdValarray*)

Corresponding GCS command: TMN?

Get the low end of travel range of *szAxes* in working units.

Arguments:

ilID ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pdValarray minimum travel range of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qTMX** (long *ilID*, const char* *szAxes*, double * *pdValarray*)

Corresponding GCS command: TMX?

Get the high end of the travel range of *szAxes* in working units.

Arguments:

ilID ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pdValarray maximum travel range of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qTNR** (long *ilID*, long* *iNrOfTables*)

Corresponding command: TNR?

Returns the number of data recorder tables.

For more information regarding data recording see the notes in C843_qDRR.

Arguments:

ilID ID of controller
iNrOfTables variable to receive number of data recorder tables

Returns:

TRUE if no error, FALSE otherwise

BOOL C843_FUNC_DECL C843 **C843_qTRO** (long *ilID*, const long* *iTriggerLinesArray*, BOOL* *pbValarray*, long *iArraySize*)

Corresponding command: TRO?

Gets trigger output-mode enable-status for given trigger output line (the trigger output configuration is made with C843_CTO()).

Arguments:

ilID ID of controller
iTriggerLinesArray is an array with the trigger output lines of the C-843 card. See C843_CTO() for details.
pbValarray pointer to array to receive modes of the specified trigger lines: **TRUE** for "enabled", **FALSE** for "disabled"
iArraySize number of trigger lines

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qTVI** (long *ilD*, char * *axes*, long *maxlen*)

Corresponding GCS command: TVI?

Get valid characters for axes. Each character in the returned string is a valid axis identifier that can be used to "name" an axis.

Arguments:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qUPA** (long *ilD*, const char* *szClusters*, const long* *iCmdarray,* long* *iPararray*)

Corresponding GCS command: UPA?

User Profile Mode; Gets the numbers of the Blocks from which the specified Datasets originated.

Arguments:

iID ID of controller
szClusters string with clusters
iCmdarray indices of datasets in clusters
iPararray array to receive numbers of blocks from which corresponding datasets originated. Values of -1 indicate that no block has been activated for the corresponding Dataset index.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qUPB** (long *ilD*, const char* *szClusters*, const long* *iCmdarray,* const long* *iPararray,* long* *iValarray*)

Corresponding GCS command: UPB?

User Profile Mode: Reads Data Block configuration

Arguments:

iID ID of controller
szClusters string with clusters
iCmdarray indices of corresponding block in each cluster
iPararray parameter ID of corresponding parameter to read
iValarray array to receive values of parameters being queried

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qUPC** (long *iID*, char* *szClusters*, char* *szAxes,* long* *iCmdarray*, long* *iPararray*)

Corresponding GCS command: UPC?

User Profile Mode: Read cluster configuration.

Arguments:

iID ID of controller
szClusters string with clusters; if "" or null, all defined Clusters are queried.
szAxes string to receive axes assigned to corresponding clusters (must have sufficient length)
iCmdarray array to receive number of datasets in the corresponding clusters
iPararray array to receive the lengths of the datasets in the corresponding clusters

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qUPD** (long *iID,* const char* *szClusters*, const long* *iCmdarray*, const long* *iPararray,*double* *dValarray*)

Corresponding GCS command: UPD?

User Profile Mode: Read values from datasets.

Arguments:

iID ID of controller
szClusters string with clusters
iCmdarray array with indices of blocks of corresponding clusters
iPararray array with numbers of datasets in corresponding blocks of corresponding clusters
dValarray array to receive values from queried datasets; size must be ≥ sum of the lengths of the queried datasets (max. 5 x length of *szClusters*)

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qVEL** (long *iID*, const char* *szAxes*, double * *pdValarray*)

Corresponding GCS command: VEL?

Get the velocities of *szAxes*.

Arguments:

iID ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pdValarray array to be filled with the velocities of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qVER** (long *iID*, char* *szVersion*, int *iBufferSize*)

Corresponding command: VER?

Reports the versions of drivers and libraries used.

Arguments:

iID ID of controller
szVersion buffer for storing the string read in
iBufferSize size of *szVersion*, must be given to avoid buffer overflow.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_qVST** (long *ilD*, char * *buffer*, long *maxlen*)

Corresponding GCS command: VST?

Lists the names of stages selectable by C843_CST().

The list comprises the content of the stage databases (PIStages2.dat, C843Userstages2.dat, M-xxx.dat files) used by the C843_GCS_DLL. See “Parameter Databases” on p. 71 for more information.

Arguments:

ilD ID of controller
buffer buffer for storing the string read in from controller, lines are separated by \n (line feed)
maxlen size of *buffer*, must be given to avoid a buffer overflow.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_REF** (long *ilD*, const char* *szAxes*)

Corresponding GCS command: REF

Reference move of *szAxes*. For fast (synchronous) referencing of multiple axes call **C843_FRF**() (p. 26) instead. Call **C843_IsReferencing**() (p.31) to find out if the axes are still moving and **C843_GetRefResult**() (p.29) to get the results from the controller. The controller will be "busy" while referencing, so most other commands will cause a **PI_CONTROLLER_BUSY** error. Use **C843_STP**() (p.56*)* to stop it.

Note: Calling the **C843_Ref** function resets the current position. That means moving to the same position using the **MOV**() command (absolute move) before and after a call of this function may move the stage to a different physical position! You should call this function only once after a call of the **INI**() command.

Arguments:

ilD ID of controller
szAxes string with axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_RemoveStage** (long *ilD*, char * *szStageName*)

Removes the stage with the given name from the *C843UserStages2.dat* file, which contains the user-defined stages.

Arguments:

ilD ID of controller
szStageName the stage name as string.

Returns:

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

BOOL C843_FUNC_DECL **C843_RON** (long *ilD*, const char* *szAxes*, const BOOL * *pbValarray*)

Corresponding command: RON

Sets reference mode for given axes.

If the reference mode of an axis is ON, the axis must be driven to the reference switch (**C843_REF**() (p.51), **C843_FRF**() (p. 26) or, if no reference switch is available, to a limit switch (using **C843_MPL**() (*p.*34), **C843_FPL**() (p. 26), **C843_MNL**() (*p.*33) or **C843_FNL**() (p. 25)) before any other motion can be commanded.

If reference mode is OFF, no referencing is required for the axis. Only relative moves can be commanded (**C843_MVR**() (p.36)), unless the actual position is set with **C843_POS**(). Afterwards, relative and absolute moves can be commanded.

For stages with neither reference nor limit switch, reference mode is automatically OFF.

Arguments:

ilD ID of controller
szAxes string with axes
pbValarray reference modes for the specified axes

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_CNTR_STAGE_HAS_NO_LIM_SWITCH if the axes have no reference or limit switches, and reference mode can not be switched ON

BOOL C843_FUNC_DECL **C843_RTR** (long *ilD*, long *iRecordTableRate*)

Corresponding command: RTR

Sets the record table rate, i.e. the number of servo-loop cycles to be used in data recording operations. Settings larger than 1 make it possible to cover longer time periods with a limited number of points.

C843_RTR() changes the value of the Data Recorder Table Rate parameter (ID 0x16000000), can also be changed with C843_SPA().

Arguments:

ilD ID of controller
iRecordTableRate is the record table rate to be used (unit: number of servo-loop cycles), must be larger than zero

Returns:

TRUE if no error, FALSE otherwise

BOOL C843_FUNC_DECL **C843_SAI** (long *ilID*, const char* *szOldAxes*, const char* *szNewAxes*)

Corresponding GCS command: SAI

Rename connected axes. *szOldAxes[index]* will be set to *szNewAxes[index]*. User can set the "names" of axes with this function. The characters in *szNewAxes* must not be in use for any other existing axes and must each be one of the valid identifiers. All characters in *szNewAxes* will be converted to uppercase letters. To find out which characters are valid, call **C843_qTVI**() (p.49). Only the **last** occurence of an axis identifier in *szNewAxes* will be used to change the name.

Arguments:

ilID ID of controller
szOldAxes old axis identifiers
szNewAxes new identifiers for the axes

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_INVALID_AXIS_IDENTIFIER if the any of the characters are not valid
PI_UNKNOWN_AXIS_IDENTIFIER if *szOldAxes* contains any unknown axes
PI_AXIS_ALREADY_EXISTS if one of *szNewAxes* is already in use
PI_INVALID_ARGUMENT if `szOldAxes` and `szNewAxes` have different lengths or if a character in `szNewAxes` is used for more than one old axis

BOOL C843_FUNC_DECL **C843_SetErrorCheck** (long *ilID*, BOOL *bErrorCheck*)

Set error check mode of the library. With this call you can specify whether the library should check (with "ERR?") the error state of the C-843 after sending a command. This will slow down the communication, so if you need a high data rate, switch off the error checking and call **C843_GetError**() (p.27) by yourself when there is time to do so. You can use the permanent error check to debug your application. At startup this mode is switched on.

Arguments:

ilID ID of controller
bErrorCheck new state, **TRUE** to switch on error check, **FALSE** to switch it off.

Returns:

the old state before this call

BOOL C843_FUNC_DECL **C843_SetQMC** (long *ilID*, BYTE *bCmd*, BYTE *bAxis*, int *Param*)

Sends a QMC command (command set of the motion processor) with one argument (16 and 32 bit) to the C-843 controller.

Arguments:

ilID ID of controller
bCmd the QMC command.
bAxis the axis (The first axis is axis 0).
Param the QMC argument.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_SetQMCA** (long *iID*, BYTE *bCmd*, BYTE *bAxis*, WORD *Param1*, int *lParam2*)

Sends a QMC command (command set of the motion processor) with two arguments (2 * 32-bit data words) to the C-843 controller.

Arguments:

iID ID of controller
bCmd the QMC command.
bAxis the axis (The first axis is axis 0).
Param1 the first data word.
Param2 the second data word.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_SMO** (long *iID*, const char* *szAxes*, const long* *pnValarray*)

Corresponding GCS command: SMO

Set control value for the motor output directly to move the axis. Note that this is basically a velocity setting, not a position setting. Trajectory generator and servo filter are omitted.

Servo must be switched off (open-loop operation; see **C843_SVO**() (p.57)) when using this function.

CAUTION: Limit switches are deactivated so that the stage can run into the hard stop. This can cause damage to equipment.

Arguments:

iID ID of controller
szAxes string with axes
pnValarray array with motor output parameters. All must be in [-32767 to 32767]

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_SPA** (long *iID*, const char* *szAxes*, const long* *iCmdarray*, const double* *dValarray*, const char* *szStageNames*)

Corresponding GCS command: SPA

Set parameters for *szAxes*. For each parameter you must specify an axis in *szAxes* and a parameter ID in the corresponding element of *iCmdarray*. The most important parameter IDs are listed below. For a complete list, see ”Parameter List” p. 62.

- 0x1 for P-Term
- 0x2 for I-Term
- 0x3 for D-Term
- 0x4 for I-Limit (integration limit)
- 0x5 for VFF (velocity feed forward)
- 0x10 maximum velocity
- 0x11 acceleration
- 0x6 for Kout (output scale factor)
- 0x7 for Bias (motor bias)
- 0x8 for the maximum position error

Whenever you start working, first C843_CST() and C843_INI() must be called: C843_CST() loads stage parameters suitable for your hardware from a stage database, and C843_INI() writes the loaded values to the controller to initialize the motion control chip on the C-843 board. Afterwards, you can change parameters using C843_SPA(). Note that some parameters should normally not be changed (see marks in the parameter list).

Note that all parameter changes with C843_SPA() are temporarily (done in C843_GCS_DLL and in the motion processor of the C-843 board). To store parameter values, save them to the

C843UserStages2.dat stage database (see "Functions for User-Defined Stages" on p. 17 for more information).

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

With C843-qHPA() you can obtain a list of the available parameters and their IDs.

Note:

If the same axis has the same parameter ID more than once, only the **last** value will be used. For example `C843_SPA(id, "111", {0x1, 0x1,0x2}, {100, 200, 150})` will set the P-term of '`0x1`' to 200 and the I-term to 150. Some stages have additional parameters which can be set with SPA. For **Userdefined stages** see (p. 17).

Arguments:

iID ID of controller
szAxes axes for which the corresponding parameter should be set
iCmdarray IDs of parameters
dValarray array with the values for the parameters
szStageNames string when needed, set to **NULL** if numeric values are used

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_INVALID_SPA_CMD_ID *one* of the IDs in *iCmdarray* is not valid.

BOOL C843_FUNC_DECL **C843_SRA** (long *iID*, const char* *szAxes*, double* *pdValarray*)

Corresponding command: SRA

Gear ratio setting for electronic gearing: the given ratio is applied when electronic gearing is enabled for the *szAxes* which are then the slaves. The ratio is defined as
Ratio = Travel of Master / Travel of Slave

See C843_EGE() for further details regarding electronic gearing.

Parameters:

iID ID of controller
szAxes string with axes
pdValarray array with ratios for the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_STE** (long *ilD*, const char* *cAxis*, double* *dOffset*)

Corresponding GCS command: STE

Starts performing a step and recording up to 32,640 position values as the axis response.

A "step" is a motion pos. a → pos. b, performed relative to the current position.

The recorded data can be read with C843_qSTE(), C843_qDRR() or C843_qDRR_sync().

C843_STE() automatically changes the record option of the first data recorder table to "actual position" (2) (see also C843_DRC()). The configuration of record tables 2 to 4 is not changed by C843_STE(). This makes it possible to record additional data, but these data can only be read with C843_qDRR() or C843_qDRR_sync().

The number of points which are recorded with C843_STE() depends on the C843_DRC() settings: the points available for recording are in equal shares allocated to the tables with non-zero record options (for the total number of points to allocate ask C843_qSPA() with parameter 0x16000200, maximum value is 32,256).

C843_STE() automatically resets the data recorder sample period to 1 (see also C843_RTR() or C843_SPA() parameter 0x16000000).

If you do not want to deal with the restrictions induced by C843_STE(), you can use C843_MVR() instead to make a relative step move and configure recording according to your requirements.

Note that the data recorder shares the 32,256 points of volatile memory provided on the C-843 card (referred to as “external RAM” in the MotionProcessor Users Guide) with the multi-axis motion profiles which can be created by the User Profile Mode commands (C843_Upx() functions). It may be necessary to free memory occupied by user-defined motion profiles using C843_UPC() to have enough memory for data recording.

Note that with some older C-843 hardware models, you can not use the data recorder if the digital output lines have been set with C843_DIO() before any data-recorder-related command was sent. To use the data recorder, reconnect the C-843.

Motion commands like C843_STE() are not allowed when the joystick is active for the axis.

Arguments:

ilD ID of controller
cAxis axis for which the step response will be recorded
dOffset position offset for *cAxis*

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_STP** (long *ilD*)

Corresponding GCS command: STP

Stop all axes abruptly.

Arguments:

ilD ID of controller

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_SVO** (long *iID*, const char* *szAxes*, const BOOL * *pbValarray*)

Corresponding GCS 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".

With servo OFF only direct motor output (velocity-related) is possible (see C843_SMO(), p. 54).

Stages with brake: The brake is activated automatically when the servo is switched off with C843_SVO(), and deactivated when the servo is switched on.

CAUTION

Before setting servo-control off make sure that the stage can not perform unwanted motion in servo-off mode. Unwanted motion could cause irreparable damage to the stage and the application setup.

Setting the brake with C843_BRA() does not affect the servo state of the axis. I.e. if you activate the brake, the servo remains on so that the motor may work against the brake which can cause overheating. In this case, it may be necessary to switch the servo off temporarily. Do not deactivate the brake when the servo is switched off! Otherwise unwanted motion can occur.

Arguments:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_TRO** (long *iID*, const long* *iTriggerLinesArray*, const BOOL* *pbValarray*, long *iArraySize*)

Corresponding command: TR0

Enables or disables the TRrigger Output mode which was set with C843_CTO() for the given digital output line. If *pbValarray[index]* is **FALSE** the mode is "off", if **TRUE** it is set to "on".

Arguments:

iID ID of controller
iTriggerLinesArray is an array with the digital output lines located on the J8 (“All-axes”) connector on the C-843 board (digital output from the motion processor, TTL, max. 5 mA).
with C-843.21: can be 1 and 2
with C-843.41: can be 1 to 4
The lines can be brought out of the PC housing using an adapter bracket with a sub-D 15m connector (included with C-843).
pbValarray pointer to boolean array with modes for the specified trigger lines, **TRUE** for "on", **FALSE** for "off"
iArraySize number of trigger lines

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_TranslateError** (int *errNr*, char * *szBuffer*, const int *maxlen*)

Translate error number to error message.

Arguments:

errNr number of error, as returned from **C843_GetError**() (p.27).
szBuffer pointer to buffer that will store the message
maxlen size of the buffer

Returns:

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

BOOL C843_FUNC_DECL **C843_UPA** (long *iID*, const char* *szClusters*, const long* *iCmdarray*)

Corresponding GCS command: UPA

User Profile Mode: Activate block.

See separate Technical Note A000T0014_100_UserProfileModeSoftware and the C-843 GCS Command manual (SM149E) for more information.

Note that with some older C-843 hardware models, you can not use the User Profile Mode if the digital output lines have been set with C843_DIO() before any User-Profile-Mode-related command was sent. To use the User Profile Mode, reconnect the C-843.

Arguments:

iID ID of controller
szClusters string with clusters
iCmdarray indices of blocks of corresponding clusters to be activated (swapped in)

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_UPB** (long *iID*, const char* *szClusters*, const long* *iCmdarray,* , const long* *iPararray,* const long* *iValarray*)

Corresponding GCS command: UPB

User Profile Mode: Create or modify Block or delete all Blocks.

See separate Technical Note A000T0014_100_UserProfileModeSoftware and the C-843 GCS Command manual (SM149E) for more information.

Note that with some older C-843 hardware models, you can not use the User Profile Mode if the digital output lines have been set with C843_DIO() before any User-Profile-Mode-related command was sent. To use the User Profile Mode, reconnect the C-843.

Arguments:

iID ID of controller
szClusters string with clusters
iCmdarray indices of blocks to be created or modified for corresponding clusters
iPararray parameter IDs of parameters of corresponding blocks to be set
iValarray values to be assigned to corresponding parameters

Note: If block index, parameter ID and value are all -1, all blocks of the corresponding Cluster are deleted.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_UPC** (long *iID*, const char* *szAxes,* const char* *szClusters*, const long* *iCmdarray*, const long* *iPararray*)

Corresponding GCS command: UPC

User Profile Mode: Create Cluster or delete all Clusters and Blocks.

See separate Technical Note A000T0014_100_UserProfileModeSoftware and the C-843 GCS Command manual (SM149E) for more information.

Note that with some older C-843 hardware models, you can not use the User Profile Mode if the digital output lines have been set with C843_DIO() before any User-Profile-Mode-related command was sent. To use the User Profile Mode, reconnect the C-843.

Arguments:

iID ID of controller
szAxes string with axes
szClusters string with clusters to be assigned to these axes
iCmdarray array with maximum numbers of datasets in corresponding clusters

iPararray array with lengths of corresponding datasets

Note: If *szAxes* and *szClusters* are both "$" and the corresponding values are both -1, all Clusters and all Blocks, if any, are deleted

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_UPD** (long *iID,* const char* *szClusters*, const long* *iCmdarray*, const long* *iPararray, const* double* *dValarray*)

Corresponding GCS command: UPD

User Profile Mode: Write data to one dataset in Block of Cluster.

See separate Technical Note A000T0014_100_UserProfileModeSoftware and the C-843 GCS Command manual (SM149E) for more information.

Note that with some older C-843 hardware models, you can not use the User Profile Mode if the digital output lines have been set with C843_DIO() before any User-Profile-Mode-related command was sent. To use the User Profile Mode, reconnect the C-843.

Arguments:

iID ID of controller
szClusters string of length 1 with cluster
iCmdarray array with index of block to be written to in first cell
iPararray array with number of data set in block in first cell
dValarray array with values for dataset to be written

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_UPR** (long *iID*, const char* *szAxes,* const char* *szClusters*, const long* *iCmdarray*)

Corresponding GCS command: UPR

User Profile Mode: Start motion.

See separate Technical Note A000T0014_100_UserProfileModeSoftware and the C-843 GCS Command manual (SM149E) for more information.

Note that with some older C-843 hardware models, you can not use the User Profile Mode if the digital output lines have been set with C843_DIO() before any User-Profile-Mode-related command was sent. To use the User Profile Mode, reconnect the C-843.

Arguments:

iID ID of controller
szAxes string with axes
szClusters string with clusters, each assigned to corresponding axis
iCmdarray offset of Dataset in Cluster to start with

Returns:

TRUE if successful, **FALSE** otherwise

BOOL C843_FUNC_DECL **C843_VEL** (long *ilD*, const char* *szAxes*, const double * *pdValarray*)

Corresponding GCS command: VEL

Set the velocities of szAxes.

The maximum value which can be set with C843_VEL() is given by the Maximum closed-loop velocity parameter, ID 0xA (can be changed with C843_SPA()).

During vectorial moves started with C843_MVE(), velocities, accelerations and decelerations will be calculated to ensure that all axes follow the path. The current settings for velocity, acceleration and deceleration define the maximum possible values, and the slowest axis determines the resulting velocities.

Arguments:

ilD ID of controller
szAxes string with axes
pdValarray velocities for the axes

Returns:

TRUE if successful, **FALSE** otherwise

5. Motion Parameters

5.1. Parameter Handling

The C843_GCS_DLL supports a mechanism which mirrors the hardware basics of the connected stage and the required closed-loop control settings in parameters. The parameter values have to be adjusted properly before initial operation of a stage. For that purpose, call C843_CST() and C843_INI() whenever you start working: C843_CST() loads stage parameters suitable for your hardware from a stage database, and C843_INI() writes the loaded values to the controller to initialize the motion control chip on the C-843 board. This is done on a per-axis basis. Note that PIMikroMove™ performs this procedure automatically upon stage selection. See “Parameter Databases” (p. 71) for the available stage databases.

With C843_qHPA() you can obtain a list of all available parameters with information about each (e.g. short descriptions). The current valid parameter values can be read with C843_qSPA(). Using C843_qVST() you can list the names of stages selectable by C843_CST() (i.e. the stages for which parameter sets are available in the stage databases).

Using the "general" modification function C843_SPA(), parameters can be changed. In addition to this general modification command, there are some commands which change certain specific parameters (see table in “Parameter List” below). Note that all that parameter changes are temporarily (done in C843_GCS_DLL and in the motion processor of the C-843 board). To store parameter values, save them to the C843UserStages2.dat stage database (see “Functions for User-Defined Stages” on p. 17 for more information).

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

The interrelation of the hardware-dependent parameters 0x15, 0x16, 0x17, 0x2F and 0x30 is described in "Travel Range Adjustment” (p. 68).

5.2. Parameter List

For additional information regarding most of the parameters listed below, see the User's Guide and the Programmer's Reference for the PMD Navigator MC2140CP Motion Processor which are on the C-843 CD.

Parameter ID (hexadecimal / decimal)	Data Type	Parameter Description	Changing with C843_SPA not recommended	Possible Values/Notes
0x1 / 1	FLOAT	P-term for position control		0 to 32767 Gives the P-term of the proportional-integral-derivative (PID) algorithm applied by the servo filter. See MotionProcessors User Guide for more information.
0x2 / 2	FLOAT	I-term for position control		0 to 32767 Gives the I-term of the proportional-integral-derivative (PID) algorithm applied by the servo filter. See MotionProcessors User Guide for more information.
0x3 / 3	FLOAT	D-term for position control		0 to 32767 Gives the D-term of the proportional-integral-derivative (PID) algorithm applied by the servo filter. See MotionProcessors User Guide for more information.
0x4 / 4	FLOAT	I-limit for position control		0 to 32767 Gives the integration limit for the accumulated error processed by the servo filter. See MotionProcessors User Guide for more information.
0x5 / 5	FLOAT	Velocity feed forward		0 to 32767 Gives the velocity feed forward term of the servo filter. See MotionProcessors User Guide for more information.
0x6 / 6	FLOAT	Output scaling factor		0 to 65536 Gives an output scale factor which is applied by the servo filter to produce the final motor output. See MotionProcessors User Guide for more information.
0x7 / 7	FLOAT	Output bias value		0 to 32767 Gives a bias value which is added by the servo filter to produce the final motor output. When an axis is subject to a net external force in one direction (such as a vertical axis pulled downward by gravity), the servo filter can compensate for it by adding a constant DC bias to the filter output. See MotionProcessors User Guide for more information.

Parameter ID (hexadecimal / decimal)	Data Type	Parameter Description	Changing with C843_SPA not recommended	Possible Values/Notes
0x8 / 8	FLOAT	Maximum position error (user unit)		0 to 32767 Used for stall detection. If the position error (i.e. the absolute value of the difference between current position and commanded position) in closed-loop operation exceeds the given maximum, the C843_GCS DLL sets error code -1024 (“Motion error”), the servo will be switched off automatically for the axis concerned, and motion of all axes is stopped immediately.
0x9 /9	FLOAT	Motor output limit	X	Gives a limiting value for the output of the servo filter. The motor output limit prevents the filter output from exceeding a boundary magnitude in either direction. If the filter produces a value greater than the limit, the motor command takes the limiting value. The motor limit applies only in closed-loop operation (servo on). It does not affect the motor command value set with C843_SMO() in open-loop operation. See MotionProcessors User Guide for more information.
0xA / 10	FLOAT	Maximum closed-loop velocity (user unit/s)	X	> 0 Gives the maximum value for parameter 0x49.
0xB / 11	FLOAT	Current closed-loop acceleration (user unit/s^2) also changed by C843_ACC()		Gives the current acceleration, limited by parameter 0x4A
0xC / 12	FLOAT	Current closed-loop deceleration (user unit/s^2) also changed by C843_DEC()		Gives the current deceleration, limited by parameter 0x4B
0xD / 13	FLOAT	Maximum allowed jerk (user unit/s^3)	X	Gives the maximum allowed jerk. Limits the slope of the acceleration. In S-curve profile mode, reducing the jerk will smoothen the motion.
0xE / 14	FLOAT	Numerator of the counts-per-physical-unit factor	X	1 to 2147483647 for each parameter. The counts-per-physical-unit factor determines the “user” unit for closed-loop motion commands. When you change this factor, all other parameters whose unit is based on the “user” unit are adapted automatically, e.g. closed-loop velocity and parameters regarding the travel range. Note: To customize your physical unit use C843_DFF() instead (see parameter 0x12).
0xF / 15	FLOAT	Denominator of the counts-per-physical-unit factor	X	

Parameter ID (hexadecimal / decimal)	Data Type	Parameter Description	Changing with C843_SPA not recommended	Possible Values/Notes
0x10 / 16	FLOAT	Output mode	X	0 = Analog 1 = PWM
0x11 / 17	FLOAT	Invert direction	X	-1 = invert direction 1 = do not invert
0x12 / 18	FLOAT	Scaling factor, also changed by C843_DFF()		This factor can be used to change the physical unit of the stage, e.g. a factor of 25.4 converts a physical unit of mm to inches. It is recommended to use C843_DFF() to change this factor.
0x13 / 19	FLOAT	Rotary stage	X	1 = rotary stage 0 = no rotary stage
0x14 / 20	FLOAT	Stage has a reference switch	X	1 = stage has a reference switch 0 = stage has no reference switch
0x15 / 21	FLOAT	MAX_TRAVEL_RANGE_POS The maximum travel in positive direction (user unit)	X	"Soft limit", based on the home (zero) position. If the soft limit is smaller than the position value for the positive limit switch (which is given by the sum of the parameters 0x16 and 0x2F), the positive limit switch can not be used for referencing. Can be negative.
0x16 / 22	FLOAT	VALUE_AT_REF_POS The position value at the reference switch position (user unit)	X	The position value which is to be set when the mechanics performs a reference move to the reference switch. Must be set even if no reference switch is present in the mechanics because it is used to to calculate the position values to be set after reference moves to the limit switches.
0x17 / 23	FLOAT	DISTANCE_REF_TO_N_LIM The distance between reference switch and negative limit switch (user unit)	X	Represents the physical distance between the reference switch and the negative limit switch integrated in the mechanics. Must be set even if no reference switch is present in the mechanics because the position is set to the difference of VALUE_AT_REF_POS and DISTANCE_REF_TO_N_LIM when the mechanics performs a reference move to the negative limit switch.
0x18 / 24	FLOAT	Limit switch polarity	X	0 = positive limit switch active high (pos-HI), negative limit switch active high (neg-HI) 1 = positive limit switch active low (pos-LO), neg-HI 2 = pos-HI, neg-LO 3 = pos-LO, neg-LO

Parameter ID (hexadecimal / decimal)	Data Type	Parameter Description	Changing with C843_SPA not recommended	Possible Values/Notes
0x19 / 25	FLOAT	Stage type	X	0 = DC motor 2 = Voice coil
0x1A / 26	FLOAT	Stage has brakes	X	0 = Stage has no brakes 1 = Stage has brakes
0x1B / 27	FLOAT	Current profile mode	X	0 = Trapezoidal point-to-point 1 = Velocity contouring (use only with rotary stages) 2 = S-curve point-to-point 4 = User profile mode (read only) Note that S-curve profile mode does not support changes to any of the profile parameters while the axis is in motion. See MotionProcessors User Guide for more information.
0x2F / 47	FLOAT	DISTANCE_REF_TO_P_LIM The distance between reference switch and positive limit switch (user unit)	X	Represents the physical distance between the reference switch and the positive limit switch integrated in the mechanics. Must be set even if no reference switch is present in the mechanics because the position is set to the sum of VALUE_AT_REF_POS and DISTANCE_REF_TO_P_LIM when the mechanics performs a reference move to the positive limit switch.
0x30 / 48	FLOAT	MAX_TRAVEL_RANGE_NEG The maximum travel in negative direction (user unit)	X	"Soft limit", based on the home (zero) position. If the soft limit is larger than the position value for the negative limit switch (which is given by the difference of the parameters 0x16 and 0x17), the negative limit switch can not be used for referencing. Can be negative.
0x31 / 49	FLOAT	Invert reference switch signal	X	1 = invert reference switch signal 0 = do not invert
0x32 / 50	FLOAT	Stage has limit switches; enables / disables the stopping of the motion at the limit switches	X	0 = Stage has limit switches 1 = Stage has no limit switches
0x36 / 54	FLOAT	Settle window (counts)		0 to 2^{31} The settle window is centered around the target position. The on-target status becomes "true" when the current position stays in this window for at least the settle time (parameters 0x3F / 0x38).

Parameter ID (hexadecimal / decimal)	Data Type	Parameter Description	Changing with C843_SPA not recommended	Possible Values/Notes
0x38 / 56	FLOAT	Settle time (cycles)		0 to 32767 Used for on-target detection: The on-target status becomes "true" when the current position stays in the settle window (parameter 0x36) for at least the settle time. If the settle time is set to 0, then the axis is on target when the trajectory has finished, irrespective of the current position. Parameter 0x38 has the same value as parameter 0x3F, but given in number of cycles. If parameter 0x3F is changed, parameter 0x38 is adapted automatically and vice versa.
0x3C / 60	CHAR	Stage name	X	Maximum 31 characters Can be queried with CST?. Note: To connect a stage, always use CST. Do not set the stage name parameter with C843_SPA(). Otherwise the stage parameters will not be loaded properly from the stage database.
0x3F / 63	FLOAT	Settle time (s)		The same value as parameter 0x38, but given in seconds. If parameter 0x38 is changed, parameter 0x3F is adapted automatically and vice versa.
0x49 / 73	FLOAT	Current closed-loop velocity (user unit/s) also changed by C843_VEL()		Gives the current velocity, limited by parameter 0xA
0x4A / 74	FLOAT	Maximum closed-loop acceleration (user unit/s^2)	X	Gives the maximum value for parameter 0xB
0x4B / 75	FLOAT	Maximum closed-loop deceleration (user unit/s^2)	X	Gives the maximum value for parameter 0xC
0x50 / 80	FLOAT	Velocity for reference moves and find-edge moves (user unit/s)		Gives the maximum velocity to be used for reference moves with C843_REF(),C843_FRF(),C843_MPL(), C843_FPL(),C843_MNL(),C843_FNL() and for find-edge moves with C843_FED(). If set to 0, reference moves or find-edge moves are not possible.
0x59 / 89	FLOAT	Acceleration feed forward		0 to 32767 Gives the acceleration feed forward term of the servo filter. See MotionProcessors User Guide for more information.
0x16000000 / 369098752	INT	Data Recorder Table rate (cycles) also changed by C843_RTR()		Gives the data recorder sampling period (default value is one servo cycle). You can cover longer periods by increasing this value.

Parameter ID (hexadecimal / decimal)	Data Type	Parameter Description	Changing with C843_SPA not recommended	Possible Values/Notes
0x16000200 / 369099264	INT	Data Recorder Maximum record points	X	Gives the total number of points available for data recording (max. 32,256). The points available are in equal shares allocated to the tables with non-zero DRC record options. Note that the data recorder shares the 32,256 points of volatile memory provided on the C-843 card (referred to as “external RAM” in the MotionProcessor Users Guide) with the multi-axis motion profiles which can be created by the User Profile Mode commands (C843_Upx() functions).

5.3. Transmission Ratio and Scaling Factor

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

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

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

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

*Number is the parameter ID for C843_SPA() and C843_qSPA(), see also parameter list beginning on p. 62.

**See C843_DFF()

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

CAUTION

To customize the physical unit of a stage do not change parameter 0xE and parameter 0xF but use C843_DFF() instead. Although C843_DFF() has the same effect as changing parameter 0x12 with C843_SPA(), you should only use C843_DFF() and not C843_SPA() to modify the scaling factor.

Example: If you set with C843_DFF() a value of 25.4 for an axis, the physical unit for this axis is converted from mm to inches.

5.4. Travel Range Adjustment

The figures below give a universal hardware scheme of a positioning stage with incremental sensor, reference and limit switches. To work with such a stage, the stage parameters must be adjusted properly (see "Parameter Handling" on p. 61 for how to modify parameter values).

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

DISTANCE_REF_TO_N_LIM (parameter ID 0x17) = 8

DISTANCE_REF_TO_P_LIM (parameter ID 0x2F) = 12

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

VALUE_AT_REF_POS (parameter ID 0x16) = 8

To allow for absolute moves, either an absolute "initial" position can be set with C843_POS(), or the stage can perform a reference move to a known position where a defined position value will be set as the current position (see also C843_RON()). By default, a reference move is required. In the example, known positions for reference moves are given by the reference switch and the limit switches. Depending on the switch used for the reference move, a certain combination of the above-mentioned parameters is used to calculate the position to be set at the end of the move:

- Reference switch (C843_REF() or C843_FRF()): the stage is moved to the reference switch, and the value of VALUE_AT_REF_POS is set as the current position.
- Negative limit switch (C843_MNL() or C843_FNL()): the stage is moved to the negative limit switch and the difference of VALUE_AT_REF_POS and DISTANCE_REF_TO_N_LIM is set as the current position (can be negative).
- Positive limit switch (C843_MPL() or C843_FPL()): the stage is moved to the positive limit switch and the sum of VALUE_AT_REF_POS and DISTANCE_REF_TO_P_LIM is set as the current position.

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

MAX_TRAVEL_RANGE_POS (parameter ID 0x15) = 20 mm

MAX_TRAVEL_RANGE_NEG (parameter ID 0x30) = 0 mm

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

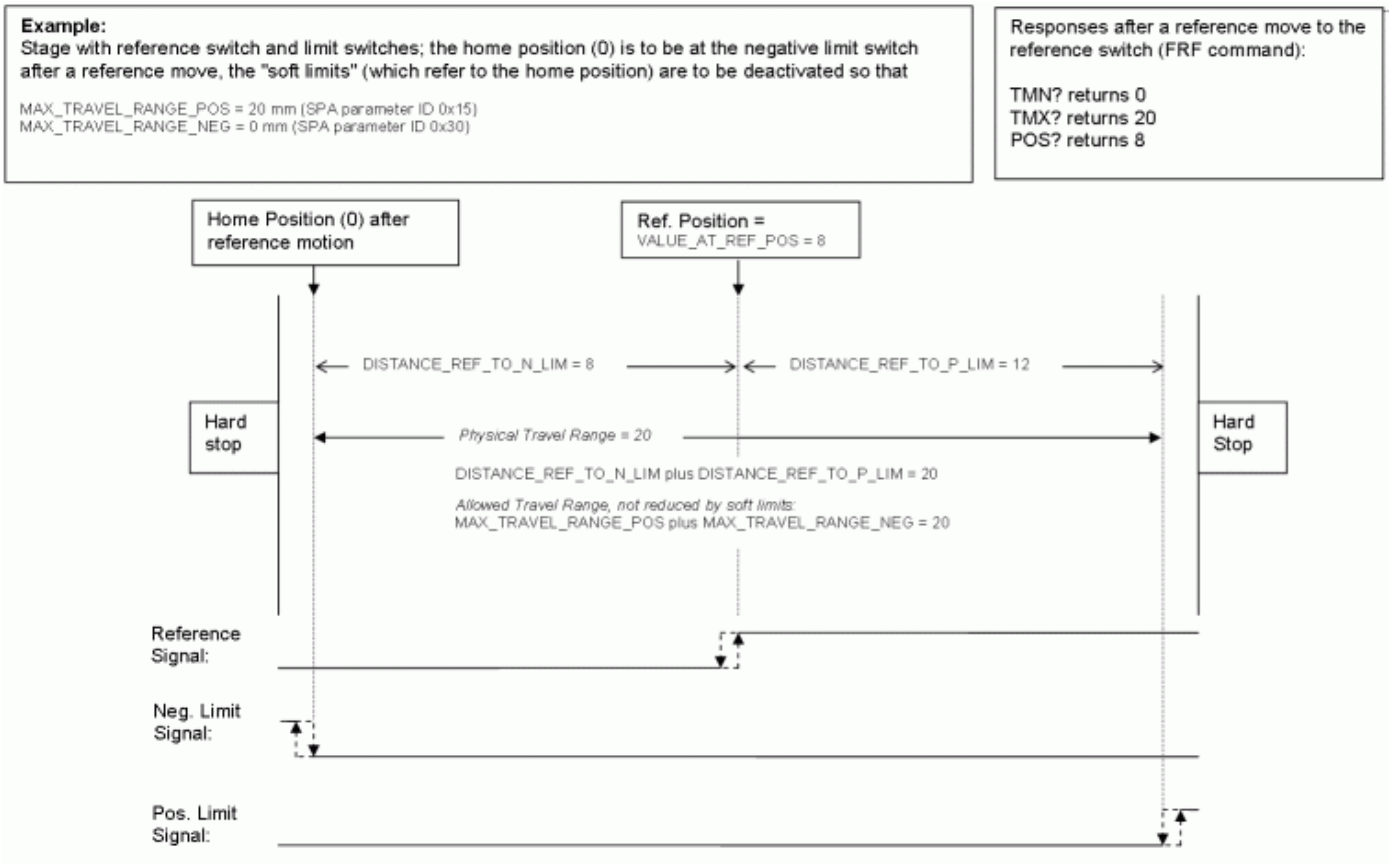


Figure 1: Positioning stage and corresponding controller parameters

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

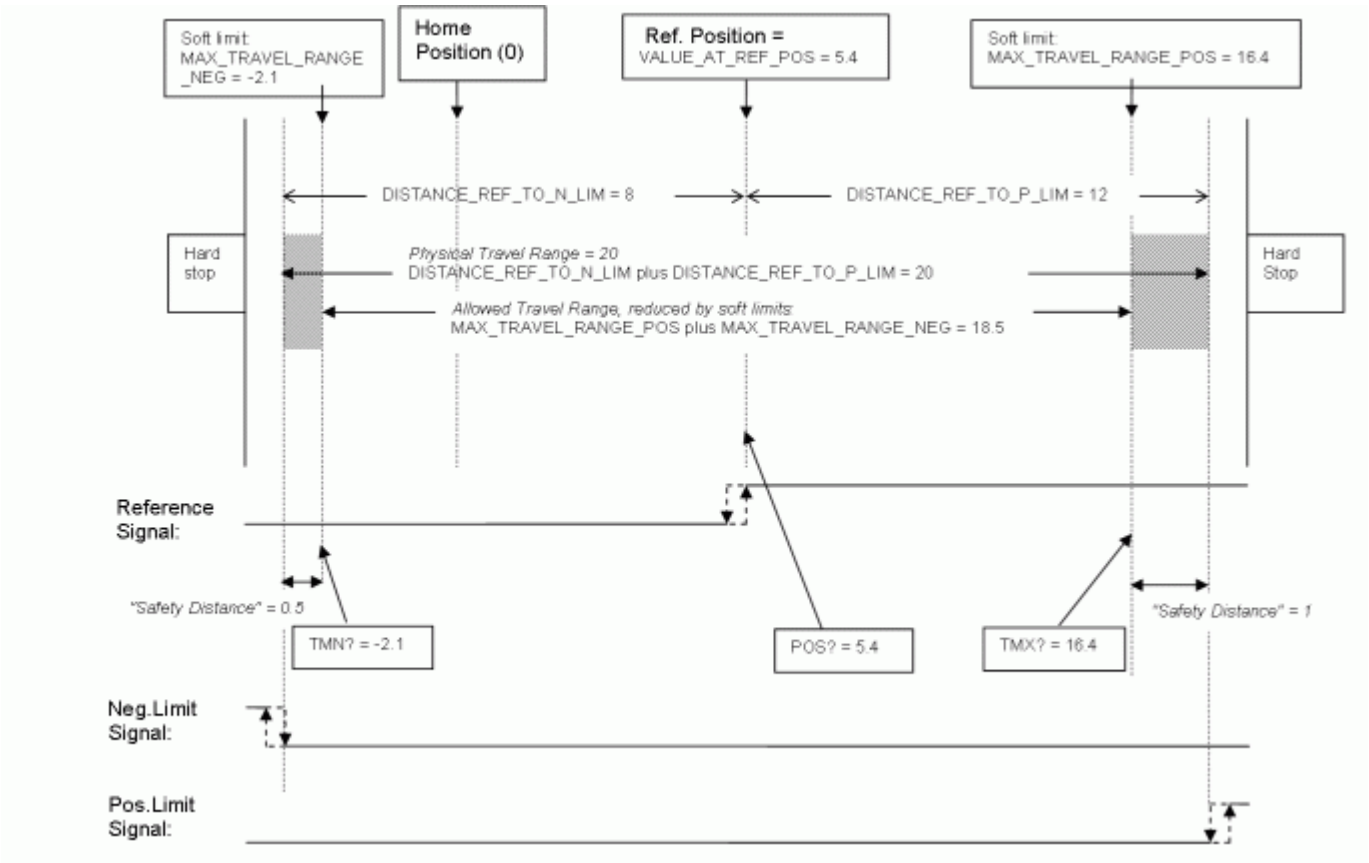


Figure 2: Positioning stage, soft limits set in the controller to reduce the travel range

After the stage was referenced again by moving it to the reference switch (with C843_REF() or C843_FRF()), the following responses will be given:

C843_qTMN() returns -2.1

C843_qTMX() returns 16.4

C843_qPOS() returns 5.4

CAUTION

If the soft limits (MAX_TRAVEL_RANGE_POS and MAX_TRAVEL_RANGE_NEG) are used to reduce the travel range, the limit switches can not be used for reference moves. C843_MNL(),C843_FNL(),C843_MPL() and C843_FPL() will provoke an error message, and only the reference switch can be used for a reference move (C843_REF() or C843_FRF()).

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

The soft limits may not be outside of the physical travel range:
MAX_TRAVEL_RANGE_POS ≤ DISTANCE_REF_TO_P_LIM + VALUE_AT_REF_POS
MAX_TRAVEL_RANGE_NEG ≥ VALUE_AT_REF_POS - DISTANCE_REF_TO_N_LIM
Otherwise, reference moves to the limit switches would have incorrect results because the values of the soft limits would be set at the end of the referencing procedure.

Be careful when referencing the stage by setting an initial absolute position with C843_POS() since the values for MAX_TRAVEL_RANGE_POS and MAX_TRAVEL_RANGE_NEG are not adapted. In the worst case, the soft limits will now be outside of the physical travel range, and the stage will no longer be able to move since the move commands check the soft limit settings.

5.5. Parameter Databases

The C843_GCS_DLL and the GCS-based host software from PI use multiple databases for stage parameters:

- **PIStages2.dat** contains parameter sets for all standard stages from PI and is automatically installed on the host PC with the setup. It cannot be edited; should changes in the file become necessary, you must obtain a new version from PI and install it on your host PC (see “Updating PIStages2.dat”, p. 72).
- **C843UserStages2.dat** allows you to create and save your own stages (see “Functions for User-Defined Stages” on p. 17). This database is created the first time you connect stages in the host software (i.e. the first time the C843_qVST() or C843_CST() functions of the C843_GCS library are used which is the case, for example, when VST? or CST are sent in PITerminal or the *Select connected stages* startup step is performed in PIMikroMove™).
- **M-xxx.dat** files contain parameter sets for custom stages delivered by PI. Those files are provided by CDs which come with the stages and have to be copied to the host PC according to the accompanying instructions. M-xxx.dat files can not be edited; should changes become necessary, you must obtain a new version from PI.

The PIStages2.dat, C843UserStages2.dat and M-xxx.dat databases are located in the ...\PI\GcsTranslator directory on the host PC. The location of the PI directory is that specified upon installation, usually in C:\Documents and Settings\All Users\Application Data (Windows XP) or C:\ProgramData (Windows Vista). If this directory does not exist, the program that needs the stage databases will look in its own directory. In PIMikroMove™, you can use the *Version Info* item in the controller menu or the *Search for controller software* item in the *Connections* menu to identify the GcsTranslator path.

Notes for users which have already installed older versions of C843_GCS_DLL, PIMikroMove™ and PIStageEditor:

- The format of the stage parameter (DAT) files has changed (more parameters provided), realized by a file version change from 1 to 2. Note that PIStages and C843Userstages DAT files with version 2 contain a "2" in their file name, e.g. PIStages2.dat (instead of PIStages.dat for version 1).
- Existing C843Userstages DAT files of version 1 are automatically converted to version 2 files the first time you connect stages in the host software, i.e. the first time the C843_qVST() or C843_CST() functions of the C843_GCS library are used which is the case, for example, when VST? or CST are sent in PITerminal or the *Select connected stages* startup step is performed in PIMikroMove™. The *Edit user stages data…* item in the controller menu of PIMikroMove™ opens the PIStageEditor tool with the version 2 file (C843UserStages2.dat). Parameters which were not present in version 1 are set to default values during conversion.
- Version 4 and newer of the PIStageEditor supports stage parameter files of version 2 (in PIMikroMove™, you can check the version of the PIStageEditor with *Help → About PiStageEditor*). If it is necessary to update the PIStageEditor, run either the setup from the latest revision of the CD for your controller, or download

the latest revision of the PIStageEditor from the PI website. It can be found there in the same directory like the PIStages2.dat stage database. See “Updating PIStages2.dat” below for download instructions and make sure to copy the PiStageEditor.dll to the ...\PI\GcsTranslator directory.

5.6. Updating PIStages2.dat

To install the latest version of PIStages2.dat from the PI Website proceed as follows:

1. On the www.pi.ws front page, move the cursor to *Manuals, Software, ISO Statements* in the *Service* section on the left.
2. Select *Link to Software Server* from the list that pops up.
3. On the *PI Support Site* page, click on the *General Software* category (no login or password is required).
4. Click on *PI Stages.*
5. Click on *pistages2.*
6. In the download window, switch to the ...\PI\GcsTranslator directory. The location of the PI directory is that specified upon installation, usually in C:\Documents and Settings\All Users\Application Data (Windows XP) or C:\ProgramData (Windows Vista) (may differ in other-language Windows versions).

   Note that in PIMikroMove™, you can use the *Version Info* entry in the controller menu or the *Search for controller software* entry in the *Connections* menu to identify the GcsTranslator path.
7. If desired, rename the existing PIStages2.dat (if present) so as to preserve a copy for safety reasons.
8. Download the file from the server as PIStages2.dat.

6. Error Codes

The error codes listed here are those of the PI General Command Set. As such, some are not relevant to the C-843 and will simply never occur with the systems this manual describes.
The error codes are defined in separate header files `"InterfaceErrors.h"` and `"PIControllerErrors.h"` shipped with the C843 GCS_DLL

Controller Errors

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

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

42	PI_CNTR_SLAVE_POSITION_OUT_OF_RANGE	Position of slave axis is out of range
43	PI_CNTR_COMMAND_EGE_SLAVE	Slave axis cannot be commanded directly when electronic gearing is enabled
44	PI_CNTR_JOYSTICK_CALIBRATION_FAILED	Calibration of joystick failed
45	PI_CNTR_REFERENCING_FAILED	Referencing failed
46	PI_CNTR_OPM_MISSING	OPM (Optical Power Meter) missing
47	PI_CNTR_OPM_NOT_INITIALIZED	OPM (Optical Power Meter) not initialized or cannot be initialized
48	PI_CNTR_OPM_COM_ERROR	OPM (Optical Power Meter) Communication Error
49	PI_CNTR_MOVE_TO_LIMIT_SWITCH_FAILED	Move to limit switch failed
50	PI_CNTR_REF_WITH_REF_DISABLED	Attempt to reference axis with referencing disabled
51	PI_CNTR_AXIS_UNDER_JOYSTICK_CONTROL	Selected axis is controlled by joystick
52	PI_CNTR_COMMUNICATION_ERROR	Controller detected communication error
53	PI_CNTR_DYNAMIC_MOVE_IN_PROCESS	MOV! motion still in progress
54	PI_CNTR_UNKNOWN_PARAMETER	Unknown parameter
55	PI_CNTR_NO_REP_RECORDED	No commands were recorded with REP
56	PI_CNTR_INVALID_PASSWORD	Password invalid
57	PI_CNTR_INVALID_RECORDER_CHAN	Data Record Table does not exist
58	PI_CNTR_INVALID_RECORDER_SRC_OPT	Source does not exist; number too low or too high
59	PI_CNTR_INVALID_RECORDER_SRC_CHAN	Source Record Table number too low or too high
60	PI_CNTR_PARAM_PROTECTION	Protected Param: current Command Level (CCL) too low
61	PI_CNTR_AUTOZERO_RUNNING	Command execution not possible while Autozero is running
62	PI_CNTR_NO_LINEAR_AXIS	Autozero requires at least one linear axis

63	PI_CNTR_INIT_RUNNING	Initialization still in progress
64	PI_CNTR_READ_ONLY_PARAMETER	Parameter is read-only
65	PI_CNTR_PAM_NOT_FOUND	Parameter not found in non-volatile memory
66	PI_CNTR_VOL_OUT_OF_LIMITS	Voltage out of limits
67	PI_CNTR_WAVE_TOO_LARGE	Not enough memory available for requested wave curve
68	PI_CNTR_NOT_ENOUGH_DDL_MEMORY	Not enough memory available for DDL table; DDL can not be started
69	PI_CNTR_DDL_TIME_DELAY_TOO_LARGE	Time delay larger than DDL table; DDL can not be started
70	PI_CNTR_DIFFERENT_ARRAY_LENGTH	The requested arrays have different lengths; query them separately
71	PI_CNTR_GEN_SINGLE_MODE_RESTART	Attempt to restart the generator while it is running in single step mode
72	PI_CNTR_ANALOG_TARGET_ACTIVE	Motion commands and wave generator activation are not allowed when analog target is active
73	PI_CNTR_WAVE_GENERATOR_ACTIVE	Motion commands are not allowed when wave generator is active
74	PI_CNTR_AUTOZERO_DISABLED	No sensor channel or no piezo channel connected to selected axis (sensor and piezo matrix)
75	PI_CNTR_NO_WAVE_SELECTED	Generator started (WGO) without having selected a wave table (WSL).
76	PI_CNTR_IF_BUFFER_OVERRUN	Interface buffer did overrun and command couldn't be received correctly
77	PI_CNTR_NOT_ENOUGH_RECORDED_DATA	Data Record Table does not hold enough recorded data
78	PI_CNTR_TABLE_DEACTIVATED	Data Record Table is not configured for recording
79	PI_CNTR_OPENLOOP_VALUE_SET_WHEN_SERVO_ON	Open-loop commands (SVA, SVR) are not allowed when servo is on
80	PI_CNTR_RAM_ERROR	Hardware error affecting RAM

81	PI_CNTR_MACRO_UNKNOWN_COMMAND	Not macro command
82	PI_CNTR_MACRO_PC_ERROR	Macro counter out of range
83	PI_CNTR_JOYSTICK_ACTIVE	Joystick is active
84	PI_CNTR_MOTOR_IS_OFF	Motor is off
85	PI_CNTR_ONLY_IN_MACRO	Macro-only command
86	PI_CNTR_JOYSTICK_UNKNOWN_AXIS	Invalid joystick axis
87	PI_CNTR_JOYSTICK_UNKNOWN_ID	Joystick unknown
88	PI_CNTR_REF_MODE_IS_ON	Move without referenced stage
89	PI_CNTR_NOT_ALLOWED_IN_CURRENT_MOTION_MODE	Command not allowed in current motion mode
90	PI_CNTR_DIO_AND_TRACING_NOT_POSSIBLE	No tracing possible while digital IOs are used on this HW revision. Reconnect to switch operation mode.
91	PI_CNTR_COLLISION	Move not possible, would cause collision
100	PI_LABVIEW_ERROR	PI LabVIEW driver reports error. See source control for details.
200	PI_CNTR_NO_AXIS	No stage connected to axis
201	PI_CNTR_NO_AXIS_PARAM_FILE	File with axis parameters not found
202	PI_CNTR_INVALID_AXIS_PARAM_FILE	Invalid axis parameter file
203	PI_CNTR_NO_AXIS_PARAM_BACKUP	Backup file with axis parameters not found
204	PI_CNTR_RESERVED_204	PI internal error code 204
205	PI_CNTR_SMO_WITH_SERVO_ON	SMO with servo on
206	PI_CNTR_UUDECODE_INCOMPLETE_HEADER	uudecode: incomplete header
207	PI_CNTR_UUDECODE_NOTHING_TO_DECODE	uudecode: nothing to decode
208	PI_CNTR_UUDECODE_ILLEGAL_FORMAT	uudecode: illegal UUE format
209	PI_CNTR_CRC32_ERROR	CRC32 error

210	PI_CNTR_ILLEGAL_FILENAME	Illegal file name (must be 8-0 format)
211	PI_CNTR_FILE_NOT_FOUND	File not found on controller
212	PI_CNTR_FILE_WRITE_ERROR	Error writing file on controller
213	PI_CNTR_DTR_HINDERS_VELOCITY_CHANGE	VEL command not allowed in DTR Command Mode
214	PI_CNTR_POSITION_UNKNOWN	Position calculations failed
215	PI_CNTR_CONN_POSSIBLY_BROKEN	The connection between controller and stage may be broken
216	PI_CNTR_ON_LIMIT_SWITCH	The connected stage has driven into a limit switch, some controllers need CLR to resume operation
217	PI_CNTR_UNEXPECTED_STRUT_STOP	Strut test command failed because of an unexpected strut stop
218	PI_CNTR_POSITION_BASED_ON_ESTIMATION	While MOV! is running position can only be estimated!
219	PI_CNTR_POSITION_BASED_ON_INTERPOLATION	Position was calculated during MOV motion
230	PI_CNTR_INVALID_HANDLE	Invalid handle
231	PI_CNTR_NO_BIOS_FOUND	No bios found
232	PI_CNTR_SAVE_SYS_CFG_FAILED	Save system configuration failed
233	PI_CNTR_LOAD_SYS_CFG_FAILED	Load system configuration failed
301	PI_CNTR_SEND_BUFFER_OVERFLOW	Send buffer overflow
302	PI_CNTR_VOLTAGE_OUT_OF_LIMITS	Voltage out of limits
303	PI_CNTR_OPEN_LOOP_MOTION_SET_WHEN_SERVO_ON	Open-loop motion attempted when servo ON
304	PI_CNTR_RECEIVING_BUFFER_OVERFLOW	Received command is too long
305	PI_CNTR_EEPROM_ERROR	Error while reading/writing EEPROM
306	PI_CNTR_I2C_ERROR	Error on I2C bus
307	PI_CNTR_RECEIVING_TIMEOUT	Timeout while receiving command

308	PI_CNTR_TIMEOUT	A lengthy operation has not finished in the expected time
309	PI_CNTR_MACRO_OUT_OF_SPACE	Insufficient space to store macro
310	PI_CNTR_EUI_OLDVERSION_CFGDATA	Configuration data has old version number
311	PI_CNTR_EUI_INVALID_CFGDATA	Invalid configuration data
333	PI_CNTR_HARDWARE_ERROR	Internal hardware error
400	PI_CNTR_WAV_INDEX_ERROR	Wave generator index error
401	PI_CNTR_WAV_NOT_DEFINED	Wave table not defined
402	PI_CNTR_WAV_TYPE_NOT_SUPPORTED	Wave type not supported
403	PI_CNTR_WAV_LENGTH_EXCEEDS_LIMIT	Wave length exceeds limit
404	PI_CNTR_WAV_PARAMETER_NR	Wave parameter number error
405	PI_CNTR_WAV_PARAMETER_OUT_OF_LIMIT	Wave parameter out of range
406	PI_CNTR_WGO_BIT_NOT_SUPPORTED	WGO command bit not supported
502	PI_CNTR_REDUNDANCY_LIMIT_EXCEEDED	Position consistency check failed
503	PI_CNTR_COLLISION_SWITCH_ACTIVATED	Hardware collision sensor(s) are activated
504	PI_CNTR_FOLLOWING_ERROR	Strut following error occurred, e.g. caused by overload or encoder failure
555	PI_CNTR_UNKNOWN_ERROR	BasMac: unknown controller error
601	PI_CNTR_NOT_ENOUGH_MEMORY	not enough memory
602	PI_CNTR_HW_VOLTAGE_ERROR	hardware voltage error
603	PI_CNTR_HW_TEMPERATURE_ERROR	hardware temperature out of range
1000	PI_CNTR_TOO_MANY_NESTED_MACROS	Too many nested macros
1001	PI_CNTR_MACRO_ALREADY_DEFINED	Macro already defined
1002	PI_CNTR_NO_MACRO_RECORDING	Macro recording not activated

1003	PI_CNTR_INVALID_MAC_PARAM	Invalid parameter for MAC
1004	PI_CNTR_RESERVED_1004	PI internal error code 1004
1005	PI_CNTR_CONTROLLER_BUSY	Controller is busy with some lengthy operation (e.g. reference move, fast scan algorithm)
2000	PI_CNTR_ALREADY_HAS_SERIAL_NUMBER	Controller already has a serial number
4000	PI_CNTR_SECTOR_ERASE_FAILED	Sector erase failed
4001	PI_CNTR_FLASH_PROGRAM_FAILED	Flash program failed
4002	PI_CNTR_FLASH_READ_FAILED	Flash read failed
4003	PI_CNTR_HW_MATCHCODE_ERROR	HW match code missing/invalid
4004	PI_CNTR_FW_MATCHCODE_ERROR	FW match code missing/invalid
4005	PI_CNTR_HW_VERSION_ERROR	HW version missing/invalid
4006	PI_CNTR_FW_VERSION_ERROR	FW version missing/invalid
4007	PI_CNTR_FW_UPDATE_ERROR	FW update failed
5200	PI_CNTR_AXIS_NOT_CONFIGURED	Axis must be configured for this action

Interface Errors

0	COM_NO_ERROR	No error occurred during function call
-1	COM_ERROR	Error during com operation (could not be specified)
-2	SEND_ERROR	Error while sending data
-3	REC_ERROR	Error while receiving data
-4	NOT_CONNECTED_ERROR	Not connected (no port with given ID open)
-5	COM_BUFFER_OVERFLOW	Buffer overflow
-6	CONNECTION_FAILED	Error while opening port

-7	COM_TIMEOUT	Timeout error
-8	COM_MULTILINE_RESPONSE	There are more lines waiting in buffer
-9	COM_INVALID_ID	There is no interface or DLL handle with the given ID
-10	COM_NOTIFY_EVENT_ERROR	Event/message for notification could not be opened
-11	COM_NOT_IMPLEMENTED	Function not supported by this interface type
-12	COM_ECHO_ERROR	Error while sending "echoed" data
-13	COM_GPIB_EDVR	IEEE488: System error
-14	COM_GPIB_ECIC	IEEE488: Function requires GPIB board to be CIC
-15	COM_GPIB_ENOL	IEEE488: Write function detected no listeners
-16	COM_GPIB_EADR	IEEE488: Interface board not addressed correctly
-17	COM_GPIB_EARG	IEEE488: Invalid argument to function call
-18	COM_GPIB_ESAC	IEEE488: Function requires GPIB board to be SAC
-19	COM_GPIB_EABO	IEEE488: I/O operation aborted
-20	COM_GPIB_ENEB	IEEE488: Interface board not found
-21	COM_GPIB_EDMA	IEEE488: Error performing DMA
-22	COM_GPIB_EOIP	IEEE488: I/O operation started before previous operation completed
-23	COM_GPIB_ECAP	IEEE488: No capability for intended operation
-24	COM_GPIB_EFSO	IEEE488: File system operation error
-25	COM_GPIB_EBUS	IEEE488: Command error during device call
-26	COM_GPIB_ESTB	IEEE488: Serial poll-status byte lost
-27	COM_GPIB_ESRQ	IEEE488: SRQ remains asserted

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

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

DLL Errors

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

-1008	PI_CONTROLLER_BUSY	Controller is busy with some lengthy operation (e.g. reference move, fast scan algorithm)
-1009	PI_THREAD_ERROR	Internal error--could not start thread
-1010	PI_IN_MACRO_MODE	Controller is (already) in macro mode--command not valid in macro mode
-1011	PI_NOT_IN_MACRO_MODE	Controller not in macro mode--command not valid unless macro mode active
-1012	PI_MACRO_FILE_ERROR	Could not open file to write or read macro
-1013	PI_NO_MACRO_OR_EMPTY	No macro with given name on controller, or macro is empty
-1014	PI_MACRO_EDITOR_ERROR	Internal error in macro editor
-1015	PI_INVALID_ARGUMENT	One or more arguments given to function is invalid (empty string, index out of range, ...)
-1016	PI_AXIS_ALREADY_EXISTS	Axis identifier is already in use by a connected stage
-1017	PI_INVALID_AXIS_IDENTIFIER	Invalid axis identifier
-1018	PI_COM_ARRAY_ERROR	Could not access array data in COM server
-1019	PI_COM_ARRAY_RANGE_ERROR	Range of array does not fit the number of parameters
-1020	PI_INVALID_SPA_CMD_ID	Invalid parameter ID given to SPA or SPA?
-1021	PI_NR_AVG_OUT_OF_RANGE	Number for AVG out of range--must be >0
-1022	PI_WAV_SAMPLES_OUT_OF_RANGE	Incorrect number of samples given to WAV
-1023	PI_WAV_FAILED	Generation of wave failed
-1024	PI_MOTION_ERROR	Motion error while axis in motion, call CLR to resume operation
-1025	PI_RUNNING_MACRO	Controller is (already) running a macro
-1026	PI_PZT_CONFIG_FAILED	Configuration of PZT stage or amplifier failed
-1027	PI_PZT_CONFIG_INVALID_PARAMS	Current settings are not valid for desired configuration

-1028	PI_UNKNOWN_CHANNEL_IDENTIFIER	Unknown channel identifier
-1029	PI_WAVE_PARAM_FILE_ERROR	Error while reading/writing wave generator parameter file
-1030	PI_UNKNOWN_WAVE_SET	Could not find description of wave form. Maybe WG.INI is missing?
-1031	PI_WAVE_EDITOR_FUNC_NOT_LOADED	The WGWaveEditor DLL function was not found at startup
-1032	PI_USER_CANCELLED	The user cancelled a dialog
-1033	PI_C844_ERROR	Error from C-844 Controller
-1034	PI_DLL_NOT_LOADED	DLL necessary to call function not loaded, or function not found in DLL
-1035	PI_PARAMETER_FILE_PROTECTED	The open parameter file is protected and cannot be edited
-1036	PI_NO_PARAMETER_FILE_OPENED	There is no parameter file open
-1037	PI_STAGE_DOES_NOT_EXIST	Selected stage does not exist
-1038	PI_PARAMETER_FILE_ALREADY_OPENED	There is already a parameter file open. Close it before opening a new file
-1039	PI_PARAMETER_FILE_OPEN_ERROR	Could not open parameter file
-1040	PI_INVALID_CONTROLLER_VERSION	The version of the connected controller is invalid
-1041	PI_PARAM_SET_ERROR	Parameter could not be set with SPA--parameter not defined for this controller!
-1042	PI_NUMBER_OF_POSSIBLE_WAVES_EXCEEDED	The maximum number of wave definitions has been exceeded
-1043	PI_NUMBER_OF_POSSIBLE_GENERATORS_EXCEEDED	The maximum number of wave generators has been exceeded
-1044	PI_NO_WAVE_FOR_AXIS_DEFINED	No wave defined for specified axis
-1045	PI_CANT_STOP_OR_START_WAV	Wave output to axis already stopped/started
-1046	PI_REFERENCE_ERROR	Not all axes could be referenced
-1047	PI_REQUIRED_WAVE_NOT_FOUND	Could not find parameter set required by frequency relation

-1048	PI_INVALID_SPP_CMD_ID	Command ID given to SPP or SPP? is not valid
-1049	PI_STAGE_NAME_ISNT_UNIQUE	A stage name given to CST is not unique
-1050	PI_FILE_TRANSFER_BEGIN_MISSING	A uuencoded file transferred did not start with "begin" followed by the proper filename
-1051	PI_FILE_TRANSFER_ERROR_TEMP_FILE	Could not create/read file on host PC
-1052	PI_FILE_TRANSFER_CRC_ERROR	Checksum error when transferring a file to/from the controller
-1053	PI_COULDNT_FIND_PISTAGES_DAT	The PiStages2.dat database could not be found. This file is required to connect a stage with the CST command
-1054	PI_NO_WAVE_RUNNING	No wave being output to specified axis
-1055	PI_INVALID_PASSWORD	Invalid password
-1056	PI_OPM_COM_ERROR	Error during communication with OPM (Optical Power Meter), maybe no OPM connected
-1057	PI_WAVE_EDITOR_WRONG_PARAMNUM	WaveEditor: Error during wave creation, incorrect number of parameters
-1058	PI_WAVE_EDITOR_FREQUENCY_OUT_OF_RANGE	WaveEditor: Frequency out of range
-1059	PI_WAVE_EDITOR_WRONG_IP_VALUE	WaveEditor: Error during wave creation, incorrect index for integer parameter
-1060	PI_WAVE_EDITOR_WRONG_DP_VALUE	WaveEditor: Error during wave creation, incorrect index for floating point parameter
-1061	PI_WAVE_EDITOR_WRONG_ITEM_VALUE	WaveEditor: Error during wave creation, could not calculate value
-1062	PI_WAVE_EDITOR_MISSING_GRAPH_COMPONENT	WaveEditor: Graph display component not installed
-1063	PI_EXT_PROFILE_UNALLOWED_CMD	User Profile Mode: Command is not allowed, check for required preparatory commands
-1064	PI_EXT_PROFILE_EXPECTING_MOTION_ERROR	User Profile Mode: First target position in User Profile is too far from current position

-1065	PI_EXT_PROFILE_ACTIVE	Controller is (already) in User Profile Mode
-1066	PI_EXT_PROFILE_INDEX_OUT_OF_RANGE	User Profile Mode: Block or Data Set index out of allowed range
-1067	PI_PROFILE_GENERATOR_NO_PROFILE	ProfileGenerator: No profile has been created yet
-1068	PI_PROFILE_GENERATOR_OUT_OF_LIMITS	ProfileGenerator: Generated profile exceeds limits of one or both axes
-1069	PI_PROFILE_GENERATOR_UNKNOWN_PARAMETER	ProfileGenerator: Unknown parameter ID in Set/Get Parameter command
-1070	PI_PROFILE_GENERATOR_PAR_OUT_OF_RANGE	ProfileGenerator: Parameter out of allowed range
-1071	PI_EXT_PROFILE_OUT_OF_MEMORY	User Profile Mode: Out of memory
-1072	PI_EXT_PROFILE_WRONG_CLUSTER	User Profile Mode: Cluster is not assigned to this axis
-1073	PI_UNKNOWN_CLUSTER_IDENTIFIER	Unknown cluster identifier
-1074	PI_INVALID_DEVICE_DRIVER_VERSION	The installed device driver doesn't match the required version. Please see the documentation to determine the required device driver version.
-1075	PI_INVALID_LIBRARY_VERSION	The library used doesn't match the required version. Please see the documentation to determine the required library version.
-1076	PI_INTERFACE_LOCKED	The interface is currently locked by another function. Please try again later.
-1077	PI_PARAM_DAT_FILE_INVALID_VERSION	Version of parameter DAT file does not match the required version. Current files are available at www.pi.ws.
-1078	PI_CANNOT_WRITE_TO_PARAM_DAT_FILE	Cannot write to parameter DAT file to store user defined stage type.
-1079	PI_CANNOT_CREATE_PARAM_DAT_FILE	Cannot create parameter DAT file to store user defined stage type.
-1080	PI_PARAM_DAT_FILE_INVALID_REVISION	Parameter DAT file does not have correct revision.
-1081	PI_USERSTAGES_DAT_FILE_INVALID_REVISION	User stages DAT file does not have correct revision.

7. Index