T e c h n i c a l   D o c u m e n t a t i o n

MS 102E Software Manual

C-848

Windows DLL

Release 1.3.0

Software Interface Description

This document describes software for use with the following products:

C-848.20	**Multi-Axis DC-Motor Controller, 2 Axes**
C-848.40	**Multi-Axis DC-Motor Controller, 4 Axes**

Release: 1.3.0
Release Date: 2003-08-18

Table of Contents

0. Disclaimer

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

Release: 1.3.0
File:C-848_DLL_MS102E.doc, 377344 Bytes

1. Introduction to C848 DLL

This library allows controlling one or more PI C-848 controllers connected to a host PC. The library insulates the user from the interface-specific interaction. Connection to each C-848 can be made with RS-232 or with a National Instruments' GPIB (IEEE 488) board.

1.1. Threads

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

1.2. Overview

- Units and GCS(p. 3) explains the units used for commanding positions
- Referencing(p. 3) explains how to proper initialize your system and the connected stages
- DLL Handling(p. **3**) explains how to load the library and how to access the functions provided by the C-848 DLL.
- Function Calls (p. **5**) and Types Used in PI Software (p.2) provides some general information about the syntax of most commands in the DLL.
- Communication Initialization (p.7) shows how to initiate communication with a C-848 controller (see also Interface Settings (p.9)).
- C-848 Commands (p. **10**) describes the functions encapsulating the embedded commands of the C-848.
- Error Codes(p. 44) has a description of the possible errors.

2. Units and GCS

The GCS system uses physical units of measure. Most controllers and GCS software have default conversion factors chosen to convert hardware-dependent units (e.g. encoder counts) into mm or degrees, as appropriate. These defaults are generally taken from a database of stages that can be connected. An additional scale factor can be applied (see DFF command description), making a second physical unit (called the working unit) available without overwriting the conversion factor for the first.

3. Referencing

Upon startup (or after a call to **C848_INI** ()) the controller has no way of knowing the absolute positions of the connected axes. The axes are said to be "unreferenced" and no moves can be made. Moves can be made allowable in the following ways:

- The axis can be referenced. This involves moving it until it trips a reference or limit switch. See the C848_REF, C848_MNL and C848_MPL functions for details
- The controller can be told to set the reference mode for the axis OFF and allow relative moves without knowledge of the absolute position. See the C848_RON function for details. Note that "Reference mode off" can be made power-up default and is the factory default for some stage types.
- For axes with reference mode OFF, the controller can be told to assume the absolute position has a given value. See the C848_POS function for details.

4. 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 `C848_DLL.DLL` in Delphi projects. Please see http://www.drbob42.com/delphi/headconv.htm for a detailed description of the steps necessary.)

4.1. Using a Static Import Library

The `C848_DLL.DLL` module is accompanied by the `C848_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:

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

4.2. 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:

1. Use a header or source file where the DLL functions have to be declared, which is needed for the compiler. In the declaration should be taken 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 that they are coming from a C environment. The VC++ compiler needs an `extern "C"` modifier. The declaration also must be aware that these functions have to be called like standard Win-API functions. Therefore the VC++ compiler need a `WINAPI` or `__stdcall` modifier in the declaration.
2. 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
   C848_DLL.C848_IsConnected
```

4.3. 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 can always be done, 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:

1. 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_C848_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_C848_IsConnected)( int );
  FP_C848_IsConnected pC848_IsConnected;
```

2. 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 us by the `GetProcAddress()` function. Example:

```
   HINSTANCE hPI_Dll = LoadLibrary("C848_DLL.DLL\0");
```

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

```
   pC848_IsConnected  = (FP_C848_IsConnected)GetProcAddress(hPI_Dll,"C848_IsConnected\0");
   if (pC848_IsConnected == NULL)
   {
       // do something, for example
       return FALSE;
   }
   BOOL bResult = (*pC848_IsConnected)(1); // call C848_IsConnected(1)
```

5. Function Calls

Almost all functions will return a boolean value of type `BOOL` (see “Types Used in PI Software” (p.6)). If the function succeeded, the return value is **TRUE**, otherwise it is **FALSE**. To find out what went wrong, call **C848_GetError**()(p.8)) and look up the value returned in “ ” (p. 44). The first argument to most function calls is the ID of the selected controller.

5.1. Controller ID

The first argument to most function calls is the ID of the selected controller. To allow the handling of multiple controllers, the user will be returned a non-negative "ID" when he or she opens a connection to a controller (see “Communication Initialization” p.7) This is a kind of index to an internal array storing the information for the different controllers. All other calls addressing the same controller have this ID as first parameter

5.2. Axes Identifiers

Many functions accept one ore more axis identifiers. If no axes are specified (either by giving an empty string or a **NULL** pointer) some functions will address all connected axes.

5.3. Axis Parameters

The parameters for the axes are stored in an array passed to the function. The parameter for the first axis is stored in `array[0]`, for the second axis in `array[1]`, and so on. So, if you call `C848_qPOS("ABC", double pos[3])`, the position for '`A`' is in `pos[0]`, for '`B`' in `pos[1]` and for '`C`' in `pos[2]`.

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

If you call `C848_MOV("AC", double pos[2])` the target position for '`A`' is in `pos[0]` and for '`C`' in `pos[1]`.

Each axis identifier is sent only once. Only the **last** occurrence of an axis identifier is actually sent to the controller with its argument. Thus, if you call `C848_MOV("AAB", pos[3])` with `pos[3] = { 1.0, 2.0, 3.0 }`, '`A`' will move to 2.0 and '`B`' to 3.0. If you then call `C848_qPOS("AAB", pos[3])`, `pos[0]` and `pos[1]` will contain 2.0 as the position of '`A`'.

(See **C848_MOV**() (p.*23*) and **C848_qPOS**() (p.*28*))

See “Types Used in PI Software” (p.6) for a description of types used for parameters.

6. Types Used in PI Software

6.1. Boolean Values

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

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

6.2. NULL Pointers

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

```
#define NULL 0
```

6.3. C-Strings

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

```
char* text = "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`".

7. Communication Initialization

7.1. Functions

- int **C848_ConnectRS232** (const int nPortNr, const long BaudRate)
- int **C848_ConnectNIgpib** (const int nChannelNr, const long nDevAddr)
- int **C848_FindOnRS** (int *pnStartPort, int *pnStartBaud)
- int **C848_InterfaceSetupDlg** (char *const szRegKeyName, BOOL bShowDetails)
- BOOL **C848_IsConnected** (const int ID)
- void **C848_CloseConnection** (const int ID)
- int **C848_GetError** (const int ID)
- BOOL **C848_TranslateError** (int errNr, char *szBuffer, const int maxlen)
- BOOL **C848_SetErrorCheck** (const int ID, BOOL bErrorCheck)

7.2. Detailed Description

To use the DLL and communicate with a C-848 controller, the user must initialize the DLL with one of the "open" functions **C848_InterfaceSetupDlg**() , **C848_ConnectNIgpib**()or **C848_ConnectRS232**(). To allow the handling of multiple controllers, the user will be returned a non-negative "ID" when he calls one of these functions. This is a kind of index to an internal array storing the information for the different controllers. All other calls addressing the same controller have this ID as first parameter. **C848_CloseConnection**()will close the connection to the specified controller and free its system resources.

7.3. Function Documentation

void **C848_CloseConnection** (const int *ID*)

Close connection to C-848 controller associated with *ID*. *ID* will not be valid any longer.

Parameters:

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

int **C848_ConnectNIgpib** (const int *nBoard*, const long *nDevAddr*)

Open a National Instruments board IEEE488 to a C-848 All future calls to control this C-848 need the ID returned by this call.

Parameters:

nBoard number of board (check with NI installation software)
nDevAddr address of connected device

Returns:

ID of new object, **-1** if interface could not be opened or no C-848 is responding.

int **C848_ConnectRS232** (const int *nPortNr*, const long *BaudRate*)

Open an RS-232 ("COM") interface to a C-848. All future calls to control this C-848 need the ID returned by this call.

Parameters:

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

Returns:

ID of new object, **-1** if interface could not be opened or no C-848 is responding.

int **C848_FindOnRS** (int* *pnStartPort*, int* *pnStartBaud*)

Tries many different RS-232 settings to open a connection to a C-848. This function will search ports from *pnStartPort* to 24 and try baudrates that are greater than *pnStartBaud*. If a C-848 responds, the settings used will be written to *pnStartPort* and *pnStartBaud,* and the interface opened, just as by **ConnectRS232**. To search on all available ports with all baudrates, set both parameters to 0.

Note:

This function will take some time. If your C-848 is on COM4 with 115200 baud, finding it will take several seconds if you start with *pnStartBaud=0* and *pnStartBaud=0*

Parameters:

pnStartPort pointer to `int` containing the first port to search on
pnStartBaud pointer to `int` containing the baud rate to start with

return

ID of new object, *-1* if no interface could be opened or no C-848 responds.

int **C848_GetError** (const int *ID*)

Get error status of C-848. This call will also clear the internal error. If there is no internal error, the function will call **C848_qERR**() (p.*26*).

Returns:

error ID, see **Error codes** (p.*44*) for the meaning of the codes.

int **C848_InterfaceSetupDlg** (char *const *szRegKeyName*)

Open dialog to let user select the interface and create a new C848 object. All future calls to control this C-848 need the ID returned by this call. See **Interface Settings** (p.9) for a detailed description of the dialogs shown.

Parameters:

szRegKeyName key in the Windows registry in which to store the settings, the key used is `"HKEY_LOCAL_MACHINE\SOFTWARE\<your keyname>"` if *keyname* is **NULL** or "" the default key `"HKEY_LOCAL_MACHINE\SOFTWARE\PI\C848_DLL"` is used.

Note:

If your programming language is C or C++, use '\\' if you want to create a key and a subkey at once. To create `"MyCompany\C848_DLL"` you must call

```
C848_InterfaceSetupDlg( "MyCompany\\C848_DLL" )
```

Returns:

ID of new object, **-1** if user pressed "CANCEL", the interface could not be opened, or no C-848 is responding.

BOOL **C848_IsConnected** (const int *ID*)

Check if there is a C-848 controller with an ID of *ID*.

Returns:

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

BOOL **C848_SetErrorCheck** (const int *ID*, BOOL *bErrorCheck*)

Set error-check mode of the library. With this call you can specify whether the library should check the error state of the C-848 (with "ERR?") after sending a command. This will slow down communications, so if you need a high data rate, switch off error checking and call **C848_GetError**() (p.8) yourself when there is time to do so. You might want to use permanent error checking to debug your application and switch it off for normal operation. At startup of the library error checking is switched on.

Parameters:

ID ID of controller

bErrorCheck switch error checking on (**TRUE**) or off (**FALSE**)

Returns:

the old state, before this call

BOOL **C848_TranslateError** (int *errNr*, char * *szBuffer*, const int *maxlen*)

Translate error number to error message.

Parameters:

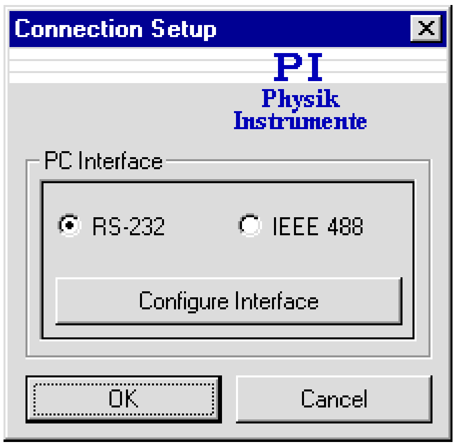
errNr number of error, as returned from **C848_GetError**()(p.8).
szBuffer pointer to buffer that will store the message
maxlen size of the buffer

Returns:

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

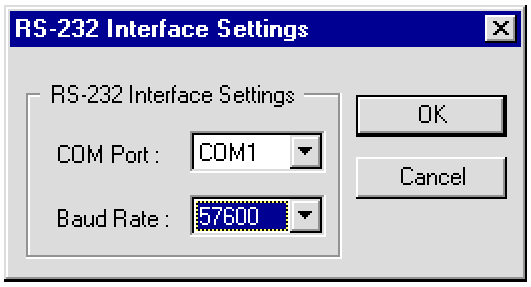
8. Interface Settings

When the interface setup dialog is shown, the user has the choice between RS-232 and IEEE 488 (currently only National Instruments IEEE boards are supported).



8.1. RS-232 Settings

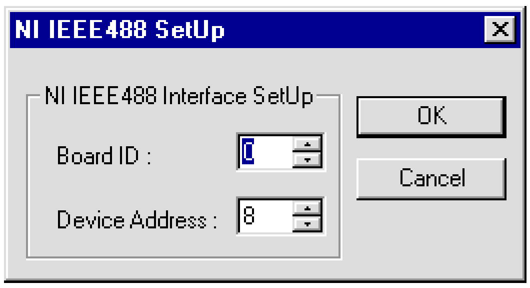
- `COM Port`: Select the desired COM port of the PC, something like "COM1" or "COM2”. the user will see only the ports available on the system.
- `Baud Rate`: The baud rate of the interface. Please read the documentation of the connected device to determine the values it supports and how to set them. The settings must match.



COM Port : COM1
Baud Rate : 57600

8.2. IEEE 488 Settings

- `Board ID`: ID of the National Instruments board installed. If only one board is installed this will be 0, as in the most cases. Use the National Instruments setup and test software to determine the board ID.



- `Device Address`: The address of the connected device. Please read the documentation of the connected device to determine its address setting and, if necessary, how to change it. The settings here and at the device must match.

9. C-848 Commands

9.1. Functions

- BOOL **C848_BRA** (const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_CLR** (const int ID, char *const szAxes)
- BOOL **C848_CLS** (const int ID)
- BOOL **C848_CST** (const int ID, const char axis, char* const name)
- BOOL **C848_DEL** (const int ID, double dSeconds)
- BOOL **C848_DEM** (const int ID)
- BOOL **C848_DFF** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_DFH** (const int ID, char *const szAxes)
- BOOL **C848_DIO** (const int ID, char *const szChannels, BOOL *pbValarray)
- BOOL **C848_DSP** (const int ID, char *const szAxes)
- BOOL **C848_EGE** (const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_GcsCommandset** (const int ID, char* const szCommand)
- BOOL **C848_GcsGetAnswer** (const int ID, char* szAnswer, const int bufsize)
- BOOL **C848_GcsGetAnswerSize** (const int ID, int* iAnswerSize)
- BOOL **C848_GetInputChannelNames** (const int ID, char *szBuffer, const int maxlen)
- BOOL **C848_GetOutputChannelNames** (const int ID, char *szBuffer, const int maxlen)
- BOOL **C848_GetRefResult** (const int ID, char *const szAxes, int *pnResult)
- BOOL **C848_GetSmallestStep** (const int ID, char *const szAxes, double * pdValarray)
- BOOL **C848_GetWaaResult** (const int ID, BOOL* pbWaaResult)
- BOOL **C848_GOH** (const int ID, char *const szAxes)
- BOOL **C848_HasPosChanged** (const const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_HID** (const int ID, char *const szAxes)
- BOOL **C848_HLT** (const int ID, char *const szAxes)
- BOOL **C848_INI** (const int ID, char *const szAxes)
- BOOL **C848_IsIdle** (const int *ID*, BOOL * *pbIdle*)
- BOOL **C848_IsMoving** (const const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_IsRecordingMacro** (const int ID, BOOL *pbRecordingMacro)
- BOOL **C848_IsReferenceOK** (const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_IsReferencing** (const int ID, char *const szAxes, BOOL *pbIsReferencing)
- BOOL **C848_IsRunningMacro** (const int ID, BOOL *pbRunningMacro)
- BOOL **C848_IsWaitingForAllAxes** (const int ID, BOOL* pbIsWaitingForAllAxes)
- BOOL **C848_ITD** (const int ID, char *const szAxes)
- BOOL **C848_JEN** (const int ID, BOOL bOnOff)
- BOOL **C848_JEN_CALIB**(const int ID)
- BOOL **C848_LimitsDialog** (const int ID, const char cAxis)
- BOOL **C848_LoadMacroFromFile** (const int ID, char *szFileName, char *szMacroName)
- BOOL **C848_MAC_BEG** (const int ID, char *szName)

- BOOL **C848_MAC_DEL** (const int ID, char *szName)
- BOOL **C848_MAC_END** (const int ID)
- BOOL **C848_MAC_START** (const int ID, char *szName)
- BOOL **C848_MacroEditor** (const int ID)
- BOOL **C848_qMAS**(const int ID, char* const szAxes, char* szMasters)
- BOOL **C848_MNL** (const int ID, char *const szAxes)
- BOOL **C848_MOV** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_MPL** (const int ID, char *const szAxes)
- BOOL **C848_MSG** (const int ID, char *szMessage)
- BOOL **C848_MVR** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_NLM** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_PLM** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_POS** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_qBRA** (const int ID, char *szBuffer, const int maxlen)
- BOOL **C848_qCST** (const int ID, char *const szAxes, char *names, const int maxlen)
- BOOL **C848_qDFF** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_qDFH** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_qDIO** (const int ID, char *const szChannels, BOOL *pbValarray)
- BOOL **C848_qDSP** (const int ID, char *szBuffer, const int maxlen)
- BOOL **C848_qEGE** (const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_EmergencyStop** (const int *ID*)
- BOOL **C848_qERR** (const int ID, int *pError)
- BOOL **C848_qHID** (const int ID, char *szBuffer, const int maxlen)
- BOOL **C848_qHLP** (const int ID, char *buffer, const int maxlen)
- BOOL **C848_qIDN** (const int ID, char *buffer, const int maxlen)
- BOOL **C848_qJEN** (const int ID, BOOL* pbOnOff)
- BOOL **C848_qLIM** (const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_qMAC** (const int ID, char *szName, char *szBuffer, const int maxlen)
- BOOL **C848_qMOV** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_qNLM** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_qONT** (const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_qPLM** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_qPOS** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_qREF** (const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_qRON** (const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_qSAI** (const int ID, char *axes, const int maxlen)
- BOOL **C848_qSCA** (const int ID, char *pcAxisLeftRight, char *pcAxisUpDown)
- BOOL **C848_qSJA** (const int ID, char* szAxes, int maxlen)
- BOOL **C848_qSMO** (const int ID, char *const szAxes, int *pnValarray)
- BOOL **C848_qSPA** (const int ID, char *const szAxes, int *iCmdarray, double *dValarray)
- BOOL **C848_qSRA** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_qSSL** (const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_qSSN** (const int ID, int *pNr)
- BOOL **C848_qSST** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_qSTA** (const int ID, char *const szAxes, int * pnValarray)
- BOOL **C848_qSTE** (const int ID, const char cAxis, const int iOffset, const int nrValues, double *pdValarray)
- BOOL **C848_qSVO** (const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_qTIM** (const int ID, int *pnTime)
- BOOL **C848_qTIO** (const int ID, int* pNr)
- BOOL **C848_qTMN** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_qTMX** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_qTNJ** (const int ID, int* pNr)
- BOOL **C848_qTVI** (const int ID, char *axes, const int maxlen)
- BOOL **C848_qVEL** (const int ID, char *const szAxes, double *valarray)
- BOOL **C848_qVER** (const int ID, char *buffer, const int maxlen)
- BOOL **C848_qVST** (const const int ID, char *buffer, const int maxlen)
- BOOL **C848_REF** (const int ID, char *const szAxes)
- BOOL **C848_RON** (const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_RST** (const int ID, char *const szAxes)
- BOOL **C848_SAI** (const int ID, char *const szOldAxes, char *const szNewAxes)

- BOOL **C848_SAV** (const int ID, char *const szAxes)
- BOOL **C848_SaveMacroToFile** (const int ID, char *szFileName, char *szMacroName)
- BOOL **C848_SCA** (const int ID, char cAxisLeftRight, char cAxisUpDown)
- BOOL **C848_SJA** (const int ID, char* szAxes, int* pJoystikAxisNr)
- BOOL **C848_SMO** (const int ID, char *const szAxes, int *pnValarray)
- BOOL **C848_SPA** (const int ID, char *const szAxes, int *iCmdarray, double *dValarray)
- BOOL **C848_SRA** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_SSL** (const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_SST** (const int ID, char *const szAxes, double *pdValarray)
- BOOL **C848_StageConfigDlg** (const int ID, char cAxis)
- BOOL **C848_STE** (const int ID, const char cAxis, double dOffset)
- BOOL **C848_STP** (const int ID)
- BOOL **C848_SVO** (const int ID, char *const szAxes, BOOL *pbValarray)
- BOOL **C848_SystemAbort** (const int ID)
- BOOL **C848_SystemInfoDlg** (const int ID)
- BOOL **C848_VEL** (const int ID, char *const szAxes, double *valarray)
- BOOL **C848_VMO** (const int ID, char *const szAxes, double *pdValarray, BOOL *pbMovePossible)
- BOOL **C848_WAI** (const int ID, char *const szAxes)
- BOOL **C848_WAA** (const int ID, unsigned int iWaitTime)

9.2. Detailed Description

These functions encapsulate the embedded commands of the C-848 and provide some "shortcuts" to make the work with C-848 easier. See **"**Function Calls**"** (p. **5**) for some general notes about the parameter syntax. **"**Types Used in PI Software**"** (p. 6) will give you some general information about the syntax of most commands.

9.3. Function Documentation

BOOL **C848_BRA** (const int *ID*, char *const *szAxes*, BOOL * *pbValarray*)

Corresponding command: `BRA`

Set brakes for *szAxes* on (**TRUE)** or off (**FALSE)**.

Parameters:

ID 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 **C848_CLR** (const int *ID*, char *const *szAxes*)

Corresponding command: `CLR`

Clear status of *szAxes*.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_CLS** (const int *ID*)

Corresponding command: `CLS`

Clear the controller display.

Parameters:

ID ID of controller

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_CST** (const int *ID,* const char *axis,* char* const *name*)

Corresponding command: `CST`

Assigns the stage specified by *name* to *axis*. With this command the stage assignment for one axis can be changed. Valid stage names can be listed with **C848_VST()**.

Parameters:

ID ID of controller
axis axis to change
name type name of the new stage

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_DEL** (const int *ID*, double *dmSeconds*)

Corresponding command: `DEL`

Delay the controller for *dmSeconds* milliseconds.

Note:

This will only affect the controller, the function will return immediately!

Parameters:

ID ID of controller
dmSeconds time in milli seconds

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_DEM** (const int *ID*)

Corresponding command: `DEM`

Run demo motion. The controller moves all connected axes randomly. This can be stopped with "F5" on the controller keyboard or with **C848_STP**() (p.*41*).

Parameters:

ID ID of controller

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_DFF** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `DFF`

Defines the physical unit scale factor for *szAxes*, e.g. a factor of 25.4 sets the working unit of all axes in *szAxes* to inches. Changing the scale factor will change the numeric results of other other commands, but not the underlying physical magnitudes.

Example:

The physical unit is mm and the scale factor is 1. The current position of a stage is 12. Now the scale factor is set to 3 with DFF. Reading the position gives 4 as result. A relative move of 1.5 causes the stage to move 4.5 mm.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_DFH** (const int *ID*, char *const *szAxes*)

Corresponding command: `DFH`

Makes current positions of *szAxes* the new home position

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_DIO** (const int *ID*, char *const *szChannels*, BOOL * *pbValarray*)

Corresponding command: `DIO`

Set digital output channels "high" or "low". If *pbValarray[index]* is **TRUE** the mode is set to HIGH, otherwise it is set to LOW (do not confuse relay channels and digital IO channels).

Parameters:

ID ID of controller
szChannels string with digital output channel identifiers
pbValarray array containing the states of specified digital output channels, **TRUE** for "HIGH", **FALSE** for "LOW"

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_DSP** (const int *ID*, char *const *szAxes*)

Corresponding command: `DSP`

Display *szAxes* on controller display.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_EGE** (const int *ID*, char *const *szAxes*, BOOL * *pbValarray*)

Corresponding command: `EGE`

Activates electronic gearing for the specified axes. If *pbValarray [index]* is **FALSE** the mode is "off", otherwise it is set to "on" .

Note:

Only axis A and B are allowed as “slaves”. Master for A is C and for B is D.

Parameters:

ID ID of controller
szAxes string with axes.
pbValarray array of `long` with modes for the specified axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_EmergencyStop** (const int *ID*)

Corresponding command: `#24` (ASCII 24)

Fast stop, must be used to interrupt REF, MNL or MPL commands. The result of the interrupted command will be set to 0.

Parameters:

ID ID of controller

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_GcsCommandset** (const int *ID*, char* const *szCommand*)

Sends a GCS command to the C-848.

Parameters:

ID ID of controller
szCommand the GCS command as string.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_GcsGetAnswer** (const int *ID*, char* *szAnswer*, const int *bufsize*)

Gets the answer of an GCS command (**C848_GcsCommandset**() (p.15)).

Parameters:

ID ID of controller
szAnswer the buffer to take the answer.
Bufsize the buffer size of the answer***.***

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_GcsGetAnswerSize** (const int *ID*, int* *iAnswerSize*)

Se Gets the size of an answer of an GCS command (**C848_GcsCommandset**() (p.15)).

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_GetInputChannelNames** (const int *ID*, char **szBuffer*, const int *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 (do not confuse relay channels and digital IO channels).

Call C848_qDIO() to get the states of the digital inputs.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_GetOutputChannelNames** (const int *ID*, char **szBuffer*, const int *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 (do not confuse relay channels and digital IO channels). Call C848_DIO() to set the states of the outputs.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_GetRefResult** (const int *ID*, char *const *szAxes*, int * *pnResult*)

Get results of last call to **C848_REF**()(p.*35*), **C848_MNL**() (p.*22*) or **C848_MPL**() (p.*23*). If still referencing or no reference move was started since startup of library, the result is 0. Call **C848_qREF**() (p.*29*) to see which axes have a reference. To reference an axis call **C848_REF**() for axes with reference or **C848_MNL**() (p.*22*) or **C848_MPL**() (p.*23*) for axes without reference. Call **C848_IsReferencing**() to find out if there are axes (still) referencing.

Parameters:

ID 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 **C848_GetSmallestStep** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Get the smallest step of *szAxes* in working units.

Note:

Reports the **theoretical** "resolution" of the axis. There are some stages which accept and report target positions with an accuracy higher than that with which they can actually control their position. Consult the data sheets of the connected stages for the "real-world" resolution.

Parameters:

ID ID of controller
szAxes string with axes
pdValarray array to be filled with the smalles steps for the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_GetWaaResult** (const int *ID*, BOOL* *pbWaaResult*)

Get results of last call to **C848_WAA()** (p.43). If still waiting result is 0. Call **C848_IsWaitingForAllAxes**() (p.19)to find out if the Controller is still waiting.

Parameters:

ID ID of controller
pbWaaResult 1 if successful, 0 if WAA failed or has not finished jet.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_GOH** (const int *ID*, char *const *szAxes*)

Corresponding command: `GOH`

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

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_HasPosChanged** (const const int *ID*, char *const *szAxes*, BOOL * *pbValarray*)

Check if the position of *szAxes* has changed since the last call to **C848_qPOS**() (p.*28*). If the positon of an axis has changed, the corresponding element of the array will be **TRUE**, otherwise it will be **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 has changed its position, **FALSE** if no axis has moved since last call to **C848_qPOS**() (p.*28*).

Parameters:

ID 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 **C848_HID** (const int *ID*, char *const *szAxes*)

Corresponding command: `HID`

Hide *szAxes* from controller display.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_HLT** (const int *ID*, char *const *szAxes*)

Corresponding command: `HLT`

Halt motion of *szAxes* immediately. Does not work for MNL, MPL or REF motion (use **C848_EmergencyStop**() instead)

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_INI** (const int *ID*, char *const *szAxes*)

Corresponding command: `INI`

Initialize *szAxes*. To move *szAxes* again, you must reference them either with **C848_REF**() (p.*35*), **C848_MNL**() (p.*22*) or **C848_MPL**() (p.*23*), even if they were referenced before INI was executed. If any affected axes are under joystick control, the joystick is disabled.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_IsIdle** (const int *ID*, BOOL * *pbIdle*)

Check if controller is currently “idle”, i.e. is not performing some lengthy operation. Implemented as polling with ASCII character #7.

Parameters:

ID ID of controller
pbIdle **TRUE** if idle , **FALSE** otherwise

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_IsMoving** (const const int *ID*, char *const *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.

Parameters:

ID 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 **C848_IsRecordingMacro** (const int *ID*, BOOL * *pbRecordingMacro*)

Check if controller is currently recording a macro.

Note:

This will only check the internal state of the libraray. If someone uses the controller's own terminal to enter a macro, the libraray has no possibility to find out.

Parameters:

ID ID of controller
pbRecordingMacro **TRUE** if recording a macro, **FALSE** otherwise

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_IsReferenceOK** (const int *ID*, char *const *szAxes*, BOOL * *pbValarray*)

Check the reference status of the given axes. Call **C848_qREF**() (p.*29*) to find out which axes have a reference. To reference an axis call **C848_REF**() (p.*35*) for axes with reference or **C848_MNL**() (p.*22*) or **C848_MPL**() (p.*23*) for axes without reference.

Parameters:

ID 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 **C848_IsReferencing** (const int *ID*, char *const *szAxes*, BOOL * *pbIsReferencing*)

Check if C848 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.

Parameters:

ID ID of controller
szAxes string with axes, if "" or **NULL** overall state ir returned.

pbIsReferencing status of axes or controller, **TRUE** if referencing, **FALSE** otherwise

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_IsRunningMacro** (const int *ID*, BOOL * *pbRunningMacro*)

Check if controller is currently running a macro.

Note:

This will only check the internal state of the library. If someone uses the controller's own terminal to enter a macro, the library has no possibility to find out.

Parameters:

ID ID of controller
pbRunningMacro **TRUE** if running a macro, **FALSE** otherwise

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_IsWaitingForAllAxes** (const int *ID*, BOOL * *pbIsWaitingForAllAxes*)

Check if C848 is busy with **WAA**() (p.43).

Parameters:

ID ID of controller
pbIsWaitingForAllAxes **TRUE** if still waiting, **FALSE** otherwise

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_ITD** (const int *ID*, char *const *szAxes*)

Corresponding command: `ITD`

Restore the initial status of *szAxes* as configured upon shipment. Call **C848_RST**() (p.*35*) to restore the status as saved with the last call to **C848_SAV**() (p.*36*). This will implicitly call **C848_INI**() (p.*17*) for *szAxes*, so you must reference the axes again.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_JEN** (const int *ID*, BOOL *bOnOff*)

Corresponding command: `JEN`

Enables the Joystick.

Parameters:

ID ID of controller
bOnOff **TRUE** = Joystick enabled, **FALSE** = Joystick disabled.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_JEN_CALIB** (const int *ID*)

Corresponding command: `JEN CALIB`

Starts the calibration of the Joystick. To get the instructions for the next calibrations step you have to call **C848_GcsGetAnswer()** (p.15) until **C848_GcsGetAnswerSize()** (p.15) returns *iAnswerSize* = 0 (each call of **C848_GcsGetAnswer**() (p.15) returns on line of the instruction, some instructions have more then one line). After the last calibration step **C848_GcsGetAnswer()** (p.15) returns the string "JEN CAL END". If you want to stop calibration, call **C848_EmergencyStop()**.

On the CD there is an example showing how to do an interactive calibration:"C848_JEN_CALIB_Test". If you have installed the Samples, you can find it on your PC in the selected directory under "C-848\Samples". It is a console application written in C but users of other programming languages should also find it easy to understand.

Parameters:

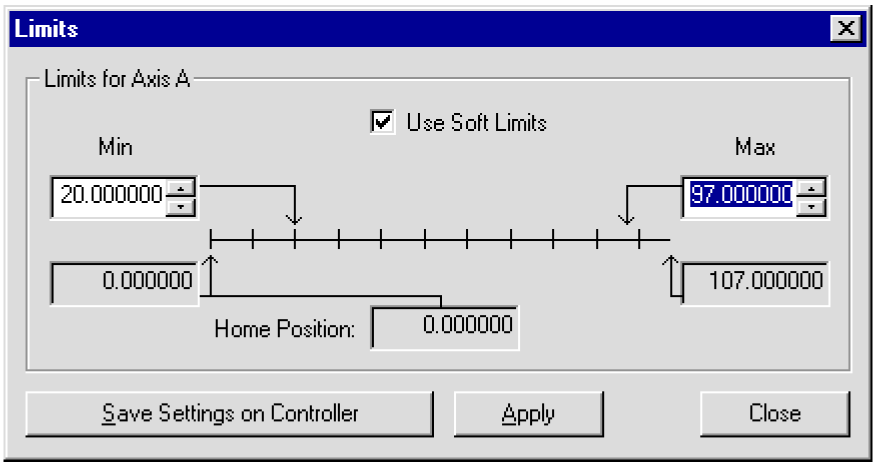
ID ID of controller

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_LimitsDialog** (const int *ID*, const char *cAxis*)

Display dialog to show and edit the limits for *cAxis*.



This dialog shows the limits and travel range for the axis *cAxis*. The limits are the user-defined soft limits (upper edit boxes) and the physical limits (lower read-only text boxes). Only the soft limits can be set by the user with this dialog (with **C848_PLM**() (p.*24*) and **C848_NLM**() (p.*24*)). With the `"Use Soft Limits"` checkbox you can (de)activate the soft limits (see **C848_SSL**() (p.*39*)).

Parameters:

ID ID of controller
cAxis axis for which the limits are shown

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_LoadMacroFromFile** (const int *ID*, char * *szFileName*, char * *szMacroName*)

Load a file and send it to the controller as a macro.

Warning:

The macros sent to the controller are sent "as is". There is no syntax check.

Parameters:

ID ID of controller
szFileName string with name of file to read in
szMacroName string with name of the new macro

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_IN_MACRO_MODE the controller is already recording a macro
PI_MACRO_FILE_ERROR could not open *szFileName*

BOOL **C848_MAC_BEG** (const int *ID*, char * *szName*)

Corresponding command: `MAC BEG`

Start macro recording. This will fail if the controller is already recording a macro.

Parameters:

ID ID of controller
szName name of macro, to be used with **C848_MAC**() (p.*21*) to call it

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_IN_MACRO_MODE if the controller is already recording a macro

BOOL **C848_MAC DEL** (const int *ID*, char * *szName*)

Corresponding command: `MAC DEL`

Delete macro with name *szName*. To find out what macros are available call **C848_qMAC**() (p.*27*).

Parameters:

ID ID of controller
szName name of the macro to delete

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_MAC_END** (const int *ID*)

Corresponding command: MAC `END`

End macro recording. This will fail if the controller is not recording a macro.

Parameters:

ID ID of controller

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_NOT_IN_MACRO_MODE the controller was not recording a macro

BOOL **C848_MAC_START** (const int *ID*, char * *szName*)

Corresponding command: `MAC START`

Start macro with name *szName*. To find out available macors call **C848_qMAC**() (p.*27*).

Parameters:

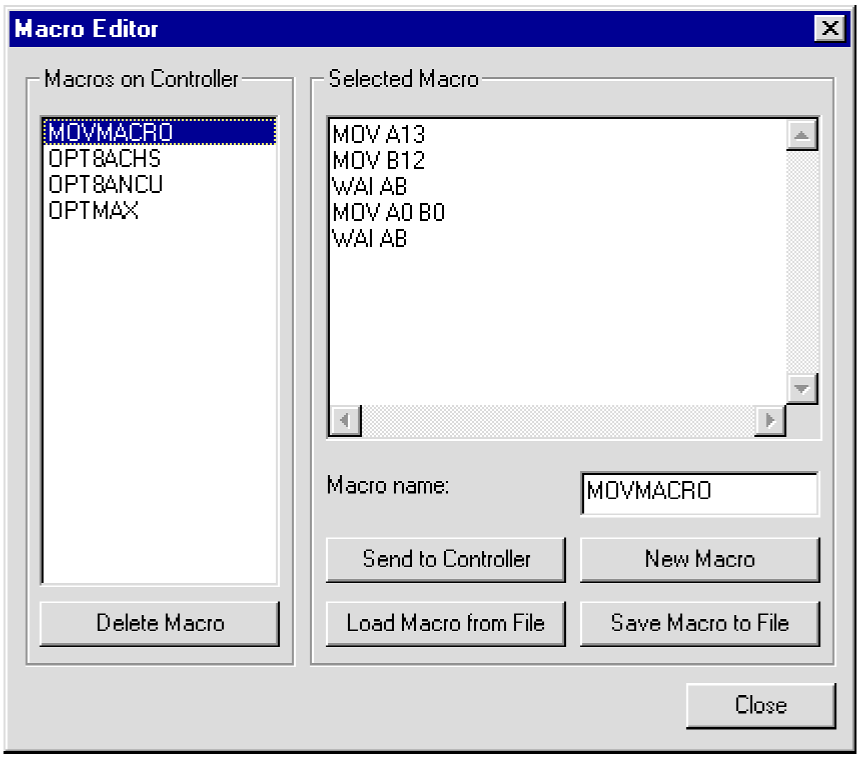
ID ID of controller
szName string with name of the macro to start

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_MacroEditor** (const int *ID*)

Display a dialog with which the user can edit, save and load macros.



Macros on Controller:

```
MOVMACRO
OPT8ACHS
OPT8ANCU
OPTMAX
```

Selected Macro:

```
MOV A13
MOV B12
WAI AB
MOV A0 B0
WAI AB
```

Macro name: MOVMACRO

The left-hand list box in the dialog contains all macros currently stored in the C-848. To edit a macro, you first edit it and then resend it. When you select a macro, its contents will be copied to the edit window.

The text of the macro can be edited in the right-hand edit window. When you have finished editing the macro, you can send the text ("Send to Controller") with the name given in the text field “Macro Name:”. The “New Macro” button will simply clear the edit field and enter the new name in the “Macro Name:” field.

You can load text files to the text window or save its content to disk on the host PC with "Load Macro from File" and "Save Macro to File" buttons. No macro will be sent to or read from the C-848 Controller!

Warning:

The macros sent to the controller are sent "as is". There is no syntax check.

Parameters:

ID ID of controller

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_IN_MACRO_MODE the controller is already recording a macro--this dialog cannot be shown.

BOOL **C848_MNL** (const int *ID*, char *const *szAxes*)

Corresponding command: `MNL`

Move to negative limit switch, initialize motion control chip (including resetting soft limits), set position counter to 0 and reference state to "referenced" for all axes in *szAxes* . This can be used to reference axes without reference switches. Call **C848_IsReferencing**() (p.*18*) to find out if the axes are still moving and **C848_GetRefResult**() (p.*16*) 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 **C848_EmergencyStop**() (p.15) to stop it.

Parameters:

ID 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 **C848_MOV** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `MOV`

Move *szAxes* to absolute position.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_MPL** (const int *ID*, char *const *szAxes*)

Corresponding command: `MPL`

Move to positive limit switch, initialize motion control chip (including resetting soft limits), set position counter to maximum travel value and reference state to "referenced" for all axes in *szAxes* . This can be used to reference axes without references. Call **C848_IsReferencing**() (p.*18*) to find out if the axes are still moving and **C848_GetRefResult**() (p.*16*) 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 **C848_EmergencyStop**() (p.15) to stop it.

Parameters:

ID 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 **C848_MSG** (const int *ID*, char * *szMessage*)

Corresponding command: `MSG`

Display a message on the controller display.

Parameters:

ID ID of controller
szMessage string with message to display

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_MVR** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `MVR`

Move *szAxes* relatively.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_NLM** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `NLM`

Set the negative soft limits of *szAxes*.

Parameters:

ID ID of controller
szAxes string with axes
pdValarray limits for the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_PLM** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `PLM`

Set the positive soft limits of *szAxes*.

Parameters:

ID ID of controller
szAxes string with axes
pdValarray limits for the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_POS** (const int *ID*, char *const *szAxes*, 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 (C848_MVR()) until the actual position is set with this command. See C848_RON() 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.

WARNING:
If the actual position is incorrectly set with this command, stages can be driven into the mechanical hard stop when moving to a position which is thought to be within the travel range of the stage, but actually is not.

Parameters:

ID 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 **C848_qBRA** (const int *ID*, char * *szBuffer*, const int *maxlen*)

Corresponding command: `BRA?`

Get axes with brakes.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qCST** (const int *ID*, char *const *szAxes*, char * *names*, const int *maxlen*)

Corresponding command: `CST?`

Get the type names of the connected stages *szAxes*. The single names begin with the axis identifier+'=' end are separated by '\n' ("line-feed"). For example "A=M-505.1PD\\nB=M-505.2PD".

Parameters:

ID ID of controller
szAxes identifiers of the stages, 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 **C848_qDFF** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `DFF?`

Get scale factors for *szAxes* set with **C848_DFF**() (p.*13*).

Parameters:

ID 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 **C848_qDFH** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `DFH?`

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

Parameters:

ID 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 **C848_qDIO** (const int *ID*, char *const *szChannels*, BOOL * *pbValarray*)

Corresponding command: `DIO?`

Get the states of *szChannels* digital input channel(s) (do not confuse relay channels and digital IO channels).

Parameters:

ID ID of controller
szChannels string with digital input channel identifiers, if "" or **NULL** all channels are affected.
pbValarray states of digital input channels, **TRUE** if "HIGH", **FALSE** if "LOW"

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qDSP** (const int *ID*, char * *szBuffer*, const int *maxlen*)

Corresponding command: `DSP?`

Get axes displayed on controller display. Each character in the returned string is a displayed axis identifier.

Parameters:

ID ID of controller

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qEGE** (const int *ID*, char *const *szAxes*, BOOL * *pbValarray*)

Corresponding command: `EGE?`

Lists the state of electronic gearing for *szAxes*

Parameters:

ID ID of controller
szAxes string with axes, if "" all axes are queried.
pbValarray array of `BOOL` to be filled with status of electronic gearing, **TRUE** if "ON", **FALSE** if "off"

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qERR** (const int *ID*, int * *pError*)

Corresponding command: `ERR?`

Get the error state of the controller. It is safer to call **C848_GetError**()(p.8) because this will also return the internal error state of the library.

Parameters:

ID ID of controller
pnError error code of the controller

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qHID** (const int *ID*, char * *szBuffer*, const int *maxlen*)

Corresponding command: `HID?`

Get axes hidden from controller display. Each character in the returned string is a hidden axis identifier.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qHLP** (const int *ID*, char * *buffer*, const int *maxlen*)

Corresponding command: `HLP?`

Read in the help string of the controller. The answer is quite long (up to 3000 characters) so be sure to provide enough space! (And you may have to wait a bit...)

Parameters:

ID 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 buffer overflow.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qIDN** (const int *ID*, char * *buffer*, const int *maxlen*)

Corresponding command: `*IDN?`

Get identification string of the controller.

Parameters:

ID 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 **C848_qJEN** (const int *ID*, BOOL* *pbOnOff*)

Corresponding command: `JEN?`

Get the state of the Joystick.

Parameters:

ID ID of controller
pbOnOff pointer to storing the state of the Joystick. **TRUE** = Joystick enabled, **FALSE** = Joystick disabled.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qLIM** (const int *ID*, char *const *szAxes*, BOOL * *pbValarray*)

Corresponding command: `LIM?`

Check if the given axes have limit switches

Parameters:

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 **C848_qMAC** (const int *ID*, char * *szName*, char * *szBuffer*, const int *maxlen*)

Corresponding command: `MAC?`

Get available macros or list specific macro. If *szName* is empty or **NULL**, all available macros are listed in *szBuffer*, separated with '\n' ("line-feed"). Otherwise the content of the macro with name *szName* is listed, the single lines separated with '\n'. If there are no macros stored or the requested macro is empty the C848 will return "".

Parameters:

ID ID of controller
szName string with name of the macro to list
szBuffer 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 **C848_qMAS** (const int *ID*, char* const *szAxes*, char* *szMasters*)

Corresponding command: `MAS?`

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

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qMOV** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `MOV?`

Read the commanded target positions for *szAxes*.

Parameters:

ID 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 **C848_qNLM** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `NLM?`

Get the negative soft limits of *szAxes*.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qONT** (const int *ID*, char *const *szAxes*, BOOL * *pbValarray*)

Corresponding command: `ONT?`

Check if *szAxes* have reached target position.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qPLM** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `PLM?`

Get the positive soft limits of *szAxes*.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qPOS** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `POS?`

Get the positions of *szAxes*. You can call **C848_HasPosChanged**() (p.*17*) to find out if this call is necessary.

Parameters:

ID 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 **C848_qREF** (const int *ID*, char *const *szAxes*, BOOL * *pbValarray*)

Corresponding command: `REF?`

Check if the given axes have a reference

Parameters:

ID 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 **C848_qRON** (const int *ID*, char *const *szAxes*, BOOL * *pbValarray*)

Corresponding command: `RON?`

Gets reference mode for given axes. See C848_RON() for a detailed description of reference mode.

Parameters:

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 **C848_qSAI** (const int *ID*, char * *axes*, const int *maxlen*)

Corresponding command: `SAI?`

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

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qSCA** (const int *ID*, char * *pcAxisLeftRight*, char * *pcAxisUpDown*)

Corresponding command: `SCA?`

Get the axes controlled by the cursor keys on the controller's keyboard.

Parameters:

ID ID of controller
pcAxisLeftRight axis controlled by the left and the right key
pcAxisUpDown axis controlled by the up and the down key

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qSJA** (const int *ID*, char* *szAxes*, int *maxlen*)

Corresponding command: `SJA?`

Get axes controlled by the Joystick.

Parameters:

ID ID of controller
szAxes buffer to store the read in string, the axis controlled by Joystick-axis #1 is in *szBuffer*[0] the axis controlled by #2 in *szBuffer*[1].
maxlen size of **buffer**, must be given to avoid a buffer overflow, for Joystick you need 2 characters and one for the terminating '\0'

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qSMO** (const int *ID*, char *const *szAxes*, int * *pnValarray*)

Corresponding command: `SMO?`

Get the motor output.

Parameters:

ID 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 **C848_qSPA** (const int *ID*, char *const *szAxes*, int * *iCmdarray*, double * *dValarray*)

Corresponding 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 parameter ID can have following values:

1 for the P-Term (0 to 32767)
2 for the I-Term (0 to 32767)
3 for the D-Term (0 to 32767)
4 for the I-Limit (0 to 32767)
5 for the Velocity feedforward (0 to 32767)
7 for the motor bias (-32768 to 32767)
8 for the maximum position error (0 to 32767)
9 for the maximum value for the motor output (0 to 32767)
10 for the maximum allowed velocity (- 1.79769313486231E308 to 1.79769313486231E308)
11 for the maximum allowed acceleration (- 1.79769313486231E308 to 1.79769313486231E308)
13 for the maximum allowed Jerk (- 1.79769313486231E308 to 1.79769313486231E308)
14 for the numerator of the counts per physical unit factor (1 to 2147483647)(factor = num./denom.)
15 for the denominator of the counts per physical unit factor (1 to 2147483647)(factor = num./denom.)

Parameters:

ID 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

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, 4, 5, 10, 11}

BOOL **C848_qSRA** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `SRA?`

Gets the electronic gear ratio for *szAxes*,

Parameters:

ID ID of controller
szAxes string with axes
pdValarray array of `double` to be filled with ratios for the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qSSL** (const int *ID*, char *const *szAxes*, BOOL * *pbValarray*)

Corresponding command: `SSL?`

Get the soft limits status for *szAxes*

Parameters:

ID ID of controller
szAxes string with axes, if "" or **NULL** all axes are affected.
pbValarray status of soft limit, **TRUE** if "ON", **FALSE** if "off"

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qSSN** (const int *ID*, int * *pNr*)

Corresponding command: `SSN?`

Get serial number of C-848 controller.

Parameters:

ID ID of controller
pnNr pointer to `int` for storing the serial number.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qSST** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `SST?`

Get the step size for *szAxes*. These step sizes are used when an axis is moved by a joystick or the cursor keys.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qSTA** (const int *ID*, char *const *szAxes*, int * *pnValarray*)

Corresponding command: `STA?`

Get the status of the axes.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise
The status word for each axis is a 16-bit register containing the following information (bit encoding is 0 = LSB, 15 = MSB):

Bit #	Description
0	Motion complete flag. This bit is set (1) when the axis trajectory has completed. This flag is only valid for the S-curve, trapezoidal, and velocity contouring profile modes.
1	Wrap-around condition flag. This bit is set (1) when the axis has reached one end of its travel range and has wrapped to the other end of the travel range. Specifically, when traveling in a positive direction past the position +1,073,741,823, the axis will wrap to position -1,073,741,824, and vice-versa. The bit can be reset with the CLR command.
2	Breakpoint reached flag. This bit is set (1) when one of the breakpoint conditions has occurred.
3	Index pulse received flag. This bit is set (1) when an index pulse has been received.
4	Motion error flag. This bit is set (1) when the maximum position error is exceeded. This bit can only be reset when the axis is no longer in a motion error condition
5	Positive limit switch flag. This bit is set (1) when the positive limit switch goes active.
6	Negative limit switch flag. This bit is set (1) when the negative limit switch goes active.
7	Command error flag. This bit is set (1) when an erroneous command has been received by the motion control chip.
8*	Servo-control on/off status (1 indicates on, 0 indicates off).
9*	Axis on/off status (1 indicates on, 0 indicates off). The C-848 always has the axis ON.
10*	In-motion flag. This bit is continuously updated and indicates whether or not the axis is in motion: 1 indicates axis is in motion, 0 not in motion.
11*	Reserved (may contain 0 or 1)
12*, 13*	Current axis # (13 bit = high bit, 12 bit = low bit). Axis encoding is as follows:
14,15	Reserved (may contain 0 or 1)

Bit 13	Bit12	MC Axis	C-848 Axis
0	0	1	A
0	1	2	B
1	0	3	C
1	1	4	D

BOOL **C848_qSTE** (const int *ID*, const char *cAxis*, const int *iOffset*, const int *nrValues*, double * *pdValarray*)

Corresponding command: `STE?`

Get the recorded positions of a step response. The controller will move the given axis to the target position and record 1024 position values from start. Call **C848_STE**() (p.*40*) to start the step response

Parameters:

ID ID of controller
cAxis axis for which the step response will be 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 1024 positions are stored.
pdValarray Array for storing the position values. Caller is responsible to provide 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 **C848_qSVO** (const int *ID*, char *const *szAxes*, BOOL * *pbValarray*)

Corresponding command: `SVO?`

Get the servo mode for *szAxes*

Parameters:

ID 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 **C848_qTIM** (const int *ID*, int * *pnTime*)

Corresponding command: `TIM?`

Get the internal controller time. The time is the number of servo-loop counts since the controller was started. The duration of one loop is 400 micro seconds.

Parameters:

ID ID of controller
pnTime pointer to `int` for storing the "time" since controller startup

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qTIO** (const int *ID*, int * *pnINr*, int * *pnONr*)

Corresponding command: `TIO?`

Get the number of digital input and output channels installed. Call **C848_GetInputChannelNames()** and **C848_GetInputChannelNames()** to find out how to address them (do not confuse relay channels and digital IO channels).

Parameters:

ID ID of controller
pnINr pointer to `int` for storing the number of digital input channels installed
pnONr pointer to `int` for storing the number of digital output channels installed

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qTMN** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `TMN?`

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

Parameters:

ID 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 **C848_qTMX** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `TMX?`

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

Parameters:

ID 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 **C848_qTNJ** (const int *ID*, int* *pNr*)

Corresponding command: `TNJ?`

Get the number of Installed Joysticks.

Parameters:

ID ID of controller
pNr pointer to *int* storing the *pnNr* number of connected joysticks.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qTVI** (const int *ID*, char * *axes*, const int *maxlen*)

Corresponding 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.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qVEL** (const int *ID*, char *const *szAxes*, double * *valarray*)

Corresponding command: `VEL?`

Get the velocities of *szAxes*.

Parameters:

ID 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 **C848_qVER** (const int *ID*, char * *buffer*, const int *maxlen*)

Corresponding command: `VER?`

Get version of the controller firmware.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_qVST** (const const int *ID*, char * *buffer*, const int *maxlen*)

Corresponding command: `VST?`

Get the names of the available stage types.

Parameters:

ID 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 **C848_REF** (const int *ID*, char *const *szAxes*)

Corresponding command: `REF`

Reference move of *szAxes*. Call **C848_IsReferencing**() (p.*18*) to find out if the axes are still moving and **C848_GetRefResult**() (p.*16*) 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 **C848_EmergencyStop**() (p.15) to stop it.

Parameters:

ID ID of controller
szAxes string with axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_RON** (const int *ID*, char *const *szAxes*, 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 (C848_REF()) or, if no reference switch is available, to a limit switch (using C848_MPL() C848_MNL()) 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 (C848_MVR()), unless the actual position is set with C848_POS(). Afterwards, relative and absolute moves can be commanded.

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

WARNINGS:
- ➢ If reference mode is switched off, and relative movements are commanded, stages can be driven into the mechanical hard stop if moving to a position which is outside the travel range!
- ➢ If reference mode is switched off, and the actual position is incorrectly set with C848_POS(), stages can be driven into the mechanical hard stop when moving to a position which is thought to be within the travel range of the stage, but actually is not.

Parameters:

ID 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 has no reference or limit switches, and reference mode can not be switched ON

BOOL **C848_RST** (const int *ID*, char *const *szAxes*)

Restore the status of *szAxes* as saved with the last call to **C848_SAV**() (p.*36*). Call **C848_ITD**() (p.*19*) to restore the initial status as configured on shipment. This will implicitly call **C848_INI**() (p.*17*) for *szAxes*, so you must reference the axes again.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_SAI** (const int *ID*, char *const *szOldAxes*, char *const *szNewAxes*)

Corresponding command: `SAI`

Rename connected axes. *szOldAxes[index]* will bew set to `szNewAxes[index]`. User can set the "names" of axes with this function. The characters in *szNewAxes* character must not be in use for an other existing axes and must be one of the valid identifiers. All characters in *szNewAxes* will be converted to uppercase letters. To find out which characters are valid, call **C848_qTVI**() (p.*34*). Only the **last** occurence of an axis identifier in *szNewAxes* will be used to change the name.

Parameters:

ID ID of controller
szOldAxes old identifiers of the axes
szNewAxes new identifiers of the axes

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_INVALID_AXIS_IDENTIFIER if the characters are not valid
PI_UNKNOWN_AXIS_IDENTIFIER if *szOldAxes* contains 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 **C848_SAV** (const int *ID*, char *const *szAxes*)

Corresponding command: `SAV`

Save current status of *szAxes*. To read in the status call **C848_RST**() (p.*35*). Call **C848_ITD**() (p.*19*) to restore the initial status as configured on shipment.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_SaveMacroToFile** (const int *ID*, char * *szFileName*, char * *szMacroName*)

Save a macro from the controller to the local disc.

Warning:

If the filename points to an exisiting file, it will be overwritten!

Parameters:

ID ID of controller
szFileName string with name of file to create
szMacroName string with name of the macro

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_IN_MACRO_MODE the controller is recording a macro
PI_NO_MACRO_OR_EMPTY no macro *szMacroName*
PI_MACRO_FILE_ERROR could not open *szFileName*

BOOL **C848_SCA** (const int *ID*, char *cAxisLeftRight*, char *cAxisUpDown*)

Corresponding command: `SCA`

Set the axes controlled by the cursor keys on the controller's keyboard.

Parameters:

ID ID of controller
cAxisLeftRight axis controlled by the left and the right key
cAxisUpDown axis controlled by the up and the down key

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_UNKNOWN_AXIS_IDENTIFIER one (or both) of *cAxisLeftRight* and *cAxisUpDown* is not valid

BOOL **C848_SJA** (const int *ID*, char* *szAxes*, int* *pJoystikAxisNr*)

Corresponding command: `SJA`

Set axes controlled by the Joystick.

Parameters:

ID ID of controller
szAxes string with axes
pJoystikAxisNr array with numbers of Joystick-axis which will control the axes.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_SMO** (const int *ID*, char *const *szAxes*, int * *pnValarray*)

Corresponding command: `SMO`

Set the motor output directly. This is only possible if servo-control is "off" (see **C848_SVO**() (p.*41*)).

Parameters:

ID ID of controller
szAxes string with axes
pnValarray array with parameters. all must be in [-32767 - 32767]

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_SPA** (const int *ID*, char *const *szAxes*, int * *iCmdarray*, double * *dValarray*)

Corresponding 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 parameter ID can have following values:

1 for the P-Term (0 to 32767)
2 for the I-Term (0 to 32767)
3 for the D-Term (0 to 32767)
4 for the I-Limit (0 to 32767)
5 for the Velocity feedforward (0 to 32767)
7 for the motor bias (-32768 to 32767)
8 for the maximum position error (0 to 32767)
9 for the maximum value for the motor output (0 to 32767)
10 for the maximum allowed velocity (- 1.79769313486231E308 to 1.79769313486231E308)
11 for the maximum allowed acceleration (- 1.79769313486231E308 to 1.79769313486231E308)
13 for the maximum allowed Jerk (- 1.79769313486231E308 to 1.79769313486231E308)
14 for the numerator of the counts per physical unit factor (1 to 2147483647)(factor = num./denom.)
15 for the denominator of the counts per physical unit factor (1 to 2147483647)(factor = num./denom.)

Unlike the other functions, C848_SPA has two arrays as arguments. The first array has the parameters which have to be modified, the second one the values. If you want to set the P-Term (ID=1) to 100, the I-Term (ID=2) to 25 and the D-Term (ID=3) to 200, you can use the following code (in C(++) syntax):

```
char szAxes[] = "AAA";
int cmd[] = {1, 2, 3};
double values[] = {100, 25, 200};
E816_SPA(id, szAxes, cmd, values);
```

szAxes = "AAA"	cmd = {1, 2, 3}	values = {100, 25, 200}
szAxes[0] = 'A'	cmd[0] = 1	values[0] = 100
szAxes[1] = 'A'	cmd[1] = 2	values[1] = 25
szAxes[2] = 'A'	cmd[2] = 3	values[2] = 200

Note:

If the same axis has the same parameter ID more than once, only the **last** value will be set. For example `C848_SPA(id, "AAA", {1, 1, 2}, {100, 200, 30})` will set the P-term of '`A`' to 200 and the I-term to 30.

Parameters:

ID ID of controller
szAxes axis for which the parameter should be set
iCmdarray IDs of parameter
dValarray array with the values for the parameters

Returns:

TRUE if successful, **FALSE** otherwise

Errors:

PI_INVALID_SPA_CMD_ID *one* of the IDs in *iCmdarray* is not valid, must be one of {1,2,3, 4, 5, 10, 11}

BOOL **C848_SRA** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `SRA`

Sets the electronic gear ratio for *szAxes*, The ratio is defined as (slave move)/(master move)

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_SSL** (const int *ID*, char *const *szAxes*, BOOL * *pbValarray*)

Corresponding command: `SSL`

Set soft limits "on" or "off". If *pbValarray[index]* is **FALSE** the mode is "off", otherwise it is set to "on"

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_SST** (const int *ID*, char *const *szAxes*, double * *pdValarray*)

Corresponding command: `SST`

Set the step size for *szAxes* in working units. These step sizes are used when an axis is moved by a joystick or the cursor keys. If you change the scale factor for physical units (see C848_DFF) , the step size isphysically unchanged, i.e. the numerical values reported by C848_qSST() change but the steps the stage actually performs stay unchanged in size.

Parameters:

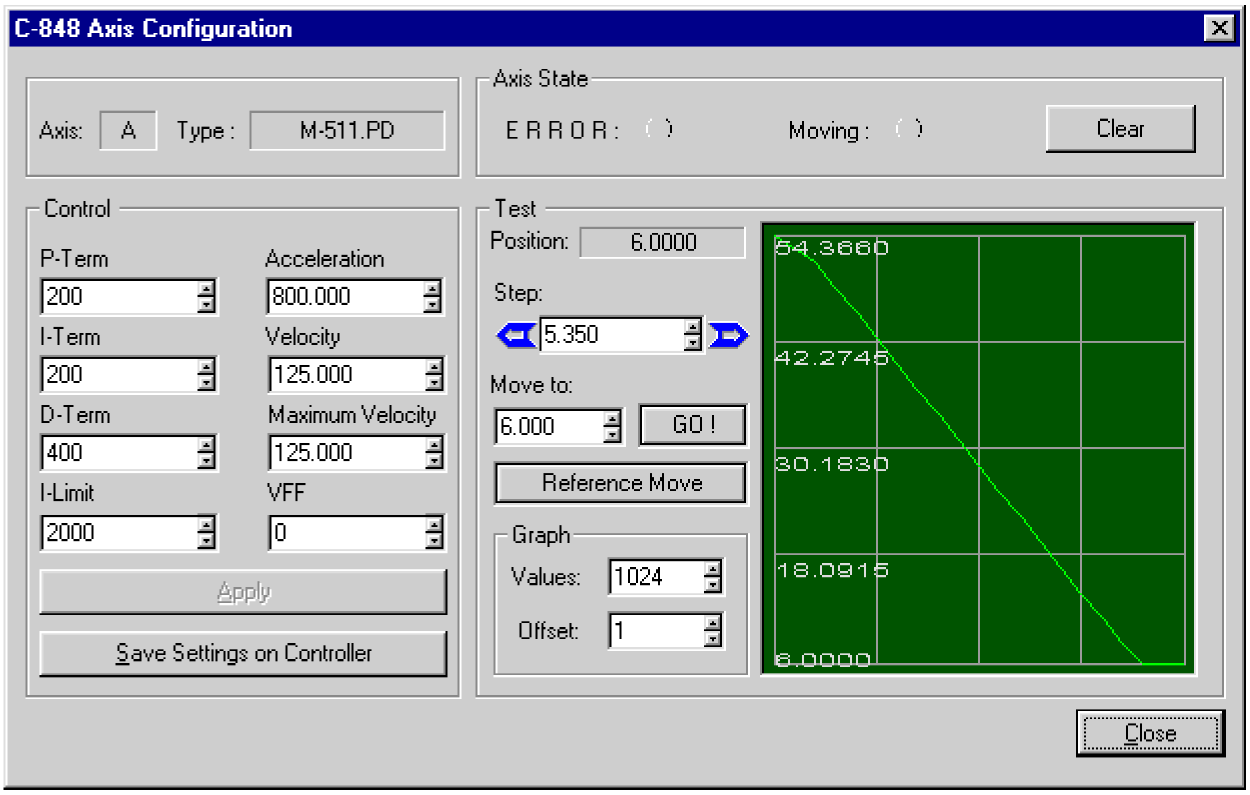
ID ID of controller
szAxes string with axes
pdValarray step size of the axes

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_StageConfigDlg** (const int *ID*, char *cAxis*)

Display an axis configuration dialog. With this dialog you can edit and test the parameters for the motor servo-loop controller.



In the "Axis State" pane you can watch the device moving, see if an error condition is set or clear the state (with the C-848 CLR command).

“Control” has all the parameters used for the control of the axis. If you make changes, use the “Apply” button to send them to the C-848.

In the “Test” pane are the tools to test your settings. If the axis has been referenced (e.g. with “Reference Move”) you can perform a “step response test” with the offset specified in the "Step" field and in the direction indicated by the blue arrow clicked. The C-848 will move relative to the current position and log the position of the axis.

Up to 1024 logged positions can be stored. After the movethe positions will be displayed graphically to help you check the settings. Only the positions specified with “Values” and “Offset” are shown. “Offset” is the index of the first position to display, and “Values” is the number of positions. If, for example, you want to see the first 200 positions set “Values” to 200 and “Offset” to 0. If you do not need the first 50 positions but want the next 500, set “Values” to 500 and “Offset” to 50.

Parameters:

ID ID of controller
cAxis axis identifier of stage

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_STE** (const int *ID*, const char *cAxis*, double *dOffset*)

Corresponding command: `STE`

Record step response for one axis. The controller will move the given axis relative to the current position and record 1024 position values from start. Call **C848_qSTE**() (p.*32*) to read them.

Parameters:

ID ID of controller

cAxis axis for which the step response will be recorded
dOffset position offset *cAxis*

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_STP** (const int *ID*)

Corresponding command: `STP`

Stop all axes. Does not work for MNL, MPL or REF motion (use **C848_EmergencyStop**() instead)

Parameters:

ID ID of controller

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_SVO** (const int *ID*, char *const *szAxes*, BOOL * *pbValarray*)

Corresponding command: `SVO`

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

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_SystemAbort** (const int *ID*)

Abort system. The C-848 won't respond after this call, and *ID* is not valid any longer. You must restart the C-848.

Parameters:

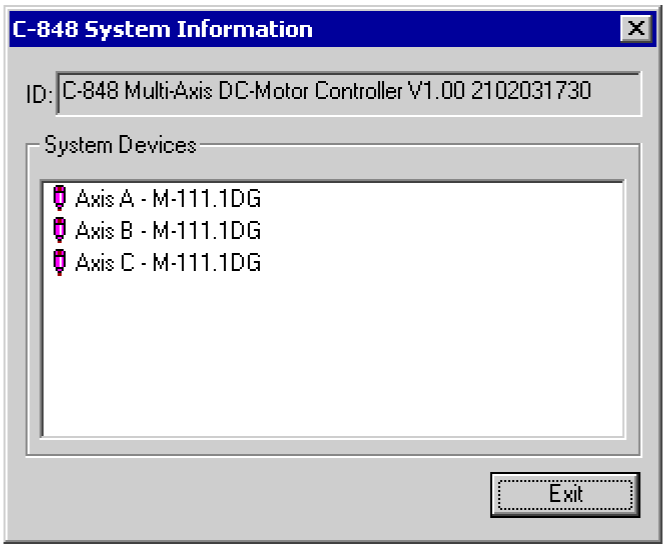
ID ID of controller

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_SystemInfoDlg** (const int *ID*)

Display a dialog showing the connected stages of the controller.


C-848 System Information

ID: C-848 Multi-Axis DC-Motor Controller V1.00 2102031730

System Devices

Axis A - M-111.1DG
Axis B - M-111.1DG
Axis C - M-111.1DG

Parameters:

ID ID of controller

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_VEL** (const int *ID*, char *const *szAxes*, double * *valarray*)

Corresponding command: `VEL`

Set the velocities of *szAxes*.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_VMO** (const int *ID*, char *const *szAxes*, double * *pdValarray*, BOOL * *pbMovePossible*)

Corresponding command: `VMO`

"Virtual" move to check if given position can be reached.

Note:

The controller will return "0" for success and "1" if one of the positions is out of limits. This function will map this to **TRUE** (1) if successful and **FALSE** (0) if out of limits (a more "natural" interpretation for C programers).

Parameters:

ID ID of controller
szAxes string with axes
pdValarray positions of the axes
pbMovePossible result of VMO, **TRUE** if position can be reached, **FALSE** if not

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_WAA** (const int *ID*, unsigned int *iWaitTime*)

Corresponding command: `WAA`

Waits a specified time until all axes have finished their movement. If all axes finished their movement within the specified time, the function **C848_GetWaaResult**() returns **TRUE** for *pbWaaResult*. Otherwise it returns **FALSE**.

Parameters:

ID ID of controller
iWaitTime the time to wait in milliseconds.

Returns:

TRUE if successful, **FALSE** otherwise

BOOL **C848_WAI** (const int *ID*, char *const *szAxes*)

Corresponding command: `WAI`

Wait for *szAxes*.

Note:

This call will send “WAI *szAxes*” to the controller and return immediatly. Only the controller will wait for *szAxes*.

Parameters:

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

Returns:

TRUE if successful, **FALSE** otherwise

10. Error Codes

10.1. Defines

- #define **COM_NO_ERROR** 0
- #define **COM_ERROR** -1
- #define **SEND_ERROR** -2
- #define **REC_ERROR** -3
- #define **NOT_CONNECTED_ERROR** -4
- #define **COM_BUFFER_OVERFLOW** -5
- #define **CONNECTION_FAILED** -6
- #define **COM_TIMEOUT** -7
- #define **COM_MULTILINE_RESPONSE** -8
- #define **COM_INVALID_ID** -9
- #define **COM_NOTIFY_EVENT_ERROR** -10
- #define **COM_NOT_IMPLEMENTED** -11
- #define **COM_ECHO_ERROR** -12
- #define **COM_GPIB_EDVR** -13
- #define **COM_GPIB_ECIC** -14
- #define **COM_GPIB_ENOL** -15
- #define **COM_GPIB_EADR** -16
- #define **COM_GPIB_EARG** -17
- #define **COM_GPIB_ESAC** -18
- #define **COM_GPIB_EABO** -19
- #define **COM_GPIB_ENEB** -20
- #define **COM_GPIB_EDMA** -21
- #define **COM_GPIB_EOIP** -22
- #define **COM_GPIB_ECAP** -23
- #define **COM_GPIB_EFSO** -24
- #define **COM_GPIB_EBUS** -25
- #define **COM_GPIB_ESTB** -26
- #define **COM_GPIB_ESRQ** -27
- #define **COM_GPIB_ETAB** -28
- #define **COM_GPIB_ELCK** -29
- #define **COM_RS_INVALID_DATA_BITS** -30
- #define **COM_ERROR_RS_SETTINGS** -31
- #define **COM_INTERNAL_RESOURCES_ERROR** -32
- #define **COM_DLL_FUNC_ERROR** -33
- #define **IS_COM_ERROR**(err) ((err)<0 && (err)>COM_MAX_ERROR)
- #define **PI_UNKNOWN_AXIS_IDENTIFIER** (COM_MAX_ERROR - 1)
- #define **PI_NR_NAV_OUT_OF_RANGE** (COM_MAX_ERROR - 2)
- #define **PI_INVALID_SGA** (COM_MAX_ERROR - 3)
- #define **PI_UNEXPECTED_RESPONSE** (COM_MAX_ERROR - 4)
- #define **PI_NO_MANUAL_PAD** (COM_MAX_ERROR - 5)
- #define **PI_INVALID_MANUAL_PAD_KNOB** (COM_MAX_ERROR - 6)
- #define **PI_INVALID_MANUAL_PAD_AXIS** (COM_MAX_ERROR - 7)
- #define **PI_CONTROLLER_BUSY** (COM_MAX_ERROR - 8)
- #define **PI_THREAD_ERROR** (COM_MAX_ERROR - 9)
- #define **PI_IN_MACRO_MODE** (COM_MAX_ERROR - 10)
- #define **PI_NOT_IN_MACRO_MODE** (COM_MAX_ERROR - 11)
- #define **PI_MACRO_FILE_ERROR** (COM_MAX_ERROR - 12)
- #define **PI_NO_MACRO_OR_EMPTY** (COM_MAX_ERROR - 13)
- #define **PI_MACRO_EDITOR_ERROR** (COM_MAX_ERROR - 14)
- #define **PI_INVALID_ARGUMENT** (COM_MAX_ERROR - 15)
- #define **PI_AXIS_ALREADY_EXISTS** (COM_MAX_ERROR - 16)
- #define **PI_INVALID_AXIS_IDENTIFIER** (COM_MAX_ERROR - 17)
- #define **PI_COM_ARRAY_ERROR** (COM_MAX_ERROR - 18)
- #define **PI_COM_ARRAY_RANGE_ERROR** (COM_MAX_ERROR - 19)

- #define **PI_INVALID_SPA_CMD_ID** (COM_MAX_ERROR - 20)
- #define **PI_NR_AVG_OUT_OF_RANGE** (COM_MAX_ERROR - 21)
- #define **PI_WAV_SAMPLES_OUT_OF_RANGE** (COM_MAX_ERROR - 22)
- #define **PI_WAV_FAILED** (COM_MAX_ERROR - 23)
- #define **PI_MOTION_ERROR** (COM_MAX_ERROR - 24)
- #define **PI_RUNNING_MACRO** (COM_MAX_ERROR - 25)
- #define **PI_PZT_CONFIG_FAILED** (COM_MAX_ERROR - 26)
- #define **PI_PZT_CONFIG_INVALID_PARAMS** (COM_MAX_ERROR - 27)
- #define **PI_UNKNOWN_CHANNEL_IDENTIFIER** (COM_MAX_ERROR - 28)
- #define **PI_WAVE_PARAM_FILE_ERROR** (COM_MAX_ERROR - 29)
- #define **PI_UNKNOWN_WAVE_MACRO** (COM_MAX_ERROR - 30)
- #define **PI_WAVE_MACRO_FUNC_NOT_LOADED** (COM_MAX_ERROR - 31)
- #define **PI_USER_CANCELLED** (COM_MAX_ERROR - 32)
- #define **PI_DLL_NOT_LOADED** (COM_MAX_ERROR - 34)
- #define **PI_PARAMETER_FILE_PROTECTED** (COM_MAX_ERROR - 35)
- #define **PI_NO_PARAMETER_FILE_OPENED** (COM_MAX_ERROR - 36)
- #define **PI_STAGE_DOES_NOT_EXIST** (COM_MAX_ERROR - 37)
- #define **PI_PARAMETER_FILE_ALLREADY_OPENED** (COM_MAX_ERROR - 38)
- #define **PI_PARAMETER_FILE_OPEN_ERROR** (COM_MAX_ERROR - 39)
- #define **PI_INVALID_CONTROLLER_VERSION** (COM_MAX_ERROR - 40)
- #define **PI_PARAM_SET_ERROR** (COM_MAX_ERROR - 41)
- #define **PI_NUMBER_OF_POSSIBLE_WAVES_EXCEEDED** (COM_MAX_ERROR - 42)
- #define **PI_NUMBER_OF_POSSIBLE_GENERATORS_EXCEEDED** (COM_MAX_ERROR - 43)
- #define **PI_NO_WAVE_FOR_AXIS_DEFINED** (COM_MAX_ERROR - 44)
- #define **PI_CANT_STOP_OR_START_WAV** (COM_MAX_ERROR - 45)
- #define **PI_REFERENCE_ERROR** (COM_MAX_ERROR - 46)
- #define **PI_REQUIRED_WAVE_MACRO_NOT_FOUND** (COM_MAX_ERROR - 47)
- #define **PI_INVALID_SPP_CMD_ID** (COM_MAX_ERROR - 48)
- #define **PI_STAGE_NAME_ISNT_UNIQUE** (COM_MAX_ERROR - 49)
- #define **PI_FILE_TRANSFER_BEGIN_MISSING** (COM_MAX_ERROR - 50)
- #define **PI_FILE_TRANSFER_ERROR_TEMP_FILE** (COM_MAX_ERROR - 51)
- #define **PI_FILE_TRANSFER_CRC_ERROR** (COM_MAX_ERROR - 52)
- #define **PI_CNTR_NO_ERROR** 0
- #define **PI_CNTR_PARAM_SYNTAX** 1
- #define **PI_CNTR_UNKNOWN_COMMAND** 2
- #define **PI_CNTR_MOVE_WITHOUT_INI** 5
- #define **PI_CNTR_INVALID_SGA_PARAM** 6
- #define **PI_CNTR_POS_OUT_OF_LIMITS** 7
- #define **PI_CNTR_VEL_OUT_OF_LIMITS** 8
- #define **PI_CNTR_SET_PIVOT_NOT_POSSIBLE** 9
- #define **PI_CNTR_STOP** 10
- #define **PI_CNTR_SST_OR_SCAN_RANGE** 11
- #define **PI_CNTR_INVALID_SCAN_AXES** 12
- #define **PI_CNTR_INVALID_NAV_PARAM** 13
- #define **PI_CNTR_INVALID_ANALOG_INPUT** 14
- #define **PI_CNTR_INVALID_AXIS_IDENTIFIER** 15
- #define **PI_CNTR_INVALID_STAGE_NAME** 16
- #define **PI_CNTR_PARAM_OUT_OF_RANGE** 17
- #define **PI_CNTR_INVALID_MACRO_NAME** 18
- #define **PI_CNTR_MACRO_RECORD** 19
- #define **PI_CNTR_MACRO_NOT_FOUND** 20
- #define **PI_CNTR_AXIS_HAS_NO_BRAKE** 21
- #define **PI_CNTR_DOUBLE_AXIS** 22
- #define **PI_CNTR_INVALID_AXIS** 23
- #define **PI_CNTR_PARAM_NR** 24
- #define **PI_CNTR_INVALID_REAL_NR** 25
- #define **PI_CNTR_MISSING_PARAM** 26
- #define **PI_CNTR_SOFT_LIMIT_OUT_OF_RANGE** 27
- #define **PI_CNTR_NO_MANUAL_PAD** 28
- #define **PI_CNTR_NO_JUMP** 29

- #define **PI_CNTR_INVALID_JUMP** 30
- #define **PI_CNTR_AXIS_HAS_NO_REFERENCE** 31
- #define **PI_CNTR_STAGE_HAS_NO_LIM_SWITCH** 32
- #define **PI_CNTR_NO_RELAY_CARD** 33
- #define **PI_CNTR_CMD_NOT_ALLOWED_FOR_STAGE** 34
- #define **PI_CNTR_NO_DIGITAL_INPUT** 35
- #define **PI_CNTR_NO_DIGITAL_OUTPUT** 36
- #define **PI_CNTR_INVALID_CNTR_NUMBER** 39
- #define **PI_CNTR_NO_JOYSTICK_CONNECTED** 40
- #define **PI_CNTR_INVALID_EGE_AXIS** 41
- #define **PI_CNTR_SLAVE_POSITION_OUT_OF_RANGE** 42
- #define **PI_CNTR_JOYSTICK_CALIBRATION_FAILED** 44
- #define **PI_CNTR_REFERENCING_FAILED** 45
- #define **PI_CNTR_MOVE_TO_LIMIT_SWITCH_FAILED** 49
- #define **PI_CNTR_REF_WITH_REF_DISABLED** 50
- #define **PI_CNTR_AXIS_UNDER_JOYSTICK_CONTROL** 51
- #define **PI_CNTR_COMMUNICATION_ERROR** 52
- #define **PI_CNTR_NO_AXIS** 200
- #define **PI_CNTR_NO_AXIS_PARAM_FILE** 201
- #define **PI_CNTR_INVALID_AXIS_PARAM_FILE** 202
- #define **PI_CNTR_NO_AXIS_PARAM_BACKUP** 203
- #define **PI_CNTR_RESERVED_204** 204
- #define **PI_CNTR_SMO_WITH_SERVO_ON** 205
- #define **PI_CNTR_UUDECODE_INCOMPLETE_HEADER** 206
- #define **PI_CNTR_UUDECODE_NOTHING_TO_DECODE** 207
- #define **PI_CNTR_UUDECODE_ILLEGAL_FORMAT** 208
- #define **PI_CNTR_CRC32_ERROR** 209
- #define **PI_CNTR_ILLEGAL_FILENAME** 210
- #define **PI_CNTR_SENDING_BUFFER_OVERFLOW** 301
- #define **PI_CNTR_VOLTAGE_OUT_OF_LIMITS** 302
- #define **PI_CNTR_VOLTAGE_SET_WHEN_SERVO_ON** 303
- #define **PI_CNTR_RECEIVING_BUFFER_OVERFLOW** 304
- #define **PI_CNTR_EEPROM_ERROR** 305
- #define **PI_CNTR_I2C_ERROR** 306
- #define **PI_CNTR_RECEIVING_TIMEOUT** 307
- #define **PI_CNTR_TOO_MANY_NESTED_MACROS** 1000
- #define **PI_CNTR_MACRO_ALREADY_DEFINED** 1001
- #define **PI_CNTR_NO_MACRO_RECORDING** 1002
- #define **PI_CNTR_INVALID_MAC_PARAM** 1003
- #define **PI_CNTR_RESERVED_1004** 1004
- #define **PI_CNTR_ALREADY_HAS_SERIAL_NUMBER** 2000
- #define **PI_CNTR_SECTOR_ERASE_FAILED** 4000
- #define **PI_CNTR_FLASH_PROGRAM_FAILED** 4001
- #define **PI_CNTR_FLASH_READ_FAILED** 4002
- #define **PI_CNTR_HW_MATCHCODE_ERROR** 4003
- #define **PI_CNTR_FW_MATCHCODE_ERROR** 4004
- #define **PI_CNTR_HW_VERSION_ERROR** 4005
- #define **PI_CNTR_FW_VERSION_ERROR** 4006

10.2. Detailed Description

The error codes listed here are those of the PI General Command Set. As such, some are not relevant to the C-848 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 C848 DLL.

10.3. Define Documentation

```
#define COM_BUFFER_OVERFLOW  -5
```

Buffer overflow

```
#define COM_DLL_FUNC_ERROR  -33
```

A DLL or one of the required functions could not be loaded

```
#define COM_ECHO_ERROR  -12
```

Error while sending "echoed" data

```
#define COM_ERROR  -1
```

Error during com operation (could not be specified)

```
#define COM_ERROR_RS_SETTINGS  -31
```

RS-232: Error when configuring the COM port

```
#define COM_GPIB_EABO  -19
```

IEEE488: I/O operation aborted

```
#define COM_GPIB_EADR  -16
```

IEEE488: Interface board not addressed correctly

```
#define COM_GPIB_EARG  -17
```

IEEE488: Invalid argument to function call

```
#define COM_GPIB_EBUS  -25
```

IEEE488: Command error during device call

```
#define COM_GPIB_ECAP  -23
```

IEEE488: No capability for intended operation

```
#define COM_GPIB_ECIC  -14
```

IEEE488: Function requires GPIB board to be CIC

```
#define COM_GPIB_EDMA  -21
```

IEEE488: Error performing DMA

```
#define COM_GPIB_EDVR  -13
```

IEEE488: System error

```
#define COM_GPIB_EFSO  -24
```

IEEE488: File system operation error

```
#define COM_GPIB_ELCK  -29
```

IEEE488: Address or board is locked.

```
#define COM_GPIB_ENEB  -20
```

IEEE488: Non-existent interface board

```
#define COM_GPIB_ENOL  -15
```

IEEE488: Write function detected no Listeners

```
#define COM_GPIB_EOIP  -22
```

IEEE488: I/O operation started before previous operation completed

```
#define COM_GPIB_ESAC  -18
```

IEEE488: Function requires GPIB board to be SAC

```
#define COM_GPIB_ESRQ  -27
```

IEEE488: SRQ remains asserted

```
#define COM_GPIB_ESTB  -26
```

IEEE488: Serial poll status byte lost

```
#define COM_GPIB_ETAB  -28
```

IEEE488: The return buffer is full.

```
#define COM_INTERNAL_RESOURCES_ERROR  -32
```

Error when dealing with internal system resources (events, threads, ...)

```
#define COM_INVALID_ID  -9
```

There is no interface open with the given ID

```
#define COM_MULTILINE_RESPONSE  -8
```

There are more lines waiting in buffer

```
#define COM_NO_ERROR  0
```

No error occurred during function call

```
#define COM_NOT_IMPLEMENTED  -11
```

The function was not implemented (e.g. only RS-232 communication provides this feature and it was called for IEEE488)

```
#define COM_NOTIFY_EVENT_ERROR  -10
```

The event for the notification could not be opened

```
#define COM_RS_INVALID_DATA_BITS  -30
```

RS-232: The use of 5 data bits with 2 stop bits is an invalid combination, as is 6, 7, or 8 data bits with 1.5 stop bits.

```
#define COM_TIMEOUT  -7
```

Timeout error

```
#define CONNECTION_FAILED  -6
```

Error while opening port

```
#define IS_COM_ERROR(err)  ( (err)<0 && (err)>COM_MAX_ERROR )
```

Simple macro to check if an error was caused by basic communication functions.

```
#define NOT_CONNECTED_ERROR  -4
```

Not connected (no port with given ID open)

```
#define PI_AXIS_ALREADY_EXISTS  (COM_MAX_ERROR - 16)
```

Axis identifier is already in use for a connected stage

```
#define PI_CANT_STOP_OR_START_WAV  (COM_MAX_ERROR - 45)
```

Attempt to stop wave output on an axis when it's already stopped, or start it when it's already started

```
#define PI_CNTR_ALREADY_HAS_SERIAL_NUMBER  2000
```

Controller already has a serial number

```
#define PI_CNTR_AXIS_HAS_NO_BRAKE  21
```

Axis has no brake

```
#define PI_CNTR_AXIS_HAS_NO_LIM_SWITCH  32
```

Axis has no limit switch

```
#define PI_CNTR_AXIS_HAS_NO_REFERENCE  31
```

Axis has no reference sensor

```
#define PI_CNTR_AXIS_UNDER_JOYSTICK_CONTROL  51
```

Selected axis is controlled by joystick

```
#define PI_CNTR_CMD_NOT_ALLOWED_FOR_STAGE  34
```

The last command was not allowed for selected stage(s)

```
#define PI_CNTR_COMMUNICATION_ERROR  209
```

Controller detected communication error

```
#define PI_CNTR_CRC32_ERROR  209
```

CRC32 error

```
#define PI_CNTR_DOUBLE_AXIS  22
```

Axis identifier given more than once

```
#define PI_CNTR_EEPROM_ERROR  305
```

Error while reading/writing EEPROM

```
#define PI_CNTR_FLASH_PROGRAM_FAILED  4001
```

Flash programm failed

```
#define PI_CNTR_FLASH_READ_FAILED  4002
```

Flash read failed

```
#define PI_CNTR_FW_MATCHCODE_ERROR  4004
```

FW Matchcode missing/invalid

```
#define PI_CNTR_FW_VERSION_ERROR  4006
```

FW Mark missing/invalid

```
#define PI_CNTR_HW_MATCHCODE_ERROR  4003
```

HW Matchcode missing/invalid

```
#define PI_CNTR_HW_VERSION_ERROR  4005
```

FW Version missing/invalid

```
#define PI_CNTR_I2C_ERROR  306
```

Error on I2C bus

```
#define PI_CNTR_ILLEGAL_FILENAME  210
```

Illegal file name

```
#define PI_CNTR_INVALID_ANALOG_INPUT  14
```

Invalid analog channel

```
#define PI_CNTR_INVALID_AXIS  23
```

Invalid axis

```
#define PI_CNTR_INVALID_AXIS_IDENTIFIER  15
```

Invalid axis identifier

```
#define PI_CNTR_INVALID_AXIS_PARAM_FILE 202
```
Invalid axis parameter file

```
#define PI_CNTR_INVALID_CNTR_NUMBER 39
```
Controller number invalid

```
#define PI_CNTR_INVALID_EGE_AXIS 41
```
Invlaid axis for electronic gearing, axis can not be slave

```
#define PI_CNTR_INVALID_JUMP 30
```
No step response values recorded

```
#define PI_CNTR_INVALID_MAC_PARAM 1003
```
Invalid parameter for `MAC`

```
#define PI_CNTR_INVALID_MACRO_NAME 18
```
Invalid macro name

```
#define PI_CNTR_INVALID_NAV_PARAM 13
```
Parameter for `NAV` out of range

```
#define PI_CNTR_INVALID_REAL_NR 25
```
Invalid floating point number

```
#define PI_CNTR_INVALID_SCAN_AXES 12
```
Invalid axis combination for fast scan

```
#define PI_CNTR_INVALID_SGA_PARAM 6
```
Parameter for `SGA` not valid

```
#define PI_CNTR_INVALID_STAGE_NAME 16
```
Unknown stage name

```
#define PI_CNTR_JOYSTICK_CALIBRATION_FAILED 44
```
Calibration of joystick failed

```
#define PI_CNTR_MACRO_ALREADY_DEFINED 1001
```
Macro already defined

```
#define PI_CNTR_MACRO_NOT_FOUND 20
```
Macro not found

```
#define PI_CNTR_MACRO_RECORD  19
```
Error while recording macro

```
#define PI_CNTR_MISSING_PARAM  26
```
Missing parameter

```
#define PI_CNTR_MOVE_TO_LIMIT_SWITCH_FAILED  49
```
Move to limit switch failed

```
#define PI_CNTR_MOVE_WITHOUT_REF_OR_NO_SERVO  5
```
Unallowable move attempted on unreferenced
axis, or move attempted with servo off

```
#define PI_CNTR_NO_AXIS  200
```
No stage connected

```
#define PI_CNTR_NO_AXIS_PARAM_BACKUP  203
```
Backup file with axis parameter not found.

```
#define PI_CNTR_NO_AXIS_PARAM_FILE  201
```
File with axis parameter not found.

```
#define PI_CNTR_NO_DIGITAL_INPUT  35
```
No digital input installed

```
#define PI_CNTR_NO_DIGITAL_OUTPUT  36
```
No digital output installed

```
#define PI_CNTR_NO_ERROR  0
```
No error

```
#define PI_CNTR_NO_JOYSTICK_CONNECTED  40
```
No joystick connected to C-848 controller

```
#define PI_CNTR_NO_JUMP  29
```
No more step response values

```
#define PI_CNTR_NO_MACRO_RECORDING  1002
```
No macro recording

```
#define PI_CNTR_NO_MANUAL_PAD  28
```
No manual pad connected

```
#define PI_CNTR_NO_RELAY_CARD 33
```
No relay card installed

```
#define PI_CNTR_PARAM_NR 24
```
Incorrect number of parameters

```
#define PI_CNTR_PARAM_OUT_OF_RANGE 17
```
Parameter out of range

```
#define PI_CNTR_PARAM_SYNTAX 1
```
Parameter syntax error

```
#define PI_CNTR_POS_OUT_OF_LIMITS 7
```
Position out of limits

```
#define PI_CNTR_RECEIVING_BUFFER_OVERFLOW 304
```
Received command is too long

```
#define PI_CNTR_RECEIVING_TIMEOUT 307
```
Timeout while receiving command

```
#define PI_CNTR_REF_WITH_REF_DISABLED 50
```
Reference attempted on an axis with referencing disabled

```
#define PI_CNTR_REFERENCING_FAILED 45
```
Referencing failed

```
#define PI_CNTR_RESERVED_1004 1004
```
PI internal error code 1004

```
#define PI_CNTR_RESERVED_204 204
```
PI internal error code 204

```
#define PI_CNTR_SECTOR_ERASE_FAILED 4000
```
Sektor Erase failed

```
#define PI_CNTR_SENDING_BUFFER_OVERFLOW 301
```
Sending Buffer Overflow

```
#define PI_CNTR_SET_PIVOT_NOT_POSSIBLE 9
```
Attempt to set pivot point while U,V or W is not equal 0

```
#define PI_CNTR_SLAVE_POSITION_OUT_OF_RANGE  42
```

Position of slave axis is out of range

```
#define PI_CNTR_SMO_WITH_SERVO_ON  205
```

SMO with servo on

```
#define PI_CNTR_SOFT_LIMIT_OUT_OF_RANGE  27
```

Soft limit out of range

```
#define PI_CNTR_SST_OR_SCAN_RANGE  11
```

Parameter for `SST` or for one of the embedded scan algorithms out of range

```
#define PI_CNTR_STAGE_HAS_NO_LIM_SWITCH  32
```

Axis has no limit switch

```
#define PI_CNTR_STOP  10
```

Controller was stopped

```
#define PI_CNTR_TOO_MANY_NESTED_MACROS  1000
```

Too many nested macros

```
#define PI_CNTR_UNKNOWN_COMMAND  2
```

Unknown command

```
#define PI_CNTR_UUDECODE_ILLEGAL_FORMAT  208
```

uudecode : illegal UUE format

```
#define PI_CNTR_UUDECODE_INCOMPLETE_HEADER  206
```

uudecode : incomplete header

```
#define PI_CNTR_UUDECODE_NOTHING_TO_DECODE  207
```

uudecode : nothing to decode

```
#define PI_CNTR_VEL_OUT_OF_LIMITS  8
```

Velocity out of limits

```
#define PI_CNTR_VOLTAGE_OUT_OF_LIMITS  302
```

Voltage out of limits

```
#define PI_CNTR_VOLTAGE_SET_WHEN_SERVO_ON  303
```

Attempt to set voltage when servo on

```
#define PI_COM_ARRAY_ERROR  (COM_MAX_ERROR - 18)
```

Could not access array data in COM server

```
#define PI_COM_ARRAY_RANGE_ERROR  (COM_MAX_ERROR - 19)
```

Range of array does not fit the number of parameters

```
#define PI_CONTROLLER_BUSY  (COM_MAX_ERROR -  8)
```

Controller is busy with some lengthy operation (e.g. reference movement, fast scan algorithm)

```
#define PI_DLL_NOT_LOADED  (COM_MAX_ERROR - 34)
```

DLL neccessary to call function not loaded, or function not found in DLL

```
#define PI_FILE_TRANSFER_BEGIN_MISSING  (COM_MAX_ERROR - 50)
```

A uuencoded file transfered did not start with "begin" and the proper filename

```
#define PI_FILE_TRANSFER_CRC_ERROR  (COM_MAX_ERROR - 52)
```

Checksum error when transfering a file to/from the controller

```
#define PI_FILE_TRANSFER_ERROR_TEMP_FILE  (COM_MAX_ERROR - 51)
```

Could not create/read file on host PC

```
#define PI_IN_MACRO_MODE  (COM_MAX_ERROR - 10)
```

Controller is (already) in macro mode - command not valid in macro mode

```
#define PI_INVALID_ARGUMENT  (COM_MAX_ERROR - 15)
```

One of the arguments given to the function is invalid (empty string, index out of range, ...)

```
#define PI_INVALID_AXIS_IDENTIFIER  (COM_MAX_ERROR - 17)
```

Invalid axis identifier

```
#define PI_INVALID_CONTROLLER_VERSION  (COM_MAX_ERROR - 40)
```

The Version of the connected controller is invalid.

```
#define PI_INVALID_MANUAL_PAD_AXIS  (COM_MAX_ERROR -  7)
```

Axis not currently controlled by a manual control pad

```
#define PI_INVALID_MANUAL_PAD_KNOB  (COM_MAX_ERROR -  6)
```

Invalid number for manual control pad knob

```
#define PI_INVALID_SGA  (COM_MAX_ERROR -  3)
```

Invalid value for `SGA` - must be one of {1, 10, 100, 1000}

#define PI_INVALID_SPA_CMD_ID (COM_MAX_ERROR - 20)

Command ID given to `SPA` or `SPA`? is not valid

#define PI_INVALID_SPP_CMD_ID (COM_MAX_ERROR - 48)

Command ID given to `SPP` or `SPP`? is not valid

#define PI_MACRO_EDITOR_ERROR (COM_MAX_ERROR - 14)

Internal error in macro editor

#define PI_MACRO_FILE_ERROR (COM_MAX_ERROR - 12)

Could not open file to write macro or to read macro

#define PI_MOTION_ERROR (COM_MAX_ERROR - 24)

motion error while stage was moving

#define PI_NO_MACRO_OR_EMPTY (COM_MAX_ERROR - 13)

No macro with given name on controller or macro is empty

#define PI_NO_MANUAL_PAD (COM_MAX_ERROR - 5)

No manual control pad installed, calls to `SMA` and related commands are not allowed

#define PI_NO_PARAMETER_FILE_OPENED (COM_MAX_ERROR - 36)

There is no parameter file opened

#define PI_NO_WAVE_FOR_AXIS_DEFINED (COM_MAX_ERROR - 44)

There is no wave for the given axis defined

#define PI_NOT_IN_MACRO_MODE (COM_MAX_ERROR - 11)

Controller not in macro mode - command not valid unless macro mode active

#define PI_NR_AVG_OUT_OF_RANGE (COM_MAX_ERROR - 21)

Number for `AVG` out of range - must be >0

#define PI_NR_NAV_OUT_OF_RANGE (COM_MAX_ERROR - 2)

Number for `NAV` out of range - must be in [1,10000]

#define PI_NUMBER_OF_POSSIBLE_GENERATORS_EXCEEDED (COM_MAX_ERROR - 43)

The Number of the possible waves generators has exceeded

#define PI_NUMBER_OF_POSSIBLE_WAVES_EXCEEDED (COM_MAX_ERROR - 42)

The Number of the possible waves has exceeded

```
#define PI_PARAM_SET_ERROR  (COM_MAX_ERROR - 41)
```

parameter could not be set with SPA, parameter on controller undefined!

```
#define PI_PARAMETER_FILE_ALLREADY_OPENED  (COM_MAX_ERROR - 38)
```

There is already a parameter file opened. Please close this file before openig a new file

```
#define PI_PARAMETER_FILE_OPEN_ERROR  (COM_MAX_ERROR - 39)
```

DLL necessary to call function not loaded, or function not found in DLL

```
#define PI_PARAMETER_FILE_PROTECTED  (COM_MAX_ERROR - 35)
```

The opened parameter file is protected and cannot be edited

```
#define PI_PZT_CONFIG_FAILED  (COM_MAX_ERROR - 26)
```

Configuration of PZT stage or amplifier failed.

```
#define PI_PZT_CONFIG_INVALID_PARAMS  (COM_MAX_ERROR - 27)
```

Current settings are not valid for desired configuration.

```
#define PI_REFERENCE_ERROR  (COM_MAX_ERROR - 46)
```

Not all axes could be referenced

```
#define PI_REQUIRED_WAVE_MACRO_NOT_FOUND  (COM_MAX_ERROR - 47)
```

Could not find parameter set, required by frequency relation.

```
#define PI_RUNNING_MACRO  (COM_MAX_ERROR - 25)
```

Controller is (already) running a macro

```
#define PI_STAGE_DOES_NOT_EXIST  (COM_MAX_ERROR - 37)
```

The selected stages does not exist

```
#define PI_STAGE_NAME_ISNT_UNIQUE  (COM_MAX_ERROR - 49)
```

A stage name given to `CST` isn't unique

```
#define PI_THREAD_ERROR  (COM_MAX_ERROR -  9)
```

Internal error - could not start thread

```
#define PI_UNEXPECTED_RESPONSE  (COM_MAX_ERROR -  4)
```

Controller has sent unexpected response

```
#define PI_UNKNOWN_AXIS_IDENTIFIER  (COM_MAX_ERROR -  1)
```

Unknown axis identifier

```
#define PI_UNKNOWN_CHANNEL_IDENTIFIER  (COM_MAX_ERROR - 28)
```

Unknown channel identifier

```
#define PI_UNKNOWN_WAVE_MACRO  (COM_MAX_ERROR - 30)
```

Could not find description of wave form. Maybe WG.INI is missing?

```
#define PI_USER_CANCELLED  (COM_MAX_ERROR - 32)
```

The user cancelled a dialog

```
#define PI_WAV_FAILED  (COM_MAX_ERROR - 23)
```

Generation of wave failed

```
#define PI_WAV_SAMPLES_OUT_OF_RANGE  (COM_MAX_ERROR - 22)
```

Number of samples given to `WAV` out of range

```
#define PI_WAVE_MACRO_FUNC_NOT_LOADED  (COM_MAX_ERROR - 31)
```

The WGMacro DLL function was not found at startup

```
#define PI_WAVE_PARAM_FILE_ERROR  (COM_MAX_ERROR - 29)
```

Error while reading/writing to wave generator parameter file.

```
#define REC_ERROR  -3
```

Error while receiving data

```
#define SEND_ERROR  -2
```

Error while sending data

11. Index