



SM 83E

Mercury DLL
Function Library for Windows

V 2.10

***Function Reference
&
Programming Instructions***

This document is valid for the Products:

C-860.xx Mercury Controller
C-862.xx Mercury2 Controller

Release: 2.10
Release Date: 2001-12-02

© **PI** (Physik Instrumente) **GmbH & Co.**
76228 Karlsruhe, Germany FAX: (+49)721-4846-100

E**M**ail:info@pi.ws

www.pi.ws

Table of Contents :

0.	Manufacturer Declarations	2
0.1.1.	Disclaimers	2
1.	Library for Windows 95 / 98 / 2000 / NT	3
1.1.1.	For Delphi Programmers	4
1.1.2.	For VC++ Programmers	6
2.	General Programming Notes	7
3.	Function Reference	8
	int MCRS_open (BYTE PortNumber, int baudrate)	8
	int MCRS_close (void)	8
	int MCRS_setBuffer (void)	8
	WORD MCRS_EOF (void)	9
	int MCRS_clear_input (void)	9
	int MCRS_sendChar (char *character)	9
	int MCRS_sendString (char *command_string)	9
	int MCRS_moveA (int axis, int position)	10
	int MCRS_moveR (int axis, int shift)	10
	int MCRS_getChar (char *character)	10
	int MCRS_getString (char *report, WORD count)	11
	int MCRS_getStringCR (char *report)	11
	int MCRS_getMacro (int macrono, char *report)	11
	int MCRS_getPos (void)	12
	int MCRS_getPosErr (void)	12
	int MCRS_getVal (int query)	12
	int MCRS_getQuery (char *command, char *report)	13
	int MCRS_moving (void)	13
	int MCRS_waitStop (void)	13
	int MCRS_getSTB (int bytenumber)	13
	int MCRS_initNetwork (int maxAxis)	14
	int MCRS_select (int newAxis)	14
	int MCRS_getDLLversion (void)	15

0. Manufacturer Declarations

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

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

0.1.1. Disclaimers

The software described in this manual is distributed as it is.

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

© Copyright 2001 by Physik Instrumente GmbH

Release: 2.10

File:SM83E210.doc, 84480 Bytes

PDate: 03.12.01 09:54

1. Library for Windows 95 / 98 / 2000 / NT

MercLib.DLL

Version 2.10, 2001-12-02

Function Contents of MerLib210.DLL :

<u>Ordinal</u>	<u>Entry Point</u>	<u>Name</u>
0037	00000000	MCRS_EOF
0039	0001f068	MCRS_clear_input
0033	00000000	MCRS_close
0001	0001f0b4	MCRS_getChar
003b	00000000	MCRS_getDLLversion
0023	0001f128	MCRS_getMacro
000d	00000000	MCRS_getPos
000f	0001f1fc	MCRS_getPosErr
0013	00000000	MCRS_getQuery
001b	0001f2f8	MCRS_getSTB
0003	00000000	MCRS_getString
0007	0001f344	MCRS_getStringCR
0005	00000000	MCRS_getStringTerm
0011	0001f45c	MCRS_getVal
0017	00000000	MCRS_initNetwork
001f	0001f510	MCRS_moveA
0021	00000000	MCRS_moveR
0019	0001f5b4	MCRS_moving
0031	00000000	MCRS_open
0015	0001f3bc	MCRS_select
0009	00000000	MCRS_sendChar
000b	0001f8f4	MCRS_sendString
0035	00000000	MCRS_setBuffer
0025	0001f984	MCRS_set_device
001d	00000000	MCRS_waitStop

This 32 bit Windows Library allows to communicate with the Mercury Controller via RS-232 Com ports 1 through 6.

The baud rate used is 9600.

The name of the DLL is : MercLib210.DLL, dated 2001-12-02

1.1.1. For Delphi Programmers*Declaration of functions: (Pascal style)*

```

//*****
// File name : MercLib_def.PAS
// IDE       : Delphi32
// Purpose   : Type declarations for MercLib.DLL Mercury DLL
//
// Date      : 2001-12-02
//*****

const
  ExtLib = 'MercLib210.DLL';

  LF = #10;
  CR = #13;

  // Error Base Codes
  EBC_init = 16;
  EBC_setBuffer = 32;
  EBC_EOF = 48;
  EBC_getChar = 64;
  EBC_getstring = 80;
  EBC_sendChar = 96;
  EBC_sendstring = 112;
  EBC_sendstringE = 128;

  // Error codes
  ERR_readfile = 1;
  ERR_writefile = 2;
  ERR_timeout = 3;
  ERR_length = 4;
  ERR_content = 5;
  ERR_GetCommState = 6;
  ERR_SetCommState = 7;
  ERR_PurgeComm = 8;
  ERR_PortNumber = 9;
  ERR_hnadle = 10;

//-----
// Function Declarations :
//-----
// Case sensitive writing required !!

function MCRS_open(portnumber:byte; baudrate:integer):integer;
  stdcall external ExtLib; //index 50

function MCRS_close:integer;
  stdcall external ExtLib; // index 52

function MCRS_setBuffer:integer;
  stdcall external ExtLib; // index 54

function MCRS_sendString(pCmd:pChar):integer;
  stdcall external ExtLib; // index 12

function MCRS_moveA(axis,position:integer):integer;
  stdcall external ExtLib; // index 32

function MCRS_moveR(axis,shift:integer):integer;
  stdcall external ExtLib; // index 34

function MCRS_getPos:integer;
  stdcall external ExtLib; // index 14

function MCRS_getPosErr:integer;
  stdcall external ExtLib; // index 16

function MCRS_getVal(query:integer):integer;

```

```
        stdcall external ExtLib; // index 18

function MCRS_getQuery(pCmd,psRead:PChar):integer;
        stdcall external ExtLib; // index 20

function MCRS_getStringCR(psRead:PChar):integer;
        stdcall external ExtLib; // index 8

function MCRS_getString(psRead:PChar;wlang:word):integer;
        stdcall external ExtLib; // index 4

function MCRS_getMacro(macrono:integer;pMacro:PChar):integer;
        stdcall external ExtLib; // index 36

function MCRS_moving:integer;
        stdcall external ExtLib; // index 26

function MCRS_initNetwork(maxAxis:integer):integer;
        stdcall external ExtLib; // index 24

function MCRS_select(newAxis:integer):integer;
        stdcall external ExtLib; // index 22

function MCRS_set_device(newAxis:integer):integer;
        stdcall external ExtLib; // index 38

function MCRS_clear_input:integer;
        stdcall external ExtLib; // index 58

function MCRS_EOF:integer;
        stdcall external ExtLib; // index 56

function MCRS_getSTB(byteno:integer):integer;
        stdcall external ExtLib; // index 28

function MCRS_waitStop:integer;
        stdcall external ExtLib; // index 30

function MCRS_getDLLversion:integer;
        stdcall external ExtLib; // index 60
```

1.1.2. For VC++ Programmers

Declaration of error codes and functions: (C style)

```
//-----  
// Error Base Codes  
//-----  
#define EBC_init 16;  
#define EBC_setBuffer 32;  
#define EBC_EOF 48;  
#define EBC_getChar 64;  
#define EBC_getstring 80;  
#define EBC_sendChar 96;  
#define EBC_sendstring 112;  
#define EBC_sendstringE 128;  
  
//-----  
// Error Offset Codes  
//-----  
#define ERR_readfile 1;  
#define ERR_writefile 2;  
#define ERR_timeout 3;  
#define ERR_length 4;  
#define ERR_content 5;  
#define ERR_GetCommState 6;  
#define ERR_SetCommState 7;  
#define ERR_PurgeComm 8;  
#define ERR_PortNumber 9;  
#define ERR_handle 10;  
  
//-----  
// Functions C convention  
//-----  
int __stdcall MCRS_open(BYTE PortNumber, int baudrate);  
int __stdcall MCRS_close(void);  
int __stdcall MCRS_setBuffer(void);  
WORD __stdcall MCRS_EOF(void);  
int __stdcall MCRS_clear_input(void);  
int __stdcall MCRS_sendChar(char *character);  
int __stdcall MCRS_sendString(char *command_string);  
int __stdcall MCRS_moveA(int axis, int position);  
int __stdcall MCRS_moveR(int axis, int shift);  
int __stdcall MCRS_getChar(char *character);  
int __stdcall MCRS_getString(char *report, WORD count);  
int __stdcall MCRS_getStringCR(char *report);  
int __stdcall MCRS_getMacro(int macno, char *report);  
int __stdcall MCRS_getPos(void);  
int __stdcall MCRS_getPosErr(void);  
int __stdcall MCRS_getVal(int query);  
int __stdcall MCRS_getQuery(char *command, char *report);  
int __stdcall MCRS_moving(void);  
int __stdcall MCRS_waitStop(void);  
int __stdcall MCRS_getSTB(int bytenumber);  
int __stdcall MCRS_getDLLversion(void);  
int __stdcall MCRS_initNetwork(int maxAxis);  
int __stdcall MCRS_select(int newAxis);  
int __stdcall MCRS_set_device(int newAxis);
```

2. General Programming Notes

2.1. Example 1

Assuming you have one Mercury set to Address 0 (this is device #1) connected to COM-port 2. Your application may start as follows:

```
nErr = MCRS_open(2,9600) // when starting the application, open the COM port
nErr = MCRS_set_device(1) // activate device #1 ( Mercury with address set to 0)
nErr = MCRS_sendString("DP120") // set p-term parameter to 120
```

add here all other p-i-d servo parameter settings

```
nErr = MCRS_sendString("SV56000") // set velocity to 56000 counts/s
nErr = MCRS_sendString("SA450000") // set acceleration to 450,000 counts/s
nErr = MCRS_moveR(1,50000) // move Mercury #1 for 50,000 counts
```

add here all further commands

```
MCRS_close // before terminating the application close the COM port
```

2.2. Example 2

Assuming you are using COM port #1 and you have three Mercurys connected: one at address 0 (this is device #1), another at address 3 (this is device #4) and a third one at address 4 (this is device #5). Note that the functions request device numbers (running from 1 to 16) and not the physical address (running from 0 to 15) for axis specification.

Your application may start as follows:

```
nErr = MCRS_open(1,9600) // when starting the application, open the COM port
nErr = MCRS_initNetwork(5) // scan all network devices starting with device #5 down to
                           device #1.
```

```
nErr = MCRS_select(1) // select device #1
nErr = MCRS_sendString("DP120") // set p-term
nErr = MCRS_sendString("DI15") // set i-term
nErr = MCRS_sendString("DD300") // set d-term
nErr = MCRS_sendString("SV56000") // set velocity to 56000 counts/s
nErr = MCRS_sendString("SA450000") // set acceleration to 450,000 counts/s
```

```
nErr = MCRS_select(4) // select device #4
nErr = MCRS_sendString("DP120") // set p-term
nErr = MCRS_sendString("DI15") // set i-term
nErr = MCRS_sendString("DD300") // set d-term
nErr = MCRS_sendString("SV56000") // set velocity to 56000 counts/s
nErr = MCRS_sendString("SA450000") // set acceleration to 450,000 counts/s
```

```
nErr = MCRS_select(5) // select device #1
nErr = MCRS_sendString("DP120") // set p-term
nErr = MCRS_sendString("DI15") // set i-term
nErr = MCRS_sendString("DD300") // set d-term
```

```
nErr = MCRS_sendString("SV56000") // set velocity to 56000 counts/s
nErr = MCRS_sendString("SA450000") // set acceleration to 450,000 counts/s
```

```
nErr = MCRS_moveR(1,50000) // move Mercury #1
nErr = MCRS_moveR(4,-150000) // move Mercury #4
nErr = MCRS_moveR(5,-400000) // move Mercury #5
```

add here further commands of your program

MCRS_close // before terminating the application close the COM port

3. Function Reference

Handling COM port

int **MCRS_open**(BYTE PortNumber, int baudrate)

This function initializes the COM port according to the Mercury requirements. PortNumber represents the number of the COM port and ranges from 1 to 6.

Ordinal index: 50

Input: port number (byte), range 1...6
baudrate (integer), valid rates: 9600 and 4800
(use baudrate 9600)

Return: Errorcode
0 = no error
20 = timeout error
25 = wrong port number
26 = handle error
22 = get comm state error
23 = set comm state error

int **MCRS_close**(void)

This function disables the com port that was opened by MCRS_open().

Ordinal index: 52

Input: void

Return: Errorcode
0 = no error
1 = function failed

int **MCRS_setBuffer**(void)

This function defines input and output buffers for communication in cases where OS default settings are not enough. Most applications do not need that function.

Ordinal index: 54

Input: void

Return: Errorcode
0 = no error
32 = function failed

WORD MCRS_EOF(void)

This function returns the number of bytes available in the communication input buffer. If This function returns 0, there is nothing to read.

Ordinal index: 56

Input: void

Return: Number of bytes in the input buffer.

int MCRS_clear_input(void)

This function clears all bytes in the input buffer.

Ordinal index: 58

Input: void

Return: errorcode
0 = no error
1 = function failed

Send to Mercury

int MCRS_sendChar(char *character)

This function transfers one character to Mercury.

Ordinal index: 10

Input: pointer to character

Return: Errorcode:
0 = no error
98 = write error
100 = length error

int MCRS_sendString(char *command_string)

This function transfers a command string to the Mercury. The command string may contain a single command or any compound command up to a length of 240 characters.

Ordinal index: 12
Input: pointer to command string

Return: Errorcode:
0 = no error
114 = write error
116 = length error

int **MCRS_moveA**(int axis, int position)

This function moves one motor to the indicated position.

Ordinal index: 32
Input: axis : integer
position : integer

Return: Error code:
0 = no error
1 = error: wrong axis number
2 = error: axis not connected
3 = error: send string

int **MCRS_moveR**(int axis, int shift)

This function moves one motor relative to the current position by shift counts.

Ordinal index: 34
Input: axis : integer
shift : integer

Return: Error code:
0 = no error
1 = error: wrong axis number
2 = error: axis not connected
3 = error: send string

Get from Mercury

int **MCRS_getChar**(char *character)

This function reads one character from the com port input buffer.

Ordinal index: 2
Input: pointer to character

Return: Errorcodes:
 0 = no error
 65 = read file failed
 67 = timeout error

int **MCRS_getString**(char *report, WORD count)

This function reads count characters from the input buffer. This function can be used if the number of characters available is known. If count is larger than the number of characters in the buffer, a timeout error occurs.

Ordinal index: 4
Input: *report* : pointer to character buffer
 count : number of characters to be read

Return: Errorcodes:
 0 = no error
 81 = read file failed
 84 = not enough characters received, timeout error

int **MCRS_getStringCR**(char *report)

This function reads a string of characters until a carriage return is received. This allows to retrieve a complete report command from the Mercury independent of its length.

Ordinal index: 8
Input: *report* : pointer to character buffer

Return: Errorcodes:
 0 = no error
 65 = read file failed
 67 = timeout error
 255 = got nothing, may be timeout

int **MCRS_getMacro**(int macrono, char *report)

This function reads one macro string.

Ordinal index: 36
Input: *macno* : number of macro to be read
 report: macro content

Return: Errorcodes:
 0 = no error
 1 = no macro available at this index number

int MCRS_getPos(void)

This function reads the current motor position in encoder counts and delivers it as an integer value. This function can be called any time, also on the fly.

The returned value is also used as error indicator.

Ordinal index: 14

Input: *void*

Return: current position value or errorcode:
2.147.483.647 = content error
2.147.483.646 = getString error
2.147.483.645 = sendString error
2.147.483.644 = conversion error

int MCRS_getPosErr(void)

This function reads the current motor position error (difference of motor position to profile position) in encoder counts and delivers it as an integer value. This function can be called any time, also on the fly.

The returned value is also used as error indicator.

Ordinal index: 16

Input: *void*

Return: current position error value or errorcode:
2.147.483.647 = content error
2.147.483.646 = getString error
2.147.483.645 = sendString error

int MCRS_getVal(int query)

This function reads the current value of one of the requested parameters as integer. This function can be called any time, also on the fly.

The returned value is also used as error indicator.

Ordinal index: 18

Input: *query*: id number of parameter requested:
1 = TP : current motor position
2 = TT : target position
3 = TF : profile following error
4 = TE : distance to target
5 = TY : programmed velocity
6 = TL : programmed acceleration
7 = GP : programmed p-term
8 = GI : programmed i-term
9 = GD : programmed d-term
10 = GL : programmed integration limit

Return: current position error value or errorcode:
2.147.483.647 = content error

2.147.483.646 = getString error
2.147.483.645 = sendString error

int MCRS_getQuery(char *command, char *report)

This function reads the requested string. The command string must be a tell-command, otherwise an timeout occurs.

Ordinal index: 20

Input: *command*: pointer to command string
 report: character array

Return: errorcode:
 0 : no error
 > 0 : error during execution

Mercury Status Reporting

int MCRS_moving(void)

This function returns a value indicating whether the motor is moving or not.

Ordinal index: 26

Input: void

Return: statuscode :
 0 = not moving
 1 = moving
 -1 = error : content
 -2 = error: query

int MCRS_waitStop(void)

This function waits until the motor trajectory has been completed.

Ordinal index: 30

Input: void

Return: errorcode :
 0 = no error
 1 = error during execution

int MCRS_getSTB(int bytenumber)

This function reads the requested status byte. There are 6 status bytes available. For meaning of the bits see MS67E operating manual.

Ordinal index: 28

Input: *bytenumber*: identification of byte to be read, range 1..6

Return: *statuscode*:
positive number: statusbyte as integer
-1 = error content
-2 = error query
-3 = error: parameter out of range

Mercury Network

int **MCRS_initNetwork**(int maxAxis)

This function is used to initialize a Mercury network. even if only one Mercury is connected, this function has to be called.

This function assigns logical addresses and allows to access individual controllers by the MCRS_select function.

The function scans all addresses, starting with the highest address maxAxis down to address 1 and registers all controllers found. If you do not know which addresses the controllers are set to, call the function with maxAxis = 16 to find all addresses possible (valid addresses range from 1 to 16, this differs from the physical address setting that goes from 0 to 15)

The function takes some 400 ms for each address to be checked. So if you have only one Mercury at address 0 (physical) connected, then call the function with maxAxis = 1. The maxAxis parameter shall have the value of the highest Mercury address in the network (this is physical address setting + 1).

Ordinal index: 24

Input: maxAxis, range 1 to 16

Return: *statuscode*:
0 = no controller found
positive number = binary number of addressed controllers
negative number = error code

Note: This function takes about (maxAxis * 0.4 seconds) to finish

int **MCRS_select**(int newAxis)

This function activates one of the Mercury controllers in a daisy chain network. The selected controller is the only one who can be talked to and who can talk.

Each controller has its individual address setting from 0 to 3 (physically). For more convenience, the range of the `newAxis` variable ranges from 1 to 16.

Ordinal index: 22

Input: *newAxis*: axis to be selected, range 1 to 16

Return: errorcode:
0 = no error
1 = axis not in range
2 = axis not connected

int **MCRS_set_device**(int newAxis)

This function sends the address code for the specified device. The function does not check whether the axis is registered by `initNetwork`. This function can be used to re-address a device that was temporary disconnected or accidentally reset. Also when using just one Mercury at the port, then set this to communication by this command.

Ordinal index: 38

Input: *newAxis*: axis to be selected, range 1 to 16

Return: errorcode:
0 = no error

int **MCRS_getDLLversion**(void)

This function returns an integer that represents the version of the DLL used. Version 2.10 delivers the value 210.

Ordinal index: 60

Input: void

Return: version number

