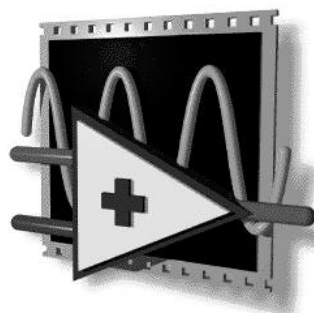


PI**Software Application Manual****SM 84E**

LabView™ Virtual Instruments
for
Mercury
DC-Motor Controllers

**National Instruments**
LabVIEW™ 5.1

This software is sold under licensing
agreement for use on only one machine at
a time.



© Copyright 1999. All rights reserved.

SID#	084
Name:	MCRS_functionname.vi
Release:	2.00
Release Date:	2001-01-18

© 2001 by Physik Instrumente (PI) GmbH

Table of Contents :

1.	Introduction.....	3
2.	Alphabetic List of Mercury vis.....	4
3.	Virtual Instrument Reference.....	5
3.1.1.	MCRS_open.vi.....	5
3.1.2.	MCRS_close.vi	6
3.1.3.	MCRS_sendString.vi.....	7
3.1.4.	MCRS_moveA.vi.....	8
3.1.5.	MCRS_moveR.vi	9
3.1.6.	MCRS_moving.vi	10
3.1.7.	MCRS_getString.vi	11
3.1.8.	MCRS_getPos.vi.....	12
3.1.9.	MCRS_getPosErr.vi.....	13
3.1.10.	MCRS_getVal.vi.....	14
3.1.11.	MCRS_getSTB.vi.....	15
3.1.12.	MCRS_initNetwork.vi	16
3.1.13.	MCRS_select.vi	17

This vi collection requires the Windows library "MercLV.DLL".

© 2001 by Physik Instrumente PI GmbH
Polytec Platz 1-7, D-76337 Waldbronn
Email to L.dietrich @physikinstrumente.com
FAX: (+49) 7243-604-145

Document: SM84E200.doc
Version: 2.00
PDate: 19.01.01 11:48

1. Introduction

Mercury is a DC-Motor controller for one channel (order # C-860.00). Up to 4 Mercurys can be networked and operated from one RS-232 interface (COM port). used

Mercury DC-Motor Controllers are precision motion controllers for DC-motors using incremental encoder feedback. The device offers RS-232 interfaces and can be operated by ASCII strings.

Programming details and command formats are described in the MS67E operating manual.

The LabView driver library for the Mercury is provided to help the customer to generate his own LabView program. The LabView programmer may use general purpose Vis for communication or command specific Vis for executing the desired function.

Most Mercury commands can be accessed via specific VIs that can be implemented in your LabView program. Connecting the Vi-icons with the desired parameter values and function attributes, the entire Mercury functionality can be implemented easily into your LV applications.

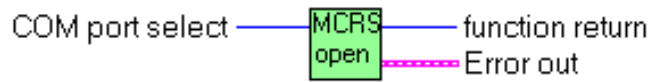
2. Alphabetic List of Mercury vis

Release date of vis: 2000-12-02 (preliminary evaluation release, not yet complete)

vi Name	Function	
Serial Communication initialization		
MCRS_open.vi	opens and configures a COM port for Mercury specific data transfer	
MCRS_close.vi	close COM port	
MCRS_clear_input.vi	erases the communication input buffer	
Mercury initialization		
MCRS_initNetwork.vi	initializes Mercury network	
MCRS_select.vi	selects one member of the network	
Sending Commands		
MCRS_sendString.vi	sends a command string	
MCRS_moveA.vi	move to an absolute position	
MCRS_moveR.vi	move for a relative increment	
MCRS_moving?.vi	reads the moving status	
Receiving data		
MCRS_getString.vi	requests a string from input buffer	
MCRS_getPosvi	reads the current motor position	
MCRS_getPosErr.vi	reads the current position error	
MCRS_getVal.vi	reads the current value	
MCRS_STB.vi	reads one status byte	

3. Virtual Instrument Reference

3.1.1. MCRS_open.vi



MCRS_open.vi

This vi opens the RS-232 port for Mercury communication

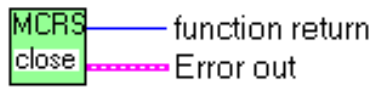
Purpose: This vi opens and configures the specified COM port for communication with the Mercury controller. Before any other vi is executed, this vi has to be called.

Inputs : *Com port select*
COM port number 1..4

Outputs : *function return:*
error code indicating what happened during execution:
0 = no error
20 = timeout error
25 = wrong port number
26 = handle error
22 = get comm state error
23 = set comm state error

Error out:
Standard error cluster

3.1.2. MCRS_close.vi



MCRS_close.vi

This vi closes the COM port for Mercury communications

Purpose: This vi closes the open Com port. It has to be executed before the port can be opened again, normally it is called prior execution of the program.

Inputs : none

Outputs : *function return:*
error code indicating what happened during execution:
0 = no error
1 = handle error

Error out:
Standard error cluster

3.1.3. MCRS_sendString.vi



MCRS_sendString.vi

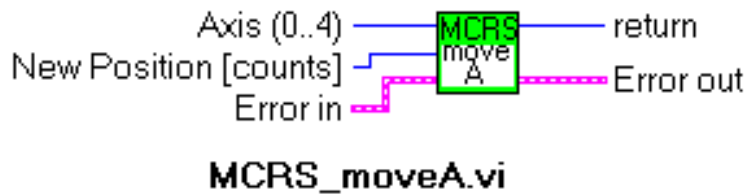
This vi sends a command string

Purpose: This vi transfers one command string to the Mercury, e.g. "MR5000". When the string is sent, the command is executed immediately. Command causing the Mercury to send back a report message, like "TT" Tell Target, require to read the response by MCRS_getString.vi.

Inputs : *Command String:*
ASCII string
Error in:
standard error cluster

Outputs : *function return:*
error code indicating what happened during execution:
0 = no error
114 = write error
116 = string length error
Error out:
Standard error cluster

3.1.4. MCRS_moveA.vi



This vi moves one axis to a new absolute position

Purpose: This vi sends a move command that moves the motor to a new absolute count position.

Inputs : *Axis number :* If the axis number is greater 0, then the controller is a node in a Mercury network and is addressed prior the move command. If the axis number is 0, no addressing is performed as it is in the case of a single controller application.

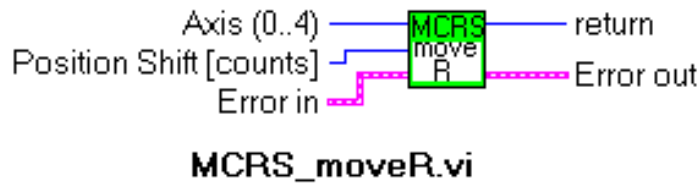
New Position: New motor position in encoder counts.

Error in:
standard error cluster

Outputs : *function return:*
error code indicating what happened during execution:
0 = no error
1 = wrong axis number
2 = axis not connected
3 = error during send operation

Error out:
Standard error cluster

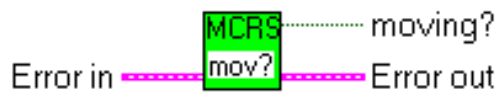
3.1.5. MCRS_moveR.vi



This vi moves one axis for a relative increment

- Purpose: This vi sends a move command that moves the motor for a relative number of counts.
- Inputs :
- Axis number :* If the axis number is greater 0, then the controller is a node in a Mercury network and is addressed prior the move command. If the axis number is 0, no addressing is performed as it is in the case of a single controller application.
- Position Shift:* Increment to move.
- Error in:*
standard error cluster
- Outputs :
- function return:*
error code indicating what happened during execution:
 0 = no error
 1 = wrong axis number
 2 = axis not connected
 3 = error during send operation
- Error out:*
Standard error cluster

3.1.6. MCRS_moving.vi



MCRS_moving.vi

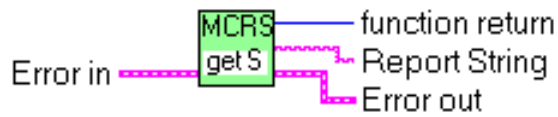
This vi delivers the current moving status

Purpose: This vi returns the moving status of the current active controller.

Inputs : *Error in:*
standard error cluster

Outputs : *moving?:*
boolean value
Error out:
Standard error cluster

3.1.7. MCRS_getString.vi



MCRS_getString.vi

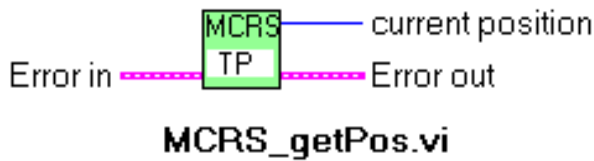
This vi reads a report string

Purpose: This vi reads a string of characters from the input buffer until a CR is received.

Inputs : *Command String:*
ASCII string
Error in:
standard error cluster

Outputs : *function return:*
error code indicating what happened during execution:
0 = no error
114 = write error
116 = string length error
Error out:
Standard error cluster

3.1.8. MCRS_getPos.vi



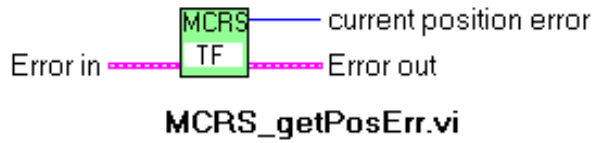
This vi reads the current motor position

Purpose: This vi reads the current motor position in encoder counts and delivers it as an integer value. This vi can read the motor position also on the fly.

Inputs : *Error in:*
standard error cluster

Outputs : *current position:*
current motor position:
The position value is also used for error indication. If the returned value equals the largest positive integer number and next lower numbers, an error has occurred:
2.147.483.647 = content error
2.147.483.646 = getString error
2.147.483.645 = sendString error
2.147.483.644 = conversion error
Error out:
Standard error cluster

3.1.9. MCRS_getPosErr.vi



This vi reads the current error of motor from profile position

- Purpose: This vi reads the current profile position error and delivers an integer value. This vi can read the motor position also on the fly.
- Inputs : *Error in:*
standard error cluster
- Outputs : *current position error:*
current motor profile position error:
The return value is also used for error indication. If the returned value equals the largest positive integer number and next lower numbers, an error has occurred:
2.147.483.647 = content error
2.147.483.646 = getString error
2.147.483.645 = sendString error
- Error out:*
Standard error cluster

3.1.10. MCRS_getVal.vi



MCRS_getVal.vi

This vi reads the requested value

Purpose: This vi reads a numeric value from the controller.

Inputs : *value select:*
defines which value is to be read:

- 1 : TP
- 2 : TT
- 3 : TF
- 4 : TE
- 5 : TY
- 6 : TL
- 7 : GP
- 8 : GI
- 9 : GD
- 10 : GL

Error in:
standard error cluster

Outputs : *current value:*
current value requested.
The return value is also used for error indication. If the returned value equals the largest positive integer number and next lower numbers, an error has occurred:
2.147.483.647 = content error
2.147.483.646 = getString error
2.147.483.645 = sendString error

Error out:
Standard error cluster

3.1.11. MCRS_getSTB.vi



MCRS_getSTB.vi

This vi reads one of the 6 status bytes

Purpose: This vi reads the Mercury status report and outputs one of the six bytes.

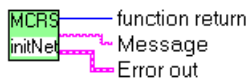
Inputs : *byte number* (1 to 6)
Error in:
 standard error cluster

Outputs : *status byte value:*
Error out:
 Standard error cluster

Byte 1	LM629 status	Bit 0 : busy Bit 1 : command error Bit 2 : Trajectory complete Bit 3 : Index pulse received Bit 4 : Position limit exceeded Bit 5 : Excessive position error Bit 6 : Breakpoint reached Bit 7 : Motor loop OFF
Byte 2	Internal operation flags	Bit 0 : Echo ON Bit 1 : Wait in progress Bit 2 : Command error Bit 3 : Leading zero suppression active Bit 4 : Macro command called Bit 5 : Leading zero suppression disabled Bit 6 : Number mode in effect Bit 7 : Board addressed
Byte 3	Motor loop flags	Bit 0 : Bit 1 : Bit 2 : Move direction polarity Bit 3 : Move error (MF condition occurred in WS) Bit 4 : Bit 5 : Bit 6 : Move error (Excess following error in WS) Bit 7 : Internal LM629 communication in progress
Byte 4	Limit Switch status	Bit 0 : Limit Switch ON Bit 1 : Limit switch active state HIGH Bit 2 : Find edge operation in progress Bit 3 : Brake ON Bit 4 : Bit 5 : Bit 6 : Bit 7 :
Byte 5	Limit switch inputs	Bit 0 : Bit 1 : Reference switch input Bit 2 : Positive limit switch input Bit 3 : Negative limit switch input

	Bit 4 :
	Bit 5 :
	Bit 6 :
	Bit 7 :
Byte 6 Error Codes	(numbers reported):
	01 : command not found
	02 : First command character must be a letter
	05 : Character following command must be a number
	06 : Value too large
	07 : Value too small
	08 : Continuation character must be a comma
	09 : Command buffer overflow
	0A : macro storage overflow

3.1.12. MCRS_initNetwork.vi

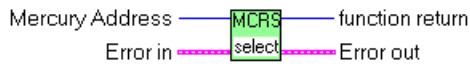


MCRS_initNetwork.vi

This vi has to be called if more than one Mercury is used in a network

<u>Purpose:</u>	This vi has to be executed if more than one Mercury is connected on one Com port. In this case the second Mercury has a different address setting and this vi enables the communication to all controllers networked.
<u>Inputs :</u>	none
<u>Outputs :</u>	<p><i>function return:</i></p> <p>The return value is a binary coded number of all controllers found in the network. The values can be from 0 (no controller found) to 15 (controllers with addresses 0,1,2 and 3 found). If a negative number is returned, an error has occurred.</p> <p>The controller with the lowest address is set active when the vi is left.</p> <p><i>Message:</i></p> <p>information string</p> <p><i>Error out:</i></p> <p>Standard error cluster</p>

3.1.13. MCRS_select.vi



MCRS_select.vi

This vi selects one Mercury for communication. Prior to use this vi, the initNetwork.vi has to be called one time.

Purpose: This vi is required if a Mercury network is operated. Each Mercury has its individual address and this vi activates a specific controller and disables all others.

Inputs : *Mercury Address :*
range 1..4
Note: the internal hardware address setting ranges from 0 to 3.
Error in:
standard error cluster

Outputs : *function return:*
The return value is the error code:
0 = no error
1 = address not in allowed range
2 = address in range, but controller not found
Error out:
Standard error cluster

