Piezo · Nano · Positioning

MS177E Mercury™ MMC-DLL Software Manual

Mercury™-Class Windows DLL

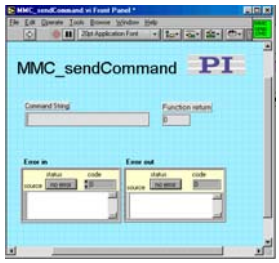
and associated LabVIEW VIs

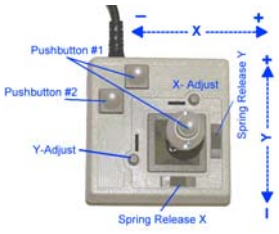
Release: 5.0.0 Date: 2007-12-21

This document is valid for the following product(s):

- **C-862** Mercury™-DC Motor Controller
- **C-863** Mercury™-DC Motor Controller
- **C-663** Mercury™-Step Motor Controller
- **C-170** Redstone PILine® Controller

Function Reference and Programming Instructions





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

Document Number MS177E, Release 5.0.0
Mercury_DLL-LV_MS177E500.doc

About This Document

Users of This Manual

This manual is designed to help the reader to install and operate the Mercury™-Class Windows DLL and associated LabVIEW VIs. It assumes that the reader has a fundamental understanding of basic servo systems, as well as motion control concepts and applicable safety procedures. This document is available as PDF file on the product CD. Updated releases are available for download from www.pi.ws or by email: contact your Physik Instrumente Sales Engineer or write info@pi.ws.

Conventions

The notes and symbols used in this manual have the following meanings:

!

CAUTION

Calls attention to a procedure, practice, or condition which, if not correctly performed or adhered to, could result in damage to equipment.

NOTE

Provides additional information or application hints.

Related Documents

The software tools and any stages which might be delivered with Mercury™-Class Windows DLL and associated LabVIEW VIs, are described in their own manuals. All documents are available as PDF files via download from the PI Website (http://www.pi.ws/).. For updated releases or other versions contact your Physik Instrumente Sales Engineer or write info@pi.ws.

Hardware User Manuals	User Manual for each hardware component
MMCRun MS139E	Mercury Operating Software (native commands)
Mercury Native DLL & LabVIEW MS177E	Windows DLL Library and LabView VIs (native-command-based, this document)
Mercury Native Commands MS176E	Native Mercury™ Commands
Mercury GCSLabVIEW_MS149E	LabView VIs based on PI GCS command set
Mercury GCS DLL_MS154E	Windows DLL Library (GCS commands)
PIMikroMove User Manual SM148E	PIMikroMove® Operating Software (GCS-based)
Mercury Commands MS163E	Mercury™ GCS Commands
PIStageEditor _SM144E	Software for managing GCS stage-data database

Contents

1 Introduction

It is possible to use either the Mercury™ native ASCII command set or the PI General Command Set (GCS) to operate Mercury™ Class controllers. A Windows DLL and LabVIEW drivers are provided based on each of these two command sets. See the list of related documents above and the corresponding manuals for detailed descriptions of the options available.

This document describes both the native-command-based DLL and the corresponding LabVIEW VIs.

1.1 MMC-DLL Features

The MMC-DLL is based on the Mercury™ native command set. It is the Mercury™ native Windows Library (DLL) for all Mercury™ Controllers. As such, it supports all native properties and functions of the Mercury™ controllers as well as the handling of the serial COM port of the PC (USB access is implemented with a driver which presents the USB to the DLL as an “extra” COM port whenever a compatible Mercury™ is connected via USB).

A small test program, MMC_DLLTest.EXE, is provided with the native-command-set software to test the DLL installation and operation.

1.1.1 Compatible Controllers & Firmware

The MMC.DLL Windows library can be used with the following Mercury™ Class controllers having firmware of the recommended version, or newer.

C-862 Mercury™ DC-Controller	Firmware 8.47
C-663 Mercury™ DC Controller	Firmware 1.10
C-663 Mercury™-Step Controller	Firmware 1.10
C-170 Redstone Controller	Firmware 2.20

1.1.2 Version information

Description:	Native Mercury™ DLL
Current version:	4.13
Application file name:	MMC.DLL
Distribution file name:	MMC413.DLL
USB driver file directory:	\Drivers (on the Mercury CD)

1.2 Installation

No formal installation of the MMC DLL or MMC_DLLTest.EXE is needed. Just copy the DLL and EXE files into your project directory or at the place where your development system requires it.

If you wish to use the USB connection, the USB driver must be installed.

NOTE

Windows NT does not have USB support as standard. PI supports only the RS-232 interface for Windows NT. Unless you manage to get USB interfacing operational, the number of networkable devices may be limited to as few as 6 units by the current-sourcing capacity of the PC's COM port output stages.

CAUTION

Never connect the RS-232-IN and USB connectors of the same controller to a PC at the same time, as damage may result.

!

To install the USB driver, simply connect and power up the controller, then follow the instructions that appear when the computer detects the new hardware. Show the Hardware Wizard the \Driver directory on the CD.

NOTE

The USB drivers will make the USB interface appear to all software on the host PC as a new COM port. That port will be present only when the controller is connected via USB and powered-up.

With current firmware, it may be necessary to power-cycle the controller while the host PC is on to establish communication with it.

To use the LabVIEW VIs, the LabVIEW environment from National Instruments Corporation is required. Follow the instructions in the LabVIEW documentation for installing third-party virtual instruments (VIs).

1.3 Networking

Up to 16 Mercury™ controllers of any kind can be networked and can be connected to one COM port (or USB port) of the host PC. Each member of the

network has to be set to an individual address. The addresses are used to associate commands with the connected controllers.

For setting the addresses, see the User Manual for the particular controller; typically DIP switches are used. For a full explanation of the command addressing *address selection* mechanism, see the Mercury Native Commands manual, MS176.

1.3.1 COM Port Addressing

The library recognizes COM ports 1 to 16 on the host (the included USB driver, if installed, makes an active USB connection appear to the library as a new COM port) and handles all timing and address selection in a user-friendly manner.

1.4 Testing

Before proceeding to custom software, of if problems arise, DLL operation and communications can be tested with MMC_DLLTest.EXE. This program is provided with the native-command-set software on the product CD. Execute it from the directory which contains the DLL and use the self-documenting interface to configure communications and invoke the indicated functions with preset arguments.

2 Function Declarations

2.1.1 C-style

The declarations of the MCC-DLL functions in C-style are as follows:

```
//-----------------------------------
// Error Base Codes
//-----------------------------------
#define EBC_open         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;
#define EBC_sendCommand 144;

//-----------------------------------
// 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;
#define ERR_axis        11;
#define ERR_parameter   12;

//-----------------------------------
// Function declarations
//-----------------------------------
int __stdcall MMC_COM_open(int PortNumber, int baudrate);
int __stdcall MMC_COM_close(void);
int __stdcall MMC_COM_setBuffer(void);
int __stdcall MMC_COM_EOF(void);
int __stdcall MMC_COM_clear(void);
int __stdcall MMC_getChar(char *character);
int __stdcall MMC_getDLLversion(void);
int __stdcall MMC_getMacro(int macno,char *report);
int __stdcall MMC_getPos(void);
int __stdcall MDC_getPosErr(void);

int __stdcall MMC_getReport(char *command, char *report);
int __stdcall MMC_getSTB(int bytenumber);
int __stdcall MMC_getString(char *report, WORD count);
int __stdcall MMC_getStringCR(char *report);
int __stdcall MMC_getVal(int command_ID);
int __stdcall MMC_initNetwork(int maxAxis);
int __stdcall MMC_moveA(int axis, int position);
```

```
int __stdcall MMC_moveR(int axis, int shift);
int __stdcall MDC_moving(void);
int __stdcall MST_moving(void);

int __stdcall MMC_setDevice(int axis);
int __stdcall MMC_select(int axis);
int __stdcall MMC_sendChar(char character);
int __stdcall MMC_sendString(char *sendString);
int __stdcall MMC_sendCommand(char *command);
int __stdcall MDC_waitStop(void);
int __stdcall MST_waitStop(void);

int __stdcall RED_getJoy(int axis);
int __stdcall RED_getSCC(int command_ID);
int __stdcall RED_getReport(int axis, int command_ID, char *report);
int __stdcall RED_moving(void);
int __stdcall RED_waitStop(int axis);

int __stdcall MRC_getDLLversion(void);
int __stdcall MRC_initNetwork(int MaxAxis);
int __stdcall MRC_select(int newAxis);
int __stdcall MRC_setDevice(int newAxis);
int __stdcall RED_getJoy(int axis);
int __stdcall RED_getReport(BYTE axis, BYTE cmd, char *report);
int __stdcall RED_getSCC(int command_ID);
```

2.1.2 Pascal-Style

The declarations of the MCC-DLL functions in Pascal style are as follows:

```
const
   ExtLib = 'MMC.DLL';

   // 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;
   EBC_sendCommand = 144;

   // 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;

//---------------------------------------------------------------------------
//  Function Declarations  :
//---------------------------------------------------------------------------
```

```
// Case sensitive notation !!

function MMC_COM_open(portnumber,baudrate:integer):integer;
         stdcall external ExtLib;
function MMC_COM_close:integer;
         stdcall external ExtLib; //
function MMC_COM_setBuffer:integer;
         stdcall external ExtLib; //
function MMC_sendString(pCmd:pChar):integer;
         stdcall external ExtLib; //
function MMC_sendCommand(pCmd:pChar):integer;
         stdcall external ExtLib; //
function MMC_getPos:integer;
         stdcall external ExtLib; //
function MDC_getPosErr:integer;
         stdcall external ExtLib; //
function MMC_getVal(query:integer):integer;
         stdcall external ExtLib; //
function MMC_getReport(pCmd,psRead:PChar):integer;
         stdcall external ExtLib; //
function MMC_getStringCR(psRead:PChar):integer;
         stdcall external ExtLib; //
function MDC_moving:integer;
         stdcall external ExtLib; //
function MST_moving:integer;
         stdcall external ExtLib; //
function MMC_initNetwork(maxaxis:integer):integer;
         stdcall external ExtLib; //
function MMC_select(newaxis:integer):integer;
         stdcall external ExtLib; //
function MMC_setDevice(newaxis:integer):integer;
         stdcall external ExtLib; //
function MMC_COM_clear:integer;
         stdcall external ExtLib; //
function MMC_COM_EOF:integer;
         stdcall external ExtLib; //
function MMC_getSTB(byteno:integer):integer;
         stdcall external ExtLib; //
function MDC_waitStop:integer;
         stdcall external ExtLib; //
function MST_waitStop:integer;
         stdcall external ExtLib; //
function MMC_getDLLversion:integer;
         stdcall external ExtLib;
function MMC_moveA(axis,position:integer):integer;
         stdcall external ExtLib;
function MMC_moveR(axis,shift:integer):integer;
         stdcall external ExtLib;
function MMC_getMacro(macno:integer;content:PChar):integer;
         stdcall external ExtLib;
function MMC_globalBreak:integer;
         stdcall external ExtLib;
//------------------------------------------------------------------------------
```

2.2 Programming Templates

NOTE

The Device Numbers required by these functions run from 1 to 16. These values are 1 greater than the bitmapped, negative-logic binary DIP switch settings typically used for setting Mercury-Class controller addresses.

2.2.1 Example 1

With one C-862 or C-863 controller with all 4 Address DIP switches set to ON (Device #1), connected to COM port 1 of the PC, your application may start as follows:

```
iErr = MMC_COM_open(1,9600)              // open COM port 1 with 9600 baud
iErr = MMC_setDevice(1)                  // activate device #1 (Mercury™ with address 0)
iErr = MMC_getReport("VE", report)       // requests version report
display(report)

// set motion parameters
iErr = MMC_sendCommand ("DP180")         // set p-term parameter to 120
iErr = MMC_sendCommand ("DI35")          // set i-term parameter to 35
iErr = MMC_sendCommand ("DD230")         // set d-term parameter to 230
iErr = MMC_sendCommand ("SV56000")       // set velocity to 56000 counts/s
iErr = MMC_sendCommand ("SA450000")      // set acceleration to 450,000 counts/s

iErr = MMC_sendCommand ("MR50000")       // start a relative move of 50000 counts

  add here all further commands

MMC_COM_close                            // before leaving the application close the COM port
```

2.2.2 Example 2

If you have connected three Mercury™ controllers to COM 1 port and the controllers are set to addresses to (negative-logic binary) 0000,0001 and 0010 (i.e. device numbers 1, 2 and 3), the application may start as follows:

(pseudo code):

```
iErr = MMC_COM_open(1,9600)              open COM port 1 at 9600 baud
iErr = MMC_initNetwork(3)                scan all networked devices starting with device #3
                                         down to device #1. This function takes about 1s per
                                         device to execute.
iErr = MMC_select(1)                     select device #1
iErr = MMC_sendCommand ("DP120")         set p-term
iErr = MMC_sendCommand ("DI15")          set i-term
iErr = MMC_sendCommand ("DD300")         set d-term
iErr = MMC_sendCommand ("SV56000")       set velocity to 56000 counts/s
iErr = MMC_sendCommand ("SA450000")      set acceleration to 450,000 counts/s

iErr = MMC_select(2)                     select device #2
iErr = MMC_sendCommand ("DP120")         set p-term
```

```
iErr = MMC_sendCommand ("DI15")          set i-term
iErr = MMC_sendCommand ("DD300")         set d-term
iErr = MMC_sendCommand ("SV56000")       set velocity to 56000 counts/s
iErr = MMC_sendCommand ("SA450000")      set acceleration to 450,000 counts/s

iErr = MRC_select(3)                     select device #3
iErr = MMC_sendCommand ("DP120")         set p-term
iErr = MMC_sendCommand ("DI15")          set i-term
iErr = MMC_sendCommand ("DD300")         set d-term
iErr = MMC_sendCommand ("SV56000")       set velocity to 56000 counts/s
iErr = MMC_sendCommand ("SA450000")      set acceleration to 450,000 counts/s

iErr = MMC_moveR(1,50000)                move Mercury™ #1
iErr = MMC_moveR(4,-150000)              move Mercury™ #4
iErr = MMC_moveR(5,-400000)              move Mercury™ #5


  add here further commands of your program

MMC_COM_close  // before terminating the application close the COM port.
```

3 Function Reference

Including LabVIEW VI designations

int **MMC_COM_open**(int portnumber, int baudrate)

Purpose / Action	Opens the specified COM port for communication with Mercury™ controllers. When connected over USB, the USB driver, when properly installed, causes the USB interface to appear to the DLL as a new COM port.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	portnumber: Number of COM port, range 1 to 16 baudrate: Baud rate, can be either 9600 or 19200, must agree with the baud rates set in DIP switches on all units in the network, even if only USB is used.
Return:	Error codes: 0: No error 20: Timeout 22: _getCommState error 23: _setCommState error 25: Wrong port number 26: Handle error
Ordinal Index	d50
LabView VI	MMC_COM_open.vi

int **MMC_COM_close**(void)

Purpose / Action	Closes the COM port previously opened by the MMC_COM_open function.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	none
Return:	Error codes: 0: No error 1: error
Ordinal Index	d52
LabView VI	MMC_COM_close.vi

int **MMC_COM_setbuffer**(void)

Purpose / Action	Defines input and output buffers for serial communication. Mainly used with Windows 95/98 and NT. No need to use this function with Windows 2000/XP.
Applicable Controllers	C-862 Mercury™ (DC motor) C-663 Mercury™ Step C-863 Mercury™ (DC motor) C-170 Redstone
Arguments:	none
Return:	Error codes: 0: No error 32: Error
Ordinal index	d54

int **MMC_COM_EOF**(void)

Purpose / Action	Returns the number of characters available in the COM-port input buffer
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	none
Return:	Number of characters in the input buffer
Ordinal index	d56
LabView VI	MMC_COM_EOF.vi

int **MMC_COM_clear**(void)

Purpose / Action	Clears the COM-port input buffer.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	none
Return:	Error codes: 0: No error 1: Error
Ordinal index	d58
LabView VI	MMC_COM_clear.vi

int **MMC_getChar**(char *character)

Purpose / Action	Reads one character from the COM-port.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	character: pointer to character
Return:	Error code: 0: No error 65: Error: readfile 67: Error: timeout
Ordinal index	d2
LabView VI	none

int **MMC_getDLLversion**(void)

Purpose / Action	Delivers the version number of the DLL
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	none
Return:	Version number of DLL as integer
Ordinal index	d62
LabView VI	none

int **MMC_getMacro**(int macronumber, char *report)

Purpose / Action	Reads the specified macro command as ASCII string
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	macronumber: Number of the macro to be read, range 0 to 31 report: Pointer to the character buffer
Return:	Error Code: 0 : No Error
Ordinal index	d10
LabView VI	MMC_getMacro.vi

int **MMC_getPos**(void)

Purpose / Action	Reads the current motor position of the currently selected Mercury™ controller. The reading process does not interrupt running compound commands.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step
Arguments:	none
Return:	Current motor position in counts/steps or error code. The error code is derived from maximum integer value minus the error number: 2,147,483,647 (maxint) : Wrong Content 2,147,483,646 (maxint-1) : Error in _getString 2,147,483,645 (maxint-2) : Error in _sendString 2,147,483,644 (maxint-3) : Error during conversion
Ordinal index	d18
LabView VI	MMC_getPos.vi

int **MDC_getPosErr**(void)

Purpose / Action	Reads the current motor-position error of the currently selected Mercury™ controller.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor)
Arguments:	none
Return:	Current motor position error in counts or error code. The error code is derived from maximum integer value minus the error number: 2,147,483,647 (maxint) : Wrong Content 2,147,483,646 (maxint-1) : Error in _getString 2,147,483,645 (maxint-2) : Error in _sendString 2,147,483,644 (maxint-3) : Error during conversion
Ordinal index	d20
LabView VI	MDC_getPosErr.vi

int **MMC_getReport**(char *command, char *report)

Purpose / Action	Sends a report command and reads the report generated. The report command string can be either a regular command like "TT" or a single character command like "%".
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	command: Pointer to command string report: Pointer to report string
Return:	Error code: 0: No error
Ordinal index	d24
LabView VI	MMC_getReport.vi

int **MMC_getSTB**(int bytenumber)

Purpose / Action	Returns one of the bytes of the status report (response to TS command) of the currently active controller. Depending on the active controller type, the TS command reports a different number of bytes: 6 Bytes for C-863 and C-862 Mercury™ (DC motor) , 5 Bytes for Redstone 3 bytes for Mercury™-Step.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	bytenumber: Index of status byte to be returned. The number should not exceed the maximum number of bytes the device connected can deliver. Parameter range: 1 to 6 for C-862 and C-863 Mercury™ (DC motor) DC, 1 to 5 for C-170 Redstone, 1 to 3 for C-663 Mercury™ Step
Return:	Byte value or error code >=0: Byte value Error codes: -1: Error, contents -2: Error, _getReport -3: Error, Parameter out of range
Ordinal index	d26
LabView VI	MDC_getSTB.vi

int **MMC_getString**(char *pReport, WORD count)

Purpose / Action	Reads a defined number of characters from the COM port.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	pReport: Pointer to the character buffer count: Number of characters to be read.
Return:	Error code: 0: No error 81: Error: readfile 84: Error: timeout, not enough characters received in time
Ordinal index	d4
LabView VI	none

int **MMC_getStringCR**(char *pReport)

Purpose / Action	Reads characters from the COM-port up to the first appearance of CR (character d13).
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	pReport: Pointer to character buffer.
Return:	Error codes: 0: No error 255: Error: timeout 65: Error: readfile 67: Error: timeout
Ordinal index	d8
LabView VI	MMC_getStringCR.vi

int **MMC_getVal**(int command_ID)

Purpose / Action	Reads the value of the requested parameter. The function can be called on the fly. Running compound commands or macros are not interrupted.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step
Parameter:	command_ID: Identifier for the requested item: 1 = TP (Tell Position) 2 = TT (Tell Target) 3 = TF (Tell profile following error) 4 = TE (Tell distance to target) 5 = TY (Tell velocity setting) 6 = TL (Tell acceleration setting) 7 = GP (Get p-term setting) 8 = GI (Get i-term setting) 9 = GD (Get d-term setting) 10 = GL (Get i-limit setting)
Return	The requested value or error code is returned as 32-bit integer. Error codes: 2,147,483,647 (MaxInt) = content error 2,147,483,646 (MaxInt-1) = getString error 2,147,483,645 (MaxInt-2) = sendString error 2,147,483,644 (MaxInt-3) = conversion error
Ordinal index:	d22
LabView VI	MMC_getVal.vi

int **MMC_initNetwork**(int maxAxis)

Purpose / Action	Searches all addresses, starting at address maxAxis down to 1 for Mercury™ devices connected. If a Mercury™ device (can be C-862, C-863, C-663 or C-170) is found, it is registered so as to allow access through the MMC_select() function. The function MMC_initNetwork is optional. If it is not used, devices can be activated anyway using the MMC_setDevice function.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	maxAxis: This parameter represents the highest device number from which the search is to run, continuing downwards. If you have 3 Mercury™s connected at the addresses 0,1 and 2 (this equals the device numbers 1,2 and 3) you may call the function as MMC_initNetwork(3). If you do no know what addresses the controllers are set to, call the function with maxAxis = 16 to find all devices connected. (Remember that valid device numbers range from 1 to 16.) The range of maxAxis is 1 to 16 Because scanning each address takes about 0.5 seconds, it saves time to not start at device numbers higher than required.
Return:	Error Codes: 0: No controller found > 0: Controller(s) found. The lower 16 bit of that number represent the device connect table (1=connected). < 0: Error
Ordinal index	d28
LabView VI	MMC_initNetwork.vi

int **MMC_moveA**(int axis, int position)

Purpose / Action	Moves the motor of the specified axis (device number) to specified position.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step
Arguments:	*axis:* If this parameter is 0 then the move command is sent to the currently selected device. If it is >0 then an address selection code will be sent for the specified axis addressed before the move command is sent. *position:* The new target position
Return:	Error codes: 0: No error 1: Error, wrong axis 2: Error, not connected 3: Error, sendString
Ordinal index	d42
LabView VI	MMC_moveA.vi

int **MMC_moveR**(int axis, int shift)

Purpose / Action	Moves the motor of the specified axis (device number) relative to its current position by *shift* counts or steps.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step
Arguments:	*axis:* If this parameter is 0 then the move command is sent to the currently selected device. If it is >0 then an *address selection code* will be sent for the specified axis before the move command is sent. *shift:* Position increment added to the current position.
Return:	Error codes: 0: No error 1: Error, wrong axis 2: Error, not connected 3: Error, sendString
Ordinal index	d44
LabView VI	MMC_moveR.vi

int **MDC_moving**(void)

Purpose / Action	Returns the motion status of the currently selected C-862 or C-863 Mercury™ DC motor controller. For C-663 Mercury™-Step controllers, an equivalent function is available.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor)
Arguments:	none
Return:	Return codes: 0: Not moving 1: moving -1: Error, content -2: Error, query
Ordinal index	d34
LabView VI	MDC_moving.vi

int **MST_moving**(void)

Purpose / Action	Returns the moving status of the currently selected Mercury™-Step controller. For Mercury™ DC motor controllers, an equivalent function is available.
Applicable Controllers	C-663 Mercury™-Step
Arguments:	none
Return:	Return codes: 0: Not moving 1: moving -1: Error, content -2: Error, query
Ordinal index	d36
LabView VI	MST_moving.vi

int **MMC_setDevice**(int axis)

Purpose / Action	Addresses the selected axis (controller). This function works anytime and it is not required to have registered the devices connected with the MMC_intNetwork function. See also MMC_select()
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	axis: Range 1 to 16, Device number of the controller that shall be selected for communication. The device number or address can be set by the controller's front panel DIP switches.
Return:	Error codes: 0: No error 1: Wrong axis number
Ordinal index	d32
LabView VI	MMC_setDevice.vi

int **MMC_select**(int axis)

Purpose / Action	Selects the specified axis (device) to enable communication with it. Unlike the MMC_setDevice function, here the registration status is checked, so this function requires that the MMC_initNetwork function have been called previously at the beginning of the program.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	axis: range 1 to 16 Device number of the controller that is to be selected for communication.
Return:	Error codes: 0: No error 1: Wrong axis number 2: Axis not registered
Ordinal index	d30
LabView VI	MMC_select.vi

int **MMC_sendChar**(char character)

Purpose / Action	Sends one character to the device that is currently selected.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	character: Character to be sent
Return:	Error codes: 0: No error 98: Write error 100: Length error
Ordinal index	d12
LabView VI	none

int **MMC_sendString**(char *sendstring)

Purpose / Action	Sends the character string to the device that is currently selected. No termination character is added to the string. If you want to send a command string that requires a CR as terminator, use the MMC_sendCommand function.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	sendstring: Pointer to the buffer holding the string to be sent.
Return:	Error codes: 0: No error 114: Write error 116: Length error
Ordinal index	d14
LabView VI	MMC_sendString.vi

int **MMC_sendCommand**(char *command)

Purpose / Action	Sends the command string to the device that is currently selected. A CR (character d13) as terminator is added to the command string for immediate execution. If you want to send a string without termination character, use the MMC_sendString function instead.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	command: Pointer to the buffer holding the command string to be send.
Return:	Error codes: 0: No error 114: Write error 116: Length error
Ordinal index	d16
LabView VI	MMC_sendCommand.vi

int **MDC_waitStop**(void)

Purpose / Action	Waits until the current move has terminated or interrupted by user command (function MCC_GlobalBreak).
Designated Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor)
Arguments:	none
Return	Error Codes: 0: No error 1: Error, query 2: User break
Ordinal index:	d38
LabView VI	MDC_waitStop.vi

int **MST_waitStop**(void)

Purpose / Action	Waits until the current move has terminated or interrupted by user command (function MCC_GlobalBreak).
Designated Controllers	C-663 Mercury™-Step
Arguments:	none
Return	Error Codes: 0: No error 1: Error, query 2: User break
Ordinal index:	d40
LabView VI	MST_waitStop.vi

int **MMC_globalBreak**(void)

Purpose / Action	This function interrupts pending operations waiting for termination of a move. Can be used with _moving() or _waitStop functions.
Applicable Controllers	C-862 Mercury™ (DC motor) C-863 Mercury™ (DC motor) C-663 Mercury™-Step C-170 Redstone
Arguments:	none
Return	Error Code: 0: No error
Ordinal index:	d60
LabView VI	MMC_globalBreak.vi

int **RED_getJoy**(int axis)

Purpose / Action	Reads the joystick values for either the X and the Y axes of a joystick connected to a Redstone controller.
Applicable Controllers	C-170 Redstone
Arguments:	axis: 1: (X-axis) 2: (Y-axis)
Return:	Joystick value, range 0 to 255 < 0: Error code
Ordinal index	d84

int **RED_getReport**(int axis, int Cmd_ID, char *report)

Purpose / Action	Sends the report command and reads the report string generated.
Applicable Controllers	C-170 Redstone
Arguments:	axis: 1: (X-axis) 2: (Y-axis) cmd_ID: Command index number: 1 : SD? 2 : SI? 3 : SR? 4 : SS? 5 : SW? 6 : SJ?
Return:	Error code 0 : No error
Ordinal index	d82

int **RED_getSCC**(int SC_ID)

Purpose / Action	Returns a value generated by a single character command.
Applicable Controllers	C-170 Redstone
Arguments:	SC_ID: Single char command index number 1 : TA1 2 : TA2 3 : TA3 4 : TA4 5 : TC
Return:	>=0 : Requested value < 0 : Error code
Ordinal index	d80

int **RED_moving**(void)

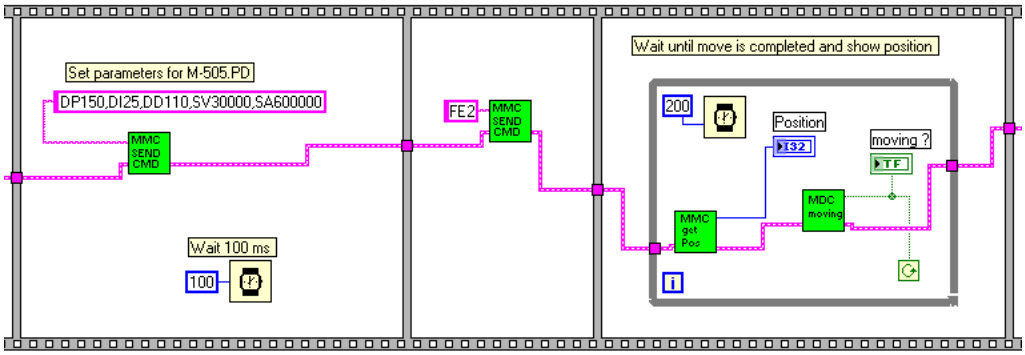
Purpose / Action	Returns a value indicating whether the motor(s) are moving or not
Applicable Controllers	C-170 Redstone
Arguments:	none
Return:	Moving Status: 0 : Not moving 1 : axis 1 is moving 2 : axis 2 is moving 3 : both axes are moving <0 : Error code
Ordinal index	d86

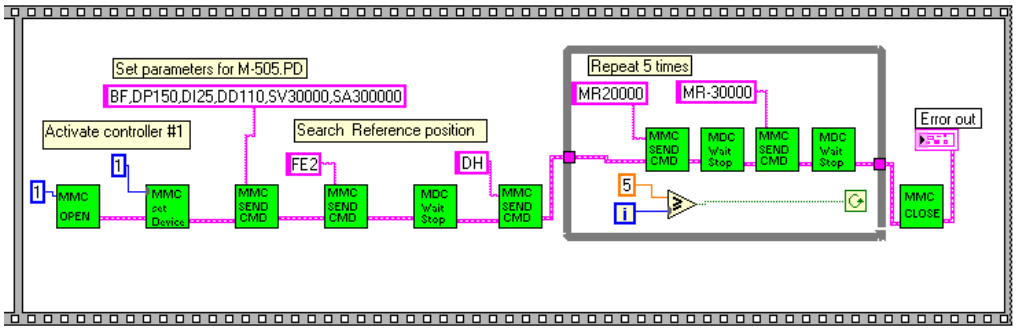
int **RED_waitStop**(int axis)

Purpose / Action	Waits until the motor has terminated its move.
Applicable Controllers	C-170 Redstone
Arguments:	axis, range 1 to 2
Return:	Status: 0 : Motion completed 1 : Unexpected result <0 : Error code
Ordinal index	d88

4 LabView Driver VIs

All LabView vis are based on the current MMC.DLL (current version is 4.13)
For detailed description of the DLL functions see the Function Reference section, starting on p. 10.





vi Name	Function	
COM port initialization		
MMC_COM_open.vi	Opens and configures a COM port for Mercury™ specific data transfer	
MMC_COM_close.vi	Close COM port	
MMC_COM_clear.vi	Erases the communication input buffer	
MMC_COM_EOF.vi	Reads the number of characters available in the input buffer	
Mercury™ Initialization		
MMC_initNetwork.vi	Initializes Mercury™ network	
MMC_select.vi	Selects one member of the network	
MMC_setDevice.vi	Sends the address code of the device	
Sending Commands		
MMC_sendString.vi	Send a command string without terminatior	
MMC_sendCommand.vi	Send a command string with terminator	
MMC_moveA.vi	Move to an absolute position	
MMC_moveR.vi	Move for a relative increment	
Receiving Data		
MMC_getStringCR.vi	Read a string until CR from input buffer	
MMC_getPos.vi	Read the current motor position	
MDC_getPosErr.vi	Read the current position error from Mercury™ (DC motor)	
MMC_getReport.vi	Read a report	
MMC_getMacro.vi	Read the specified macro	
MMC_getVal.vi	Read the requested value	
MMC_STB.vi	Read one status byte	
Waiting for Events		
MDC_WaitStop.vi	Wait for motion stop of Mercury™ (DC motor) controller	
MST_WaitStop.vi	Wait for motion stop of Mercury™-Step controller	
MDC_moving.vi	Read moving status of Mercury™ (DC motor)	
MST_moving.vi	Read moving status of Mercury™-Step	