Motion Control Application Programming Interface

MCAPI Reference Manual

Revision 3.4

Precision MicroControl Corporation
2075-N Corte del Nogal
Carlsbad, CA 92009-1415 USA

Tel: (760) 930-0101
Fax: (760) 930-0222

www.pmccorp.com

Information: info@pmccorp.com

Technical Support: support@pmc.com

LIMITED WARRANTY

All products manufactured by PRECISION MICROCONTROL CORPORATION are guaranteed to be free from defects in material and workmanship, for a period of five years from the date of shipment. Liability is limited to FOB Factory repair, or replacement, of the product. Other products supplied as part of the system carry the warranty of the manufacturer.

PRECISION MICROCONTROL CORPORATION does not assume any liability for improper use or installation or consequential damage.

Information in this document is subject to change without notice.

Precision MicroControl
2075-N Corte del Nogal
Carlsbad, CA 92009-1415

Phone: (760)930-0101
Fax: (760)930-0222
World Wide Web: www.pmccorp.com
Email:
Information: info@pmccorp.com
Technical support: support@pmccorp.com
Sales: sales@pmccorp.com

Table of Contents

User manual revision history

Revision	Date	Description
1.0	3/12/2001	Initial release
	5/4/2001	Edited to match MCAPI version 3.01
	8/15/2001	Miscellaneous edits
3.1	11/8/2001	New functions added and format changed
3.2	1/28/2002	Closed-loop stepper functionality fully supported
	2/22/2002	New functions added to support homing
3.3	1/16/2003	Updated to MCAPI 3.3 (added support for MultiFlex PCI 1000 Series)
3.4	6/5/2003	Updated to MCAPI 3.4

Prologue

This manual has been written as a reference manual. This is not meant to be the only document you should reference regarding the Motion Control Application Programming Interface (MCAPI). You will find more application specific information on how to use your motion control card with the MCAPI in your User's Manual, as well as detailed and commented code examples in the online help.

Also, you will find other valuable information on how to use your motion control card on your **MotionCD**. There, you will find the following information:

- Tutorials (PowerPoint presentations)
  - An Introduction to PMC Motion Control
  - Installing a PMC Motion Controller (Does not Address PCI bus controllers)
  - Introduction to Motion Control Programming with the Motion Control API
  - Servo Systems Primer
  - Servo Tuning

- PMC AppNOTES – detailed descriptions of specific motion control applications

- PMC TechNOTES – one page technical support documents

- PMC Product catalogs and brochures

Chapter Contents

- Introduction to the Motion Control Application Programming Interface (MCAPI)
- Controller Interface Types
- Building Application Programs using the MCAPI
- C/C++ Programming Introduction
- Visual Basic Programming Introduction
- Delphi Programming Introduction
- LabVIEW programming Introduction
- MCAPI Online Help
  - MCAPI Users Guide
  - MCAPI online function reference
  - MCAPI Common Dialog help
  - LabVIEW Motion VI Library Help

Chapter

1

Introduction

PMC's motion control cards and modules integrate seamlessly into high performance, Windows applications. The **Motion Control Application Programming Interface** (**MCAPI**) provides support for all popular high level languages. Additionally, the board level command language, **Motion Control Command Language** (**MCCL**), allows the machine designer to execute local 'macro' routines independent of the PC host and application program.

PMC's MCAPI is a group of Windows components that, taken together, provide a consistent, high level, Applications Programming Interface (API) for PMC's motion controllers. The difficulties of interfacing to new controllers, as well as resolving controller specific details, are handled by the API, leaving the applications programmer free to concentrate on the application program.

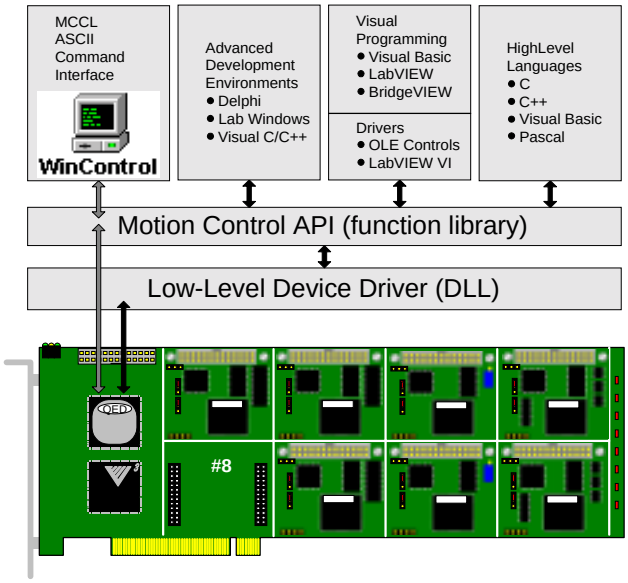


Figure 1:MCAPI and motion control card architectural diagram

The API has been constructed with a layered approach. As new versions of Windows operating systems and new PMC motion controllers become available, API support is provided by simply replacing one or more of these layers. Because the public API (the part the applications programmer sees) is above these layers, few or no changes to applications programs will be required to support new version of the MCAPI.

The API itself is implemented in three parts. The low level device driver provides communications with the motion controller, in a way that is compatible with the Microsoft Windows operating system. The MCAPI low level driver passes binary MCCL commands to the motion control card. By placing the operating system specific portions of the API here it will be possible to replace this component in the future to support new operating systems without breaking application programs, which rely on the upper layers of the API.

Sitting above that, and communicating with the driver is the API Dynamic Link Library (DLL). The DLL layer implements the high level motion functions that make up the API. This layer also handles the differences in operation of the various PMC Motion Controllers, making these differences virtually transparent to users of the API.

At the highest level are environment specific drivers and support files. These components support specific features of that particular environment or development system.

Care has been exercised in the construction of the API to ensure it meets with Windows interface guidelines. Consistency with the Windows guidelines makes the API accessible to any application that can use standard Windows components - even those that were developed after the Motion Control API!

Controller Interface Types

Each motion control card supports two onboard interfaces, an ASCII (text) based interface and a binary interface. The binary interface is used for high speed command operation, and the ASCII interface is used for interactive text based operation. The high level sample programs (CWDEMO and VBDEMO) use the binary interface, PMC Win Control uses the ASCII interface.

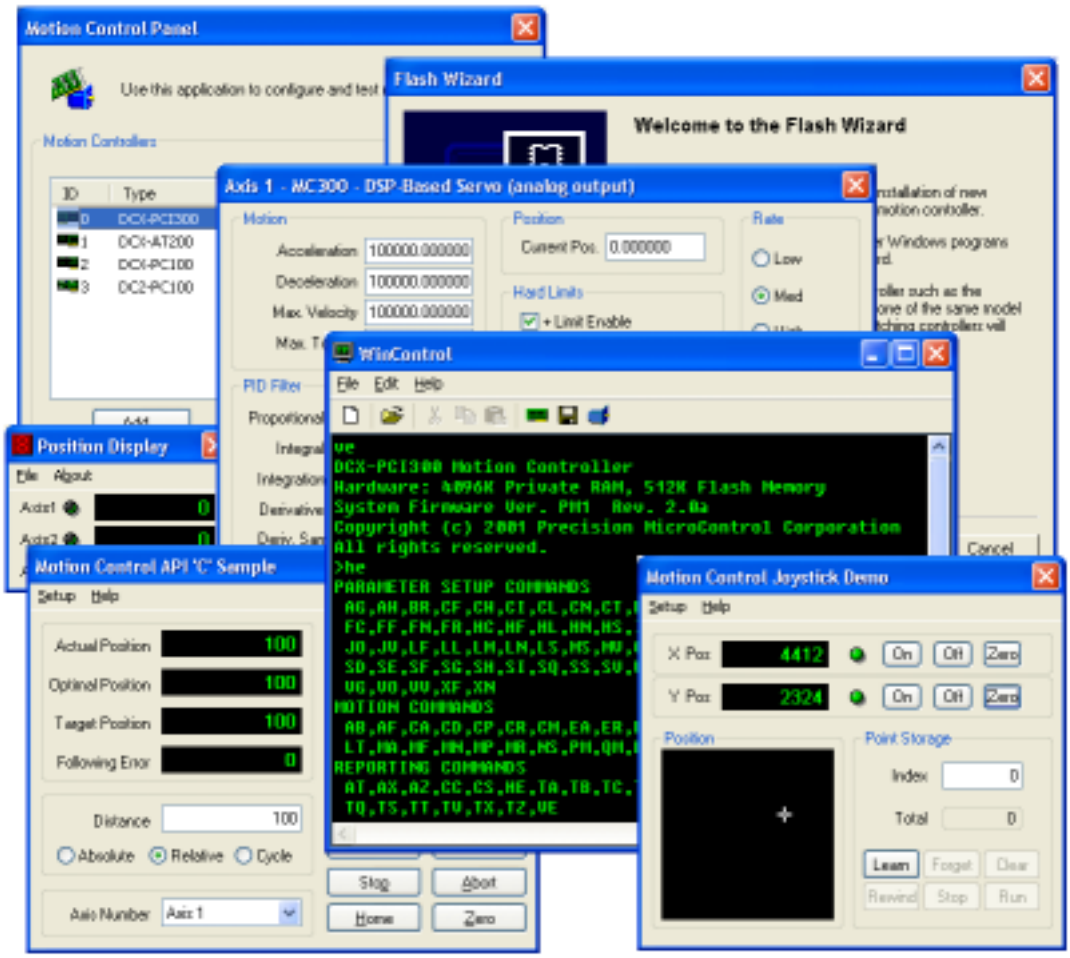
Application programs must indicate which interface they intend to use when they open a handle for a particular controller. A controller may have more than one handle open at a time. While multiple binary interfaces may be open at once, no more than one ASCII interface open at a time (with or without multiple binary interfaces open) is recommend. The open mode is specified by setting the second argument of the ***MCOpen( )*** function to either MC_OPEN_ASCII or MC_OPEN_BINARY.

Building Application Programs using Motion Control API

The Motion Control Application Programming Interface (MCAPI) is designed to allow a programmer to quickly develop sophisticated application programs using popular development tools. The MCAPI provides high level function calls for:

- Configuring the controller (servo tuning parameters, velocity and ramping, motion limits, etc.)
- Defining on-board user scaling (encoder/step units, velocity units, dwell time units, user and part zero)
- Commanding motion (Point to Point, Constant Velocity, Electronic Gearing, Lines and Arcs, Joystick control)
- Reporting controller data (motor status, position, following error, current settings)
- Monitoring Digital and Analog I/O
- Driver functions (open controller handle, close controller handle, set timeout)

Included with the installation of the MCAPI is the Sources ‘folder’. In this folder are complete program sample source files for C++, Visual Basic, Delphi.



C/C++ Programming Introduction

Included with each of the C program samples (CWDemo, Joystick demo, and Win Control) is a read me file (readme.txt) that describes how to build the sample program. The following text was reprinted from the readme.txt file for the CWDemo program sample.

Contents
========
- How to build the sample
- LIB file issues
- Contacting technical support

How to build the sample
=======================
To build the samples you will need to create a new project or make file within your C/C++ development tool. Include the following files in your project:

CWDemo.c
CWDemo.def
CWDemo.rc

For 16-bit development you will also need:

..\mcapi.lib
..\mcdlg.lib
..\ctl3d.lib

For 32-bit development you will also need:

..\mcapi32.lib
..\mcdlg32.lib

If your compiler does not define the _WIN32 constant for 32-bit projects you will need to define it at the top of the source file (before the header files are included).

LIB File Issues
===============
Library (LIB) files are included with MCAPI for all the DLLs that comprise the user portion of the API (MCAPI.DLL, MCAPI32.DLL, MCDLG.DLL, and MCDLG32.DLL). These LIB files make it easy to resolve references to functions in the DLL using static linking (typical of C/C++). Unfortunately, under WIN32 the format of the LIB files varies from compiler vendor to compiler vendor. If you cannot use the included LIB files with your compiler you will need to add an IMPORTS section to your projects DEF file. We have included skeleton DEF files for all of the DLLs for which we also include a LIB file (MCAPI.DEF, MCAPI32.DEF, MCDLG.DEF, and MCDLG32.DEF).

The 16-bit LIB files were built with Microsoft Visual C/C++ Version 1.52, and the 32-bit LIB files Microsoft Visual Studio Version 5.

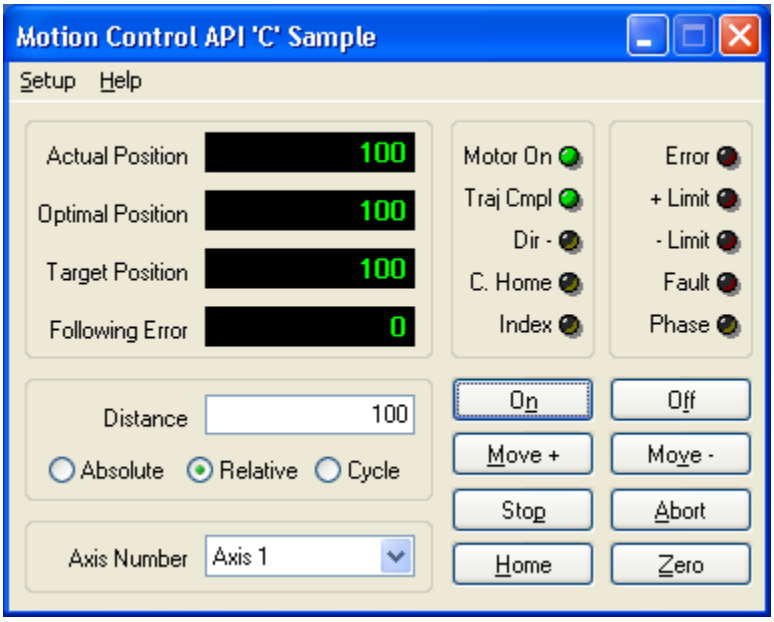


Figure 2: C/C++ program sample (CWDemo)

The C/C++ program sample (CWDemo) allow the user to:

- Move an axis (servo or stepper)
- Monitor the actual, target, and optimal positions of an axis
- Monitor axis I/O (Limits +/-, Home, Index, an Amplifier Enable)
- Define or change move parameters (Maximum velocity, acceleration/deceleration)
- Define or change the servo PID parameters

Visual Basic Programming Introduction

Included with each of the Visual Basic program samples (VBDemo, VBDemo32) is a read me file (readme.txt) that describes how to build the sample program. The following text was reprinted from the readme.txt file for the VBDemo32 program sample.

Contents
========

- About the sample
- How to build the sample
- Contacting technical support

About the sample
================
This sample demonstrates a simple user interface to one axis of a motion controller. The user may program moves and interact with the motion in a number of ways (stop it, abort it, etc.). Sample forms demonstrate how to configure servo or stepper motor axes. A number of the new MCDialog functions (such as a full-featured, ready-to-run axis configuration dialog) are also demonstrated.

How to build the sample
=======================
To build the samples you will need to create a new project or use the Visual Basic project file (created with Visual Basic v6.0) included with the sample. Include the following files if you create your own project:

About32.frm
Main32.frm
Servo32.frm
Step32.frm
VBDemo.bas

..\mcapi32.bas
..\mcdlg32.bas

Set frmMain as the startup object for the project.

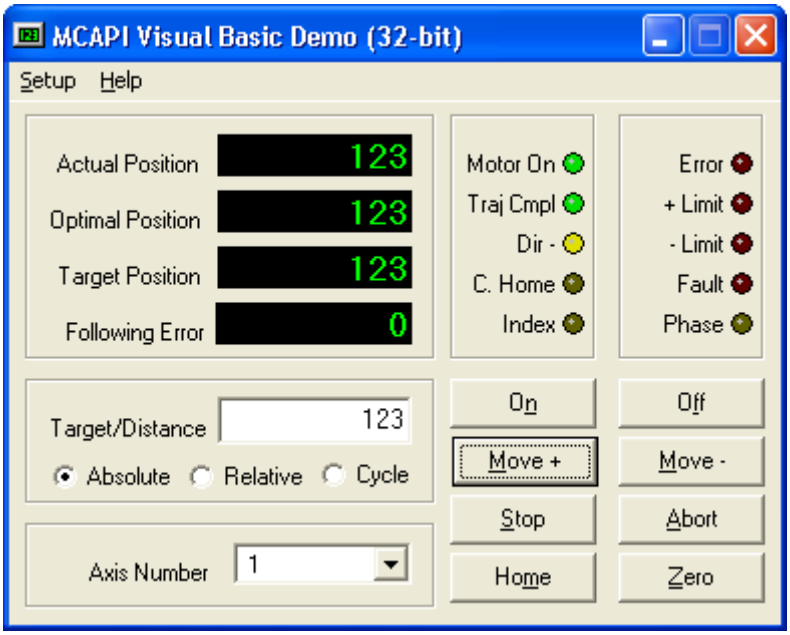


Figure 3: Visual Basic program sample (VBDemo)

The Visual Basic program sample (VBDemo) allow the user to:

- Move an axis (servo or stepper)
- Monitor the actual, target, and optimal positions of an axis
- Monitor axis I/O (Limits +/-, Home, Index, an Amplifier Enable)
- Define or change move parameters (Maximum acceleration/deceleration)
- Define or change the servo PID parameters

Delphi Programming Introduction

Included with each of the Delphi program sample (PasDemo) is a read me file (readme.txt) that describes how to build the sample program. The following text was reprinted from the readme.txt file for the PasDemo program sample.

Contents
========

- About the sample
- How to build the sample
- Contacting technical support

About the sample
================
This sample demonstrates a simple user interface to one axis of a motion controller. The user may program moves and interact with the motion in a number of ways (stop it, abort it, etc.). Sample forms demonstrate how to configure servo or stepper motor axes. A number of the new MCDialog functions (such as a full-featured, ready-to-run axis configuration dialog) are also demonstrated.

How to build the sample
=======================
To build the samples you will need to create a new project or use the Delphi project files included with the sample (Pdemo.dpr for 16-bit, Pdemo32.dpr for 32-bit). Include the following files if you create your own project:

About.pas
Global.pas
PasDemo.pas
Servo.pas
Stepper.pas

For 16-bit projects you will also need:

..\mcapi.pas
..\mcdlg.pas

For 32-bit projects you will also need:

..\mcapi32.pas
..\mcdlg32.pas

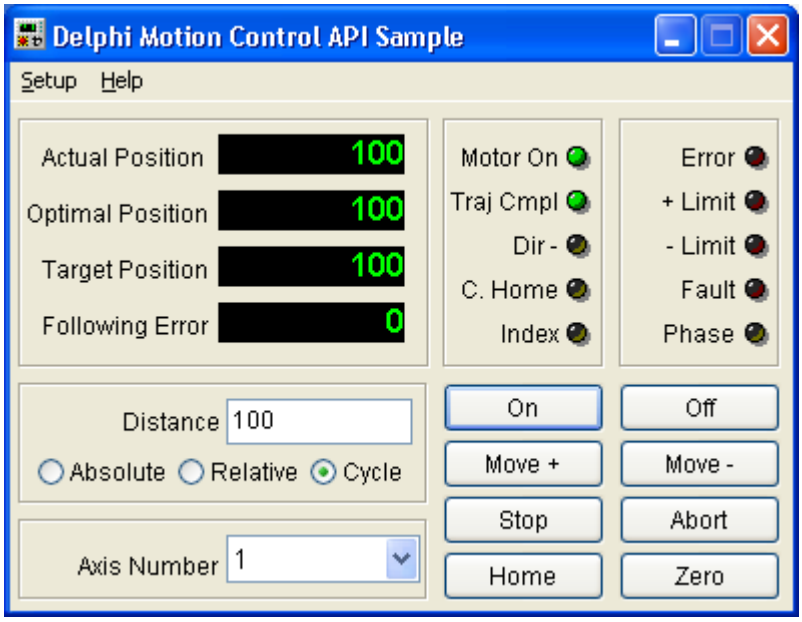


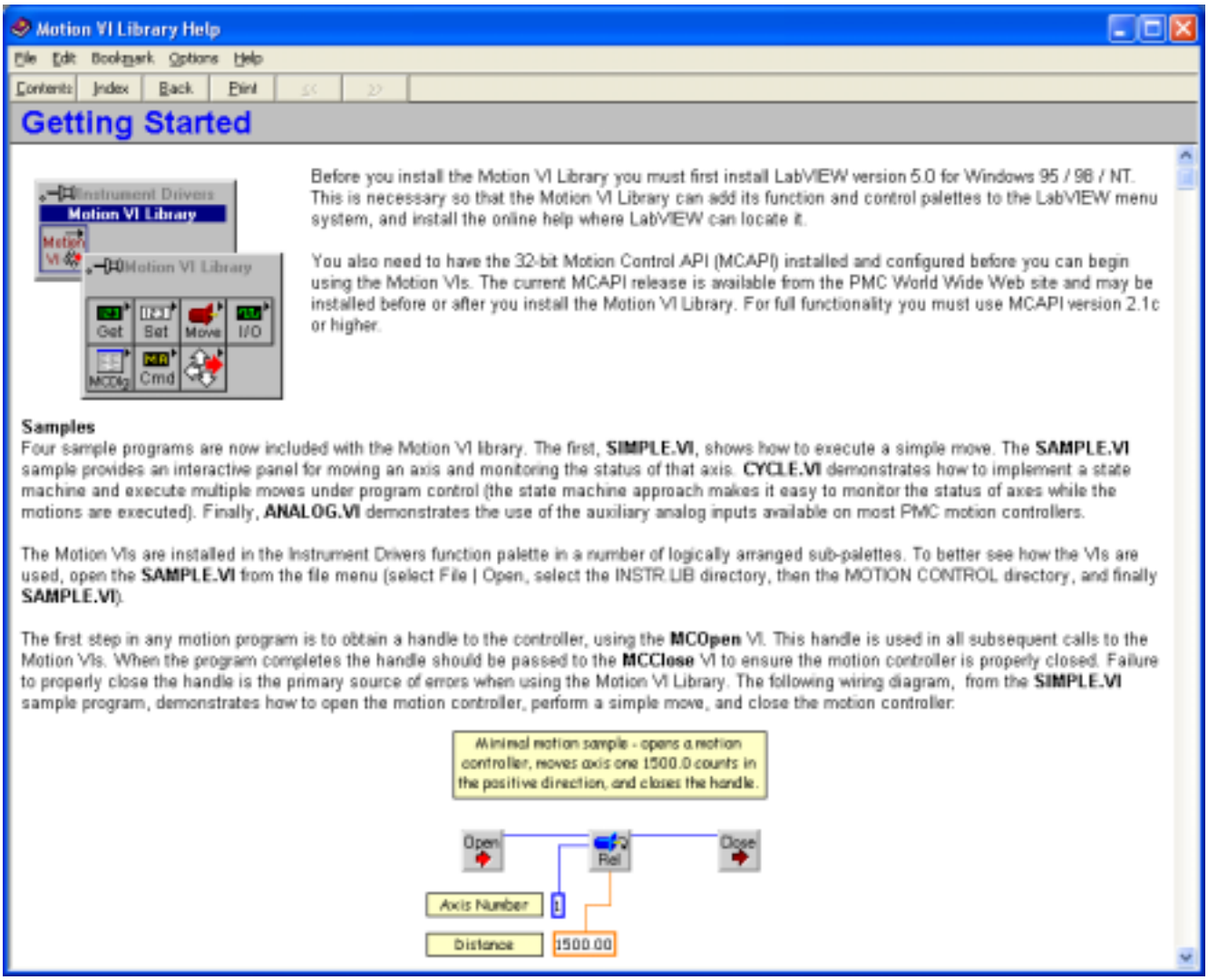
Figure 4: Delphi program sample (PasDemo)

The Delphi program sample (PasDemo) allow the user to:

- Move an axis (servo or stepper)
- Monitor the actual, target, and optimal positions of an axis
- Monitor axis I/O (Limits +/-, Home, Index, an Amplifier Enable)
- Define or change move parameters (Maximum velocity, acceleration/deceleration)
- Define or change the servo PID parameters

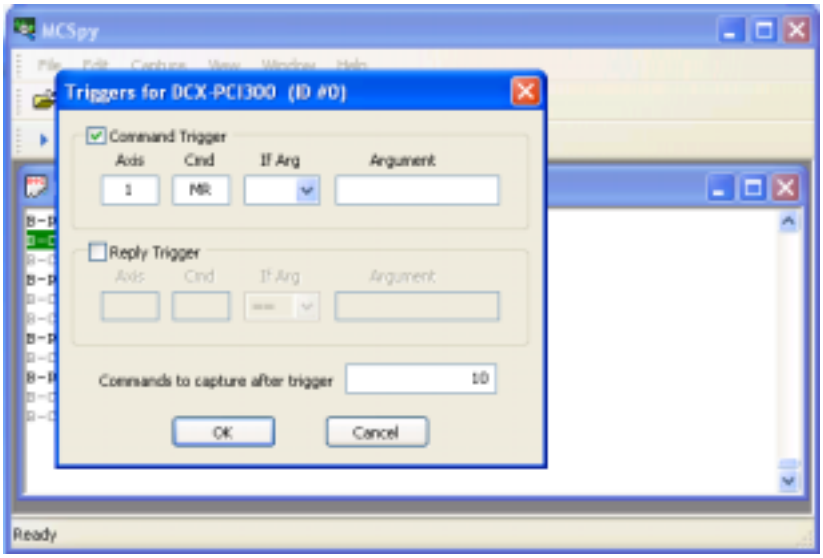
LabVIEW Programming Introduction

PMC's LabVIEW Virtual Instrument Library includes online help with a Getting Started guide.



MCSpy

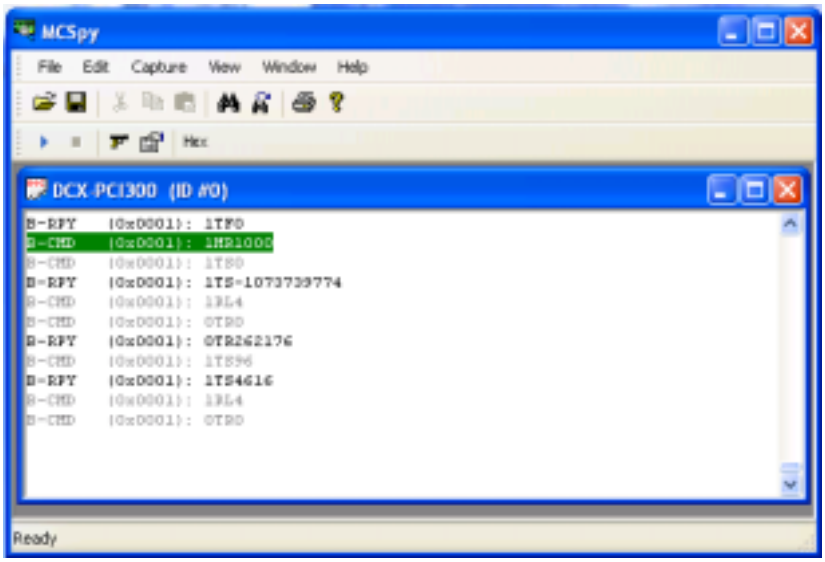
MCSpy is a debugging tool for application programs that use PMC's Motion Control API (MCAPI) programming interface. MCSpy captures commands and replies sent between the application program and the motion control card. These commands are displayed in Motion Control Command Language (MCCL), which is the language the MCAPI uses to communicate with PMC's Motion



The MCSpy Trigger Setup dialog allows the user to terminate the capturing of commands / replies data after the trigger event.

Here the command /reply capture will end 10 commands after a move relative (MR) command has been issued to axis #1.

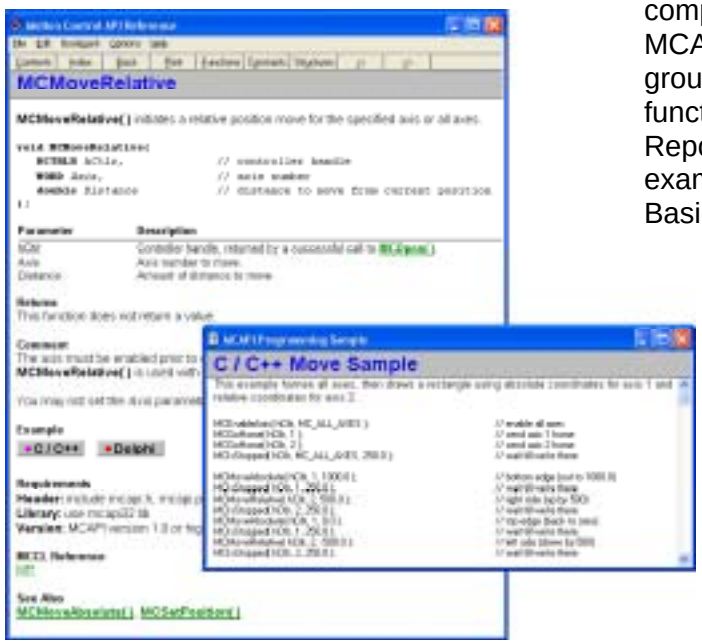
The Trigger Event (1MR1000) is highlighted in green.



MCAPI Online Help

Complete and up to date online help for PMC's Motion Control Application Programming Interface (MCAPI) at PMC's website www.pmccorp.com. Help documents include; installation and basic usage, complete function call reference and example code, high level dialog descriptions, and LabVIEW VI Library reference.

The online MCAPI Users Guide describes the basics of PMC's MCAPI. This should be the '**first stop**' for any questions about the MCAPI.

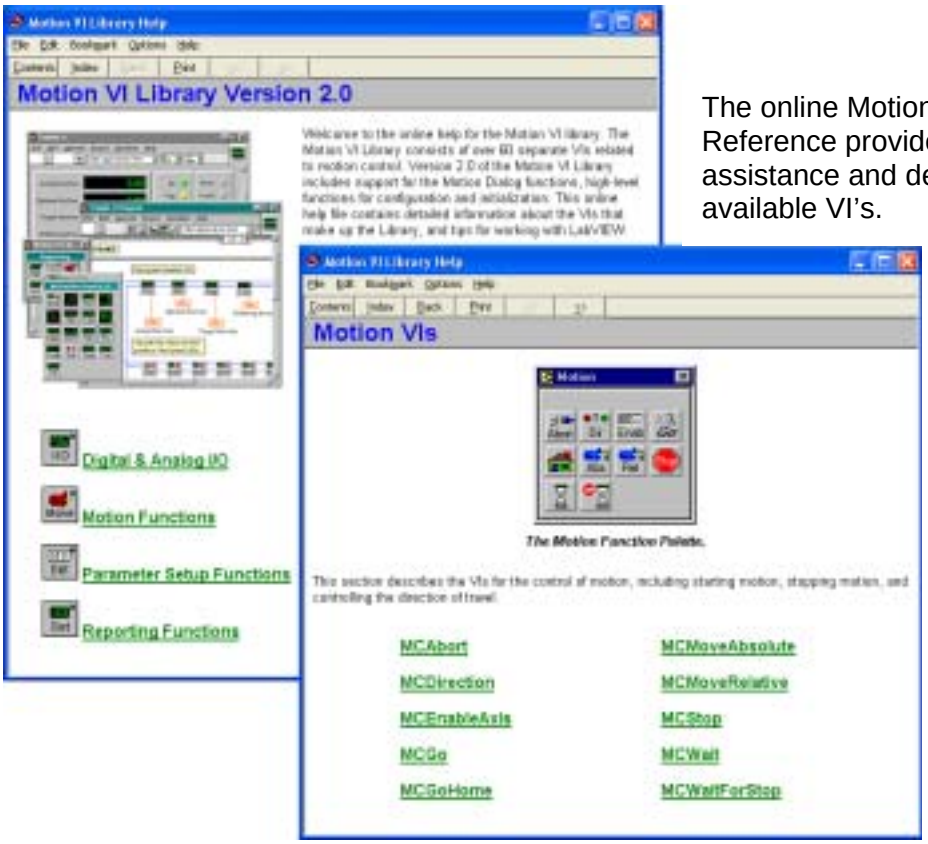


The online MCAPI Reference provides a complete listing and description of all MCAPI functions. Function calls are grouped both alphabetically and by functional groups (Motion, Setup, Reporting, Gearing, etc...). Source code examples are provided for C++, Visual Basic, and Delphi.

Mcdlg.hlp

The online MCAPI Common Dialog Reference describes the high level MCAPI Dialog functions. These operations include: Save and Restore axis configurations (PID and Trajectory), Windows Class Position and Status displays, Scaling, and I/O configuration.

Mclv.hlp



The online Motion VI Library Reference provides installation assistance and detailed descriptions of available VI's.

Chapter Contents

- Win Control and MCCL Commands

Chapter

2

Low Level Communication

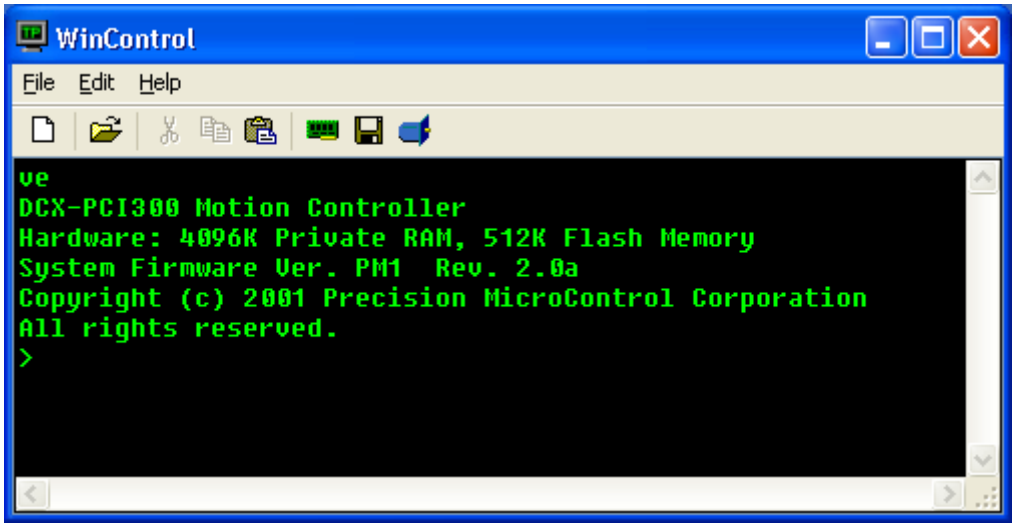
At its lowest level the operation of the motion control card is similar to that of a microprocessor, it has a predefined instruction set of operations which it can perform. This instruction set, known as Motion Control Command Language (MCCL), consists of over 200 operations which include motion, setup, conditional (if/then), mathematical, and I/O operations.

The typical PC based application will never call these low level commands directly. Instead, the programmer will call high level language MCAPI functions (in C++, Visual Basic, Delphi, or LabVIEW) which pass the appropriate native, board-level MCCL command(s) through the use of the MCAPI device driver. However, an understanding of how the low level commands work allows better command of the higher level language MCAPI functions.

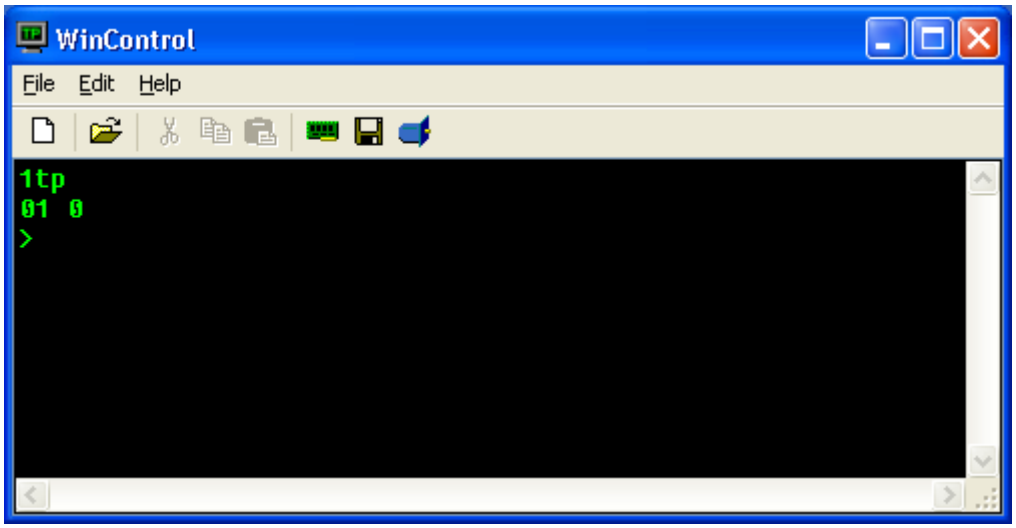
Win Control and MCCL Commands

The Win Control utility allows the user to communicate with the motion control card in its native language (MCCL). This utility communicates with the controller via the PCI ASCII interface. All MCCL commands are described in detail in the **Motion Control Command Language (MCCL) Reference Manual** specific to your controller.

MCCL commands are two character alphanumeric mnemonics built with two key characters from the description of the operation (i.e.. "MR" for **M**ove **R**elative). When the command, followed by a carriage return, is received by the motion control card, it will be executed. The following graphic shows the result of executing the VE command. This command causes the motion control card to report firmware version and the amount of installed memory.



All axis related MCCL commands will be preceded by an axis number, identifying to which axis the operation is intended. The following graphic shows the result of issuing the **T**ell **P**osition (aTP) command to axis number one.



Note that each character typed at the keyboard should be echoed to your display. If you enter an illegal character or an illegal series of valid characters, the motion control card will return a question mark character, followed by an error code. The **MCCL Error Code** listing can be found in the **Motion Control Command Language (MCCL) Reference Manual** specific to your controller. On receiving this response, you should re-enter the entire command/command string. If you make a mistake in typing, the backspace can be used to correct it. The motion control card will not begin to execute a command until a carriage return is received.

Once you are satisfied that the communication link is correctly conveying your commands and responses, you are ready to check the motor interface. When the motion control card is powered up or reset, each motor control module is automatically set to the "motor off" state. In this state, there should be no drive current to the motors. For servos it is possible for a small offset voltage to be present. This is usually too small to cause any motion, but some systems have so little friction or such high amplifier gain, that a few millivolts can cause them to drift in an objectionable manner. If this is

the case, the "null" voltage can be minimized by adjusting the offset adjustment potentiometer on the respective servo control module.

Before a motor can be successfully commanded to move certain parameters must be set by issuing commands to the motion control card. These include; PID filter gains, trajectory parameters (maximum velocity, acceleration, and deceleration), allowable following error, configuring motion limits (hard and soft).

At this point the user should refer to the Motion Control chapter and the sections that deal with Theory of Motion Control, Servo Basics and Stepper Basics in the appropriate **User's Manual** for the motion control card you are using. There the you will find more specific information for each type of motor, including which parameters must be set before a motor should be turned on and how to check the status of the axis.

Assuming that all of the required motor parameters have been defined, the axis is enabled with the **M**otor o**N** (aMN) command. Parameter 'a' of the **M**otor o**N** command allows the user to turn on a specific axis or all axes. To enable all, enter the **M**otor o**N** command with parameter 'a' = 0. To enable a single axis issue the **M**otor o**N** command where 'a' = the axis number to be enabled.

After turning a particular axis on, it should hold steady at one position without moving. The **T**ell **T**arget (aTT) and **T**ell **P**osition (aTP) commands should report the same number. There are several commands which are used to begin motion, including **M**ove **A**bsolute (MA) and **M**ove **R**elative (MR). To move axis 2 by 1000 encoder counts, enter 2MR1000 and a carriage return. If the axis is in the "**M**otor o**N**" state, it should move in the direction defined as positive for that axis. To move back to the previous position enter 2MR-1000 and a carriage return.

With the any of PMC's motion controllers, it is possible to group together several commands. This is not only useful for defining a complex motion which can be repeated by a single keystroke, but is also useful for synchronizing multiple motions. To group commands together, simply place a comma between each command, pressing the return key only after the last command.

A repeat cycle can be set up with the following compound command:

```
2MR1000,WS0.5,MR-1000,WS0.5,RP6  <return>
```

This command string will cause axis 2 to move from position 1000 to position –1000 7 times. The **R**e**P**eat (RP) command at the end causes the previous command to be repeated 6 additional times. The **W**ait for **S**top (WS) commands are required so that the motion will be completed (trajectory complete) before the return motion is started. The number 0.5 following the WS command specifies the number of seconds to wait after the axis has ceased motion to allow some time for the mechanical components to come to rest and reduce the stresses on them that could occur if the motion were reversed instantaneously. Notice that the axis number need be specified only once on a given command line.

A more complex cycle could be set up involving multiple axes. In this case, the axis that a command acts on is assumed to be the last one specified in the command string. Whenever a new command string is entered, the axis is assumed to be 0 (all) until one is specified.

Entering the following command:

```
2MR1000,3MR-500,0WS0.3,2MR1000,3MR500,0WS0.3,RP4  <return>
```

will cause axis 2 to move in the positive direction and axis 3 to move in the negative direction. When both axes have stopped moving, the WS command will cause a 0.3 second delay after which the remainder of the command line will be executed.

After going through this complex motion 5 times, it can be repeated another 5 times by simply entering a return character. All command strings are retained by the controller until some character other than a return is entered. This comes in handy for observing the position display during a move. If you enter:

```
1MR1000   <return>
1TP    <return>
(return)
(return)
(return)
(return)
```

The motion control card will respond with a succession of numbers indicating the position of the axis at that time. Many terminals have an "auto-repeat" feature which allows you to track the position of the axis by simply holding down the return key.

Another way to monitor the progress of a movement is to use the **R**e**P**eat command without a value. If you enter:

```
1MR10000  <return>
1TP,RP    <return>
```

The position will be displayed continuously. These position reports will continue until stopped by the operator pressing the Escape key.

While the motion control card is executing commands, it will ignore all alphanumeric keys that are pressed. The user can abort a currently executing command or string by pressing the escape key. If the user wishes only to pause the execution of commands, the user should press the space bar. In order to restart command execution press the space bar again. If after pausing command execution, the user decides to abort execution, this can be done by pressing the escape key.

Chapter Contents

Chapter

3

Function Library Introduction

The Motion Control Application Programming Interface (MCAPI) implements a powerful set of high level functions and data structures for programming motion control applications. Although this manual has been written for the latest version of the MCAPI software, there are still remnants of deprecated functions. The older functions will still work with this version, however, we recommend that the newer functions be migrated to when feasible.

The API is backwards compatible, and applications may use the most current version of the MCAPI for products of varying generations. Care must be taken to note the exceptions of newer features that older products might not be capable of utilizing, as well as older functions may not be relevant to new controllers. Please observe the compatibility section in each function.

Function Listing Introduction

An example of a function listing is shown below. What follows the example is a brief description of what should be found in each of the respective headings.

MCEnableAxis

MCEnableAxis( ) turns the specified axis on or off.

```
void MCEnableAxis(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   short int state                // Boolean flag for on/off setting of axis
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.

axis Axis number to turn on or off.
state Flag to indicate if this axis should be turned on or turned off:

Value	Description
TRUE	Turn on *axis*.
FALSE	Turn off *axis*.

Returns
This function does not return a value.

Comments
This function does much more than just enable or disable *axis*. However, as the name implies, the selected axis(axes) will be turned on or off depending upon the value of *state*. Note that an axis must be enabled before any motion will take place. Issuing this command with *axis* set to MC_ALL_AXES will enable or disable all axes installed on *hCtlr*.

state will accept any non-zero value as TRUE, and will work correctly with most programming languages, including those that define TRUE as a non-zero value other than one (one is the Windows default value for TRUE).

If *axis* is off and then turned on, the following events will occur.

- The target and optimal positions are set to the present encoder position.
- The offset from **MCFindEdge( )**, **MCFindIndex( )** or **MCIndexArm( )** is applied.
- The data passed by **MCSetScale( )** are applied.
- MC_STAT_AMP_ENABLE will be set.
- MC_STAT_AMP_FAULT, if present, will be cleared.
- MC_STAT_ERROR, if present, will be cleared.
- MC_STAT_FOLLOWING, if present, will be cleared.
- MC_STAT_MLIM_TRIP, if present, will be cleared.
- MC_STAT_MSOFT_TRIP, if present, will be cleared.
- MC_STAT_PLIM_TRIP, if present, will be cleared.
- MC_STAT_PSOFT_TRIP, if present, will be cleared.

If *axis* is on and then turned on again, the following events will occur.

- The offset from **MCFindEdge( )**, **MCFindIndex( )** or **MCIndexArm( )** is applied.
- The data passed by **MCSetScale( )** are applied.

Calling this function to enable or disable an axis while it is in motion is not recommended. However, should it be done, *axis* will cease the current motion profile, and MC_STAT_AT_TARGET will be set.

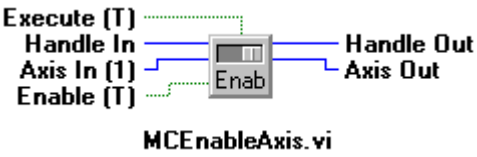
Compatibility
There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: procedure MCEnableAxis(hCtlr: HCTRLR; axis: Word; state: SmallInt); stdcall;
VB: Sub MCEnableAxis (ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal state As Integer)
LabVIEW:



MCCL Reference

MF, MN

See Also

MCAbort( ), **MCStop( )**

Each function definition begins with a brief introductory description that explains what the function is used for.

Following the description, a grey box contains the C/C++ function prototype. Here each of the parameters is listed with its type and a short description for a quick overview.

Parameters then further explains in more detail what each of the parameters means. Here a table, if applicable, will be included listing the allowable values for the preceding parameter. When values are listed, they will be given as self documenting constants. A complete listing of the self documenting constants can be found in Appendix B.

Returns describes what the function will return and explains what those values mean. The self documenting constants will be referenced when possible.

Comments describes the function in even more detail. Explanation will range from why the function is used, to how it is used, where it could cause problems and potential alternatives.

Occasionally, the following two boxes can be found in the comments section and contain relevant information that needs to be emphasized. The first box aids in the understanding of the function. The second box warns of scenarios that will more than likely cause problems.

Information to assist the programmer.

Warning to help the programmer avoid potential problems.

Compatibility gives information as to which motion control cards or modules will not work with the function. Generally, only exceptions will be listed, as to provide a more concise listing.

Requirements lists which header files, library, and the MCAPI version that must be used. Obviously, only the header file which pertains to the development environment must be used. The version of the MCAPI that is referenced is the earliest version that supports the function, so any version higher that is used will not cause a problem.

Prototypes lists the function prototypes for Delphi/Pascal, Visual Basic, and LabVIEW. As shown, each of the parameters are listed with their type. Not all functions will be available in all environments and will be noted as “Not Supported” when exceptions exist.

MCCL Reference lists the MCCL level commands that comprise the high level function. More information can be found in the **Motion Control Command Language (MCCL) Reference Manual** specific to your controller on how each of these commands works. Not all functions will be comprised of speaking to the board with MCCL commands, in which cases there will be no equivalent commands.

See Also lists related functions. Some of these functions may be alternatives to be used, while others may be the corresponding get function to a set function. Yet there will be other functions that must be used as in tandem with another function.

Motion Control API Function Quick Reference Tables

The following tables show how functions have been classified categorically. Although several functions could quite logically be listed in multiple categories, each function will appear in only one chapter, which is noted by the table’s heading. The organization follows closely to prior manuals and the online help. The grouping of functions in this manner gives a new user of the MCAPI software a chance to find similar functions in one place. For a handy quick reference printout, please refer to the **MCAPI Quick Reference Card**, which can be found on our website (www.pmccorp.com) under support and then Motion Control API. The quick reference card lists all of the following functions, as well as the data structures and the constants, in a convenient, alphabetical listing.

Data Structures

Function	Description
MCAXISCONFIG	provides basic information about the type and configuration of a single motor axis
MCCOMMUTATION	commutation parameters for an *axis*
MCCONTOUR	contains contouring parameters for an axis
MCFILTEREX	contains the PID filter parameters for a closed-loop axis
MCJOG	defines jog parameters for an axis
MCMOTIONEX	defines basic motion parameters for an axis
MCPARAMEX	provides basic information about the type and configuration of a controller, including the number of axes and modules supported
MCSCALE	defines basic scaling parameters for an axis. structure
MCSTATUSEX	defines basic status word information for an axis

Parameter Setup Functions

Function	Description
MCConfigureCompare( )	configure high-speed position compare
MCSetAcceleration( )	set Acceleration for an axis
MCSetAuxEncPos( )	set the position of the auxiliary encoder
MCSetCommutation( )	configure commutation
MCSetContourConfig( )	set contour configuration settings
MCSetDeceleration( )	set deceleration for an axis
MCSetDigitalFilter( )	configure digital filter
MCSetFilterConfigEx( )	set the PID filter parameters
MCSetGain( )	set the proportional gain for a servo axis
MCSetJogConfig( )	set jogging configuration for axis
MCSetLimits( )	configure hard and soft limits for an axis
MCSetModuleInputMode( )	configure stepper module input mode
MCSetModuleOutputMode( )	define the output type
MCSetMotionConfigEx( )	set motion parameters (velocity, accel, step rate, dead band, etc...)
MCSetOperatingMode( )	set the mode of motion (position, velocity, contour, torque)
MCSetPosition( )	set the current position of an axis
MCSetProfile( )	select a motion profile (trapezoidal, s-curve, parabolic)
MCSetRegister( )	set general purpose user register
MCSetScale( )	set the scaling factors for an axis
MCSetServoOutputPhase( )	select normal or reverse phasing for a servo axis
MCSetTorque( )	set output voltage limit for servo
MCSetVectorVelocity( )	set the vector velocity of a contoured move
MCSetVelocity( )	set the maximum velocity for a one axis move

Motion Functions

Function	Description
MCAbort( )	abort the current motion for an axis
MCArcCenter( )	sets the center point of an arc
MCArcEndAngle( )	defines the ending angle of an arc
MCArcRadius( )	defines the radius of an arc
MCCaptureData( )	initiate real time capture of position and servo loop data
MCContourDistance( )	set the path distance for user defined contour motion
MCDirection( )	set travel direction for velocity mode move
MCEdgeArm( )	arm edge input for position capture
MCEnableAxis( )	turn axis on or off
MCEnableBacklash( )	enable backlash compensation
MCEnableCapture( )	enable position capture
MCEnableCompare( )	enable position compare
MCEnableDigitalFilter( )	enable digital filter
MCEnableEncoderFault( )	enable encoder fault detection
MCEnableGearing( )	enable/disable gearing
MCEnableJog ( )	enable/disable jogging for axis
MCEnableSync( )	enables cubic spline motion, synchronizes contour motion
MCFindAuxEncIdx( )	initialize the auxiliary encoder at the location of the index
MCFindEdge( )	initialize a stepper motor at the location of the home input
MCFindIndex( )	initialize a servo motor at the location of the encoder index input
MCGoEx( )	start a velocity mode motion, begin cubic spline motion sequence
MCGoHome( )	move axis to absolute position 0
MCIndexArm( )	arms encoder index capture
MCInterruptOnPosition( )	set breakpoint reached flag of status word
MCLearnPoint( )	store position in point memory
MCMoveAbsolute( )	move axis to absolute position
MCMoveRelative( )	move axis to relative position
MCMoveToPoint( )	move to position stored in point memory
MCReset( )	perform a software reset of the controller
MCStop( )	stop motion
MCWait( )	wait for a variable time period
MCWaitForEdge( )	wait for the home input
MCWaitForIndex( )	wait for the index input to go true.
MCWaitForPosition( )	wait for axis to reach absolute position
MCWaitForRelative( )	wait for axis to reach relative position
MCWaitForStop( )	wait for the calculated trajectory to be complete
MCWaitForTarget( )	wait for axis to reach target position

Reporting Functions

Function	Description
MCDecodeStatusEx( )	axis status word decoding
MCEnableInterrupt( )	enable/disable PCI host interrupts
MCErrorNotify( )	enables/disables error messages for application window
MCGetAccelerationEx( )	get current programmed acceleration for axis
MCGetAuxEncIdxEx( )	get last observed position of auxiliary encoder index pulse
MCGetAuxEncPosEx( )	get current position of auxiliary encoder
MCGetAxisConfiguration( )	get the axis type, location, and capabilities
MCGetBreakpointEx( )	get the most recent breakpoint position
MCGetCaptureData( )	retrieve captured axis data (current position, optimal position, error)
MCGetContourConfig( )	get contour configuration settings
MCGetContouringCount( )	get current contour count
MCGetCount( )	get count parameter of various modes
MCGetDecelerationEx( )	get current programmed deceleration for axis
MCGetDigitalFilter( )	get digital filter settings
MCGetError( )	returns the most recent controller error
MCGetFilterConfigEx( )	get the PID parameters
MCGetFollowingError( )	get the current programmed following error
MCGetGain( )	get the current proportional gain setting for an axis
MCGetIndexEx( )	get the last observed position of the primary encoder index pulse
MCGetInstalledModules( )	Enumerates the type of DCX modules
MCGetJogConfig( )	get jogging configuration for axis
MCGetLimits( )	get current hard and soft limit settings
MCGetModuleInputMode( )	get the current input mode for a stepper module
MCGetMotionConfigEx( )	get motion configuration
MCGetOperatingMode( )	get the current operating mode for a motor module
MCGetOptimalEx( )	get the current optimal position of an axis
MCGetPositionEx( )	get the current position of an axis
MCGetProfile( )	get the current profile type (trapezoidal, s-curve, parabolic)
MCGetRegister( )	get the contents of a general purpose register
MCGetScale( )	get the current programmed scaling factors for an axis
MCGetServoOutputPhase( )	get the output phase (normal or reversed) of a servo
MCGetStatusEx( )	get the axis status word
MCGetTargetEx( )	get the current target of an axis
MCGetTorque( )	get the current torque setting of an axis
MCGetVectorVelocity( )	get the current programmed vector velocity of an axis
MCGetVelocityActual( )	get the current actual velocity of an axis
MCGetVelocityEx( )	get the current programmed velocity of an axis
MCIsAtTarget( )	is axis at target position?
MCIsDigitalFilter( )	is digital filter enabled?
MCIsEdgeFound( )	has edge input gone true?
MCIsIndexFound( )	has index pulse been found?
MCIsStopped( )	is axis stopped?
MCTranslateErrorEx( )	translate numeric error code to text message

I/O Functions

Function	Description
MCConfigureDigitalIO( )	configure digital I/O channels (input, output, high true, low true)
MCEnableDigitalIO( )	set the state of a digital output channel
MCGetAnalogEx( )	read analog input channel value
MCGetDigitalIO( )	get the state of a digital input channel
MCGetDigitalIOConfig( )	get digital I/O channel configuration
MCSetAnalogEx( )	set the value of an analog output
MCWaitForDigitalIO( )	wait for digital I/O channel to reach a specific state

Macro's and Multi-Tasking Functions

Function	Description
MCCancelTask( )	cancel a background task
MCMacroCall( )	call a MCCL macro
MCRepeat( )	inserts a repeat command into a macro or task sequence

MCAPI Driver Functions

Function	Description
MCBlockBegin( )	begin a compound commands (contour motion, macro's, multi-tasking)
MCBlockEnd( )	end a compound commands (contour motion, macro's, multi-tasking)
MCClose( )	close a controller (free handle)
MCGetConfigurationEx( )	obtain PMC controller hardware configuration
MCGetVersion( )	get the version of the DLL and device driver
MCOpen( )	open a controller (get handle)
MCReopen( )	re-opens existing controller handle for a new mode
MCSetTimeoutEx( )	set a timeout value for controller

OEM Low Level Functions

Function	Description
pmccmd( )	send a binary command
pmccmdex( )	send a binary command
pmcgetc( )	get ASCII character from controller
pmcgetramex( )	read directly from controller memory
pmcgets( )	get ASCII string from controller
pmcputc( )	write ASCII character to controller
pmcputramex( )	write directly to controller memory
pmcputs( )	write ASCII string to controller
pmcrdy( )	is the controller ready to accept a binary command
pmcrpy( )	read binary reply from controller
pmcrpyex( )	read binary reply from controller

Motion Dialog Functions

Function	Description
MCDLG_AboutBox( )	display a simple About dialog box
MCDLG_CommandFileExt( )	get the file extension for MCCL command files
MCDLG_ConfigureAxis( )	display a servo or stepper axis setup dialog
MCDLG_ControllerDescEx( )	get a descriptive string for a motion controller type
MCDLG_ControllerInfo( )	get configuration information about a motion controller
MCDLG_DownloadFile( )	download an ASCII command file to a motion controller
MCDLG_Initialize( )	must be called before any other MCDLG functions or classes
MCDLG_ListControllers( )	get the types of motion controllers installed
MCDLG_ModuleDescEx( )	get a descriptive string for a module
MCDLG_RestoreAxis( )	restore the settings of an axis to a previously saved state
MCDLG_RestoreDigitalIO( )	restores the settings of digital I/O channels to previously saved states
MCDLG_SaveAxis( )	save the settings of an axis to an initialization file for later use
MCDLG_SaveDigitalIO( )	save the settings of digital I/O channels to an initialization file
MCDLG_Scaling( )	display a scaling setup dialog and allow changes to scaling parameters.
MCDLG_SelectController( )	display a list of installed controllers and allow selection of a controller

Chapter Contents

Chapter

4

Data Structures

The following data structures allow the programmer to pass data to and from the controller in a simple and efficient manner. Structures are the only way, short of using MCCL, to set and get certain parameters to and from the motion control card. Functions listed in the "see also" section rely on these data structures. The chapters on Parameter Setup Functions and Reporting Functions contain the majority of the functions that require these structures.

MCAXISCONFIG

MCAXISCONFIG structure provides basic information about the type and configuration of a single motor axis.

```
typedef struct {
   long int cbSize;
   long int ModuleType;
   long int ModuleLocation;
   long int MotorType;
   long int CaptureModes;
   long int CapturePoints;
   long int CaptureAndCompare;
   double HighRate;
   double MediumRate;
   double LowRate;
   double HighStepMin;
   double HighStepMax;
   double MediumStepMin;
   double MediumStepMax;
   double LowStepMin;
   double LowStepMax;
   long int AuxEncoder;
```

```
} MCAXISCONFIG;
```

Members

cbSize Size of the **MCAXISCONFIG** data structure, in bytes.

ModuleType Array of OEM axis type specifiers, one per axis:

Value	Description
MC100	Identifies a DC Servo axis with analog signal output.
MC110	Identifies a DC Servo axis with motor output.
MC150	Identifies a stepper motor axis.
MC160	Identifies a stepper motor with encoder axis.
MC200	Identifies an Advanced Servo axis with analog signal output.
MC210	Identifies an Advanced Servo axis with PWM motor output.
MC260	Identifies an Advanced Stepper axis.
MC300	Identifies a DSP-Based Servo axis with analog signal output.
MC302	Identifies a DSP-Based Dual Servo axes with dual analog signal outputs.
MC320	Identifies a DSP-Based Brushless AC Servo axis with dual analog signal outputs.
MC360	Identifies a DSP-Based Stepper axis.
MC362	Identifies a DSP-Based Dual Stepper axes.
MF300	Identifies this axis as an RS-232 communications module. This module is not normally used with a controller installed in a PC adapter slot.
MF310	Identifies this axis as an IEEE-488 (GPIB) communications module. This module is not normally used with a controller installed in a PC adapter slot.
MC400	Identifies this axis as providing additional digital I/O channels (16).
MC500	Identifies this axis as providing additional analog channels.
DC2SERVO	Identifies the dedicated servo output of a DC2 controller.
DC2STEPPER	Identifies the optional stepper output of a DC2 controller.

MotorType Provides a simplified type identifier for the motor type (bit flags):

Value	Description
MC_TYPE_SERVO	Axis is a servo motor.
MC_TYPE_STEPPER	Axis is a stepper motor.

CaptureModes Supported data capture modes for this axis (bit flags). One or more of the following values may be OR'ed together:

Value	Description
MC_CAPTURE_ACTUAL	Axis can capture actual position data.
MC_CAPTURE_ADVANCED	Axis supports the *Delay* and *Period* settings of MCCaptureData( )
MC_CAPTURE_ERROR	Axis can capture error position data.
MC_CAPTURE_OPTIMAL	Axis can capture optimal position data.
MCCAPTURE_TORQUE	Axis can capture torque data.

CapturePoints Maximum number of data points that may be captured.
CaptureAndCompare High speed position capture and compare:

Value	Description
TRUE	Feature is supported.
FALSE	Feature isn't supported.

HighRate Servo update period, in seconds, for High Speed mode (valid only for servo modules).
MediumRate Servo update period, in seconds, for Medium Speed mode (valid only for servo modules).
LowRate Servo update period, in seconds, for Low Speed mode (valid only for servo modules).
HighStepMin Minimum step rate for High Speed mode (valid only for stepper modules).
HighStepMax Maximum step rate for High Speed mode (valid only for stepper modules).

MediumStepMin Minimum step rate for Medium Speed mode (valid only for stepper modules).
MediumStepMax Maximum step rate for Medium Speed mode (valid only for stepper modules).
LowStepMin Minimum step rate for Low Speed mode (valid only for stepper modules).
LowStepMax Maximum step rate for Low Speed mode (valid only for stepper modules).
AuxEncoder Auxiliary encoder support (added in rev. 3.4 of MCAPI).

Comments

Unlike the other MCAPI structures, the values in this structure are fixed by the hardware configuration and may not be changed.

Before you call **MCGetAxisConfiguration( )** you must set the **cbSize** member to the size of this data structure. C/C++ programmers may use **sizeof( )** , Visual Basic and Delphi programmers will find current sizes for these data structures in the appropriate MCAPI.XXX header file.

Visual Basic users please note that the value used for TRUE in the **MCAXISCONFIG** structure is the Windows standard of 1, not the Basic value of -1. Direct comparisons, such as:

```
   If (Param.CanDoScaling = True) Then
```

will fail. To get correct results use the constant WinTrue, declared in the MCAPI.BAS include file:

```
   If (Param.CanDoScaling = WinTrue) Then
```

Compatibility
There are no compatibility issues with this data structure.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Version: MCAPI 3.0 or higher

See Also
MCGetAxisConfiguration( )

MCCOMMUTATION

MCCOMMUTATION commutation parameters for an *axis*.

```
typedef struct {
   long int cbSize;
   double PhaseA;
   double PhaseB;
   long int Divisor;
   long int PreScale;
   long int Repeat;
} MCCOMMUTATION;
```

Members
cbSize	Size of the **MCCOMMUTATION** data structure, in bytes.
PhaseA	Phase A setting, in degrees.
PhaseB	Phase B setting, in degrees.
Divisor	Commutation divisor.
PreScale	Commutation prescale factor.
Repeat	Commutation repeat count.

Comments
Setting **Divisor**, **PreScale**, or **Repeat** to negative one (-1) will cause **MCSetCommutation( )** to skip setting that value.

Compatibility
The DC2, DCX-PC100, DCX-PCI100, DCX-AT100, DCX-AT200, and MFX-PCI1000 controllers do not support onboard commutation. The MC300, MC302, MC360, and the MC362 modules do not support onboard commutation.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Version: MCAPI 3.2 or higher

See Also
MCSetCommutation( )

MCCONTOUR

MCCONTOUR structure contains contouring parameters for an axis.

```
typedef struct {
   double VectorAccel;
   double VectorDecel;
   double VectorVelocity;
   double VelocityOverride;
} MCCONTOUR;
```

Members

VectorAccel Acceleration value for motion along a contour path.
VectorDecel Deceleration value for motion along a contour path.
VectorVelocity Maximum velocity for motion along a contour path.
VelocityOverride Proportional scaling factor for vector velocity, may be changed while axes are in motion.

Comments

The vector velocity parameter must be set prior to starting a contour path motion and can not be changed once the motion has begun. To change velocity on the fly, set the velocity override to a value other than 1.0. This value is used to proportionally scale the velocities.

Compatibility

The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Version: MCAPI 1.0 or higher

See Also

MCGetContourConfig( ), **MCSetContourConfig( )**

MCFILTEREX

MCFILTEREX structure contains the PID filter parameters for a servo axis, or the closed-loop parameters for a stepper axis operating in closed-loop mode. Please see the online MCAPI Reference for the **MCFILTER** structure.

```
typedef struct {
   long int cbSize;
   double Gain;
   double IntegralGain;
   double IntegrationLimit;
   long int IntegralOption;
   double DerivativeGain;
   double DerSamplePeriod;
   double FollowingError;
   double VelocityGain;
   double AccelGain;
   double DecelGain;
   double EncoderScaling;
   long UpdateRate;
} MCFILTEREX;
```

Members

cbSize Size of the **MCFILTEREX** data structure, in bytes.
Gain Proportional Gain setting of the PID loop.
IntegralGain Gain setting for the integral term of the PID loop.
IntegrationLimit Limit value for the integral term, limits the power the integral gain can use to reduce error to zero.
IntegralOption Operating mode for the integral term of the PID loop:

Value	Description
MC_INT_NORMAL	Selects the normal (always on) operation of the integral term.
MC_INT_FREEZE	Freeze the integral term while moving, re-enable after move is complete.
MC_INT_ZERO	Zero and freeze the integral term while moving, re-enable after move is complete.

DerivativeGain Gain setting for the derivative term of the PID loop.
DerSamplePeriod Time interval, in seconds, between derivative samples.
FollowingError Maximum position error, default units are encoder counts.
VelocityGain Gain setting for the feed-forward gain of the PID loop, volts per encoder count per second.
AccelGain Feed-forward acceleration gain setting.
DecelGain Feed-forward deceleration gain setting.
EncoderScaling Encoder counts per step scaling factor for closed-loop steppers (ignored for servos).

UpdateRate This parameter is used to set the feedback loop rate for servo motors and closed-loop steppers, or the maximum stepper pulse rate for open-loop stepper motor axes:

Value	Description
MC_RATE_UNKNOWN	Returned if MCAPI cannot determine the current rate.
MC_RATE_LOW	Selects the normal (always on) operation of the integral term.
MC_RATE_MEDIUM	Freeze the integral term while moving, re-enable after move is complete.
MC_RATE_HIGH	Zero and freeze the integral term while moving, re-enable after move is complete.

Comments

The servo tuning utility program offers a convenient, interactive format for determining appropriate filter settings for your servo/amplifier or closed-loop stepper.

When used with the DCX-PC100 and MC2xx series modules it is not always possible to read the **UpdateRate** parameter from the motion controller (requires recent firmware). If the MCAPI cannot read back this parameter it will return the value MC_RATE_UNKNOWN. If **UpdateRate** is set to MC_RATE_UNKNOWN and a call is made to **MCSetMotionConfigEx( )** the controller's **UpdateRate** value will not be changed.

Compatibility

VelocityGain is not supported on the DCX-PCI100 controller, MC100, MC110 modules, or closed-loop steppers. **AccelGain** is not supported on the DC2, DCX-PC100, or DCX-PCI100 controllers. **DecelGain** is not supported on the DC2, DCX-PC100, or DCX-PCI100 controllers. **EncoderScaling** is not supported on servos. **UpdateRate** is not supported on the DC2 or DCX-PCI100 controllers.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Version: MCAPI 3.2 or higher

See Also

MCGetFilterConfigEx( ), **MCSetFilterConfigEx( )**

MCJOG

MCJOG structure defines jog parameters for an axis.

```
typedef struct {
   double Acceleration;
   double MinVelocity;
   double Deadband;
   double Gain;
   double Offset;
} MCJOG;
```

Members

Acceleration Acceleration rate for use with jogging.

MinVelocity Stepper motor jog minimum velocity (this parameter has no effect for servo motors).

Deadband Deadband specifies a threshold value about the center position of the joystick below which motion of the joystick will not effect motor position. This prevents undesirable drifting of the motor due to mechanical and electrical variations in the joystick.

Gain Gain value for jogging. This parameter is effectively multiplied by the current joystick position to produce a velocity. To increase the maximum velocity, set **Gain** to a larger value. To reverse the direction of motor travel with respect to joystick direction **Gain** may be set to a negative value.

Offset Specifies the center position of the joystick, in volts.

Comments

The jog settings determine the performance of an axis when the jogging inputs are active and jogging has been enabled.

Compatibility

The DCX-PCI controllers, MFX-PCI1000 controllers, DC2 stepper axes, MC150, and MC160 modules do not support jogging.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Version: MCAPI 1.0 or higher

See Also

MCEnableJog( ), **MCGetJogConfig( )**, **MCSetJogConfig( )**

MCMOTIONEX

MCMOTIONEX structure defines basic motion parameters for an axis.

```
typedef struct {
   int cbSize;
   double Acceleration;
   double Deceleration;
   double Velocity;
   double MinVelocity;
   short int Direction;
   double Torque;
   double Deadband;
   double DeadbandDelay;
   short int StepSize;
   short int Current;
   WORD HardLimitMode;
   WORD SoftLimitMode;
   double SoftLimitLow;
   double SoftLimitHigh;
   short int EnableAmpFault;
} MCMOTIONEX;
```

Members

cbSize Size of the **MCMOTIONEX** data structure, in bytes.

Acceleration Acceleration rate for motion.

Deceleration Deceleration rate for motion.

Velocity Velocity for motion.

MinVelocity Stepper motor minimum velocity (this parameter has no effect for servo motors).

Direction Sets the direction of travel for velocity mode operation. Note that the interpretation of positive and negative will depend upon your hardware configuration:

Value	Description
MC_DIR_POSITIVE	Selects the positive travel direction.
MC_DIR_NEGATIVE	Selects the negative travel direction.

Torque Sets the maximum output torque level for servos. When a servo is operated in torque mode this value represents the continuous output level. The default output units are volts, but this may be scaled using the **Constant** member of the **MCSCALE** structure.

Deadband Sets the position dead band value.

DeadbandDelay Time limit that an axis must remain within the dead band area to qualify as "in range". If this value cannot be read back from the controller the Motion Control API function **MCGetMotionConfigEx( )** will set this value to -1. **MCSetMotionConfigEx( )** ignores this parameter if the value is equal to -1.

StepSize Sets the step size output for stepper motor operation:

Value	Description
MC_STEP_FULL	Selects full step operation.
MC_STEP_HALF	Selects half step operation.

Current Selects full or reduced current operation for stepper motors. Reduced current is typically used with stepper motors when they are stopped in a single position for an extended time to reduce motor heating.

Value	Description
MC_CURRENT_FULL	Selects full current (normal) operation.
MCCURRENT_HALF	Selects half current (idle) operation.

HardLimitMode Enables hard (physical) limit switches and selects stopping mode. One or more of the following values may be OR'ed together:

Value	Description
MC_LIMIT_LOW	Enables lower limit.
MC_LIMIT_HIGH	Enables upper limit.
MC_LIMIT_ABRUPT	Selects abrupt stopping mode when a limit is encountered.
MCLIMIT_SMOOTH	Selects smooth stopping mode when a limit is encountered.
MCLIMIT_INVERT	Inverts the polarity of the hardware limit switch inputs. This value may not be used with soft limits.

SoftLimitMode Enables soft (software) limit switches and selects stopping mode. See the description of **HardLimitMode** for details.
SoftLimitLow Sets "position" of low soft limit.
SoftLimitHigh Sets "position" of high soft limit.
EnableAmpFault Controls the amplifier fault input for servo motor axes:

Value	Description
TRUE	Enables amplifier fault input.
FALSE	Disables amplifier fault input.

Comments

All of the basic motion parameters are stored in the **MCMOTIONEX** structure. Many of these parameters also have their own Get/Set functions, to permit setting on the fly.

Compatibility

Acceleration is not supported on the DC2 stepper axes. **Deceleration** is not supported on the DCX-PCI100 controller, DC2 stepper axes, MC100, MC110, MC150, or MC160 modules. **MinVelocity** is not supported on the DCX-PCI100, DCX-PC100, or DC2 controllers. **Torque** is not supported on the DCX-PCI100 controller, MC100, or MC110 modules. **Deadband** is not supported on the DCX-PC100 controller, DC2 stepper axes, MC150, MC160, MC260, MC360, and MC362 modules. **DeadbandDelay** is not supported on the DCX-PC100 controller, DC2 stepper axes, MC150, MC160,

MC260, MC360 or MC362 modules. **StepSize** is not supported on the DC2 or DCX-PCI100 controllers. **Current** is not supported on the DC2 or DCX-PCI100 controllers. **SoftLimitMode** is not supported on the DC2 or DCX-PC100 controllers. **SoftLimitLow** is not supported on the DC2 or DCX-PC100 controllers. **SoftLimitHigh** is not supported on the DC2 or DCX-PC100 controllers. **EnableAmpFault** is not supported on the DC2 controllers.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Version: MCAPI 1.0 or higher

See Also
MCGetMotionConfigEx( ), **MCSetMotionConfigEx( )**

MCPARAMEX

MCPARAMEX structure provides basic information about the type and configuration of a controller, including the number of axes and modules supported.

```
typedef struct {
   int cbSize;
   int ID;
   int ControllerType;
   int NumberAxes;
   int MaximumAxes;
   int MaximumModules;
   int Precision;
   int DigitalIO
   int AnalogInput;
   int AnalogOutput;
   int PointStorage;
   int CanDoScaling;
   int CanDoContouring;
   int CanChangeProfile;
   int CanChangeRates;
   int SoftLimits;
   int MultiTasking;
   int AmpFault;
   double AnalogInpMin;
   double AnalogInpMax;
   long int AnalogInpRes;
   double AnalogOutMin;
   double AnalogInpMax;
   long int AnalogOutRes;
} MCPARAMEX;
```

Members
cbSize Size of the **MCPARAMEX** data structure, in bytes.
ID ID number given this controller during driver setup, permits easy translation of a controller handle back to an ID.
ControllerType OEM controller type identifier. It can be one of the following values:

Value	Description
DCXPC100	DCX series PC100 controller.
DCXAT100	DCX series AT100 controller.
DCXAT200	DCX series AT200 controller.
DC2PC100	DC2 series controller.
DC2STN	DC2 stand-alone series controller.
DCXAT300	DCX series AT300 controller.
DCXPCI300	DCX series PCI300 controller.
DCXPCI100	DCX series PCI100 controller.

NumberAxes Number of axes this controller is currently configured for.
MaximumAxes Maximum number of axes this controller supports.
MaximumModules Maximum number of modules this controller supports.
Precision Best numerical precision of controller:

Value	Description
MC_TYPE_LONG	32 bit integer precision.
MC_TYPE_DOUBLE	64 bit floating point precision.

DigitalIO Contains the number of digital IO channels installed.
AnalogInput The number of installed analog input channels.
AnalogOutput The number of analog output channels.
PointStorage Number of learned points that may be stored using **MCLearnPoint( )**
CanDoScaling Controller support for scaling (see **MCSCALE** structure) flag:

Value	Description
TRUE	Scaling is supported.
FALSE	Scaling isn't supported.

CanDoContouring Controller support for contouring (see **MCCONTOUR** structure) flag:

Value	Description
TRUE	Contouring is supported.
FALSE	Contouring not supported.

CanChangeProfile Controller can change acceleration/deceleration profile::

Value	Description
TRUE	Profile change is supported.
FALSE	Profile change not supported.

CanChangeRates Controller support for selectable rates (see **MCFILTEREX** structure) flag:

Value	Description

Value	Description
TRUE	UpdateRate changing is supported.
FALSE	UpdateRate changing isn't supported.

SoftLimits Controller supports soft limits (see **MCMOTIONEX** structure) flag:

Value	Description
TRUE	Soft Limits are supported.
FALSE	Soft Limits are not supported.

MultiTasking Controller supports multitasking flag:

Value	Description
TRUE	Multitasking is supported.
FALSE	Multitasking is not supported.

AmpFault Controller supports amplifier fault flag:

Value	Description
TRUE	Amplifier fault input is supported.
FALSE	Amplifier fault input is not supported.

AnalogInpMin Motherboard analog inputs minimum voltage (added in MCAPI ver. 3.4)
AnalogInpMax Motherboard analog inputs maximum voltage (added in MCAPI ver. 3.4)
AnalogInpRes Motherboard analog inputs resolution in bits (added in MCAPI ver. 3.4)
AnalogOutMin Motherboard analog outputs minimum voltage (added in MCAPI ver. 3.4)
AnalogOutMax Motherboard analog outputs maximum voltage (added in MCAPI ver. 3.4)
AnalogOutRes Motherboard analog outputs resolution in bits (added in MCAPI ver. 3.4)

Comments

Unlike the other MCAPI structures, the values in this structure are fixed by the hardware configuration and may not be changed. The axis type information that existed in the old **MCPARAM** structure may now be found in the **MCAXISCONFIG** structure.

Before you call **MCGetConfigurationEx( )** you must set the **cbSize** member to the size of this data structure. C/C++ programmers may use **sizeof( )**, Visual Basic and Delphi programmers will find current sizes for these data structures in the appropriate MCAPI.XXX header file.

Visual Basic users please note that the value used for TRUE in the **MCPARAMEX** structure is the Windows standard of 1, not the Basic value of -1. Direct comparisons, such as:

```
   If (Param.CanDoScaling = True) Then
```

will fail. To get correct results use the constant WinTrue, declared in the MCAPI.BAS include file:

```
   If (Param.CanDoScaling = WinTrue) Then
```

Compatibility
There are no compatibility issues with this data structure.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Version: MCAPI 3.0 or higher

See Also
MCGetConfigurationEx( )

MCSCALE

MCSCALE structure defines basic scaling parameters for an axis.

```
typedef struct {
   double Constant;
   double Offset;
   double Rate;
   double Scale;
   double Zero;
   double Time;
} MCSCALE;
```

Members
Constant This factor acts as a scale factor for servo analog outputs. By calibrating your motor/amplifier combination, it is possible to scale the output with **Constant** so that torque settings may be specified directly in ft-lbs.

Offset This offset represents an offset from a servo encoder' index pulse to a zero position.

Rate This factor acts as a multiplier for motion commands time values. The base controller time unit is the second, to convert this to minutes set **Rate** to 60.0, to convert to milliseconds rate should be set to 0.001.

Scale This scaling factor is applied to motion parameters to convert from encoder counts to real world units.

Zero Specifies that a soft zero should be located this distance from actual zero. By moving the soft zero around it is possible to have a series of position commands repeated at various spots in the range of travel without modifying the position commands. The actual zero position is not changed by this command.

Time This is the time factor for controller level wait commands. See the discussion of the **Rate** parameter above for more information on setting this value. Note that a single **Time** value is maintained per controller (i.e. **Time** is axis independent).

Comments
The scale factors provide a consistent, easy method of relating motion values to the actual physical system being controlled.

Compatibility
The DC2, and the DCX-PC100 do not support any of the aforementioned members. The DCX-PCI100 does not support **Offset** or **Constant**.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Version: MCAPI 1.0 or higher

See Also
MCGetScale( ), **MCSetScale( )**

MCSTATUSEX

MCSTATUSEX structure defines basic status word information for an axis.

```
typedef struct {
   int cbSize;
   DWORD Status;
   DWORD AuxStatus;
   DWORD ProfileStatus;
   DWORD ModeStatus;
} MCSTATUSEX;
```

Members
cbSize	Size of the **MCSTATUSEX** data structure, in bytes.
Status	Controller's primary status word.
AuxStatus	Controller's auxiliary status word.
ProfileStatus	Controller's profile status word.
ModeStatus	Controller's mode status word.

Comments
With the introduction of the MFX-PCI1000 series of motion controller it became necessary to reorganize the controller status words. The new status word interrupt feature allows the application to receive an asynchronous notification when any of the bits in the primary status word go true. Thus it was important that the primary status word of the MFX-PCI1000 contain the critical status information that an application might want to be notified of. As a result some of the commonly used but non-critical status bits were moved to other status words. The new functions **MCDecodeStatusEx( )** and **MCGetStatusEx( )** togeather with the **MCSTATUSEX** data structure allow the application programs to operate on all of the status words as though they were a single entry.

Compatibility
Only the MFX-PCI1000 series motion controller supports the **AuxStatus**, **ProfileStatus**, and **ModeStatus** members.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Version: MCAPI 3.3 or higher

See Also

MCDecodeStatusEx( ), **MCGetStatusEx( )**

Chapter Contents

Chapter

5

Parameter Setup Functions

Parameter setup functions allow the program to consistently configure the motion control card and individual modules to behave in an appropriate manner for a given application. Although trajectory parameters, PID loop gains, and end of travel limits should be set prior to commanding motion, these and other parameters may be changed during a move. However, certain parameters once passed to the card will not alter behavior until **MCEnableAxis( )** is called, which allows the specific axis to then implement several queued parameters at once in a logical and safe fashion. For first time setup, a development tool like **Motion Integrator** should be used to determine the proper tuning parameters that can be passed by the functions in this chapter.

To see examples of how the functions in this chapter are used, please refer to the online Motion Control API Reference.

MCConfigureCompare

MCConfigureCompare( ) configures an axis for high-speed position compare mode operation.

```
long int MCConfigureCompare(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* values,                // array of compare points
   long int num,                  // number of points in values array
   double inc,                    // increment between equally paced points
   long int mode,                 // output signal mode
   double period                  // output period for one shot mode
                                  // (seconds)
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to configure.

values Array of compare position values.
num Number of compare values.
inc Increment between successive compare positions when in evenly-spaced mode (see Comments, below).
mode Specifies how the controller is to signal that a compare position has been seen:

Value	Description
MC_COMPARE_DISABLE	Disables the output.
MC_COMPARE_INVERT	Inverts active level of the output – may be OR'ed together with any of the other settings for mode.
MC_COMPARE_ONESHOT	Configures the output for one-shot operation. The value for period will be used for the period of the one-shot.
MC_COMPARE_STATIC	Configures the output for static mode (see the controller documentation for details).
MC_COMPARE_TOGGLE	Configures the output to toggle between the active and inactive states each time a compare value is reached.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
Points for **MCConfigureCompare( )** may be entered in one of two ways. Discrete points, up to the number allowed by the module (typically 512) may be stored in the array *values* and passed to the controller. If the compare points are equally spaced store the beginning point in the first location of *values*, set *num* to one, and set *inc* to the per point increment. Note that *inc* is ignored if it is set equal to or less than zero, or if *num* is set to a value other than one.

The high-speed compare function signals a valid compare by way of a hardware output signal from the motor module. Use the mode flag to configure the operation of this hardware output.

Compatibility
The DC2, DCX-PC100, DCX-AT200, and DCX-PCI100 controllers do not support high-speed position compare.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.1 or higher

Prototypes
Delphi: function MCConfigureCompare(hCtlr: HCTRLR; axis: Word; values: Array of Double; num: Longint; inc: Double; mode: LongInt; period: Double): LongInt; stdcall;
VB: Function MCConfigureCompare(ByVal hCtrlr As Integer, ByVal axis As Integer, values As Double, ByVal num As Long, ByVal inc As Double, ByVal mode As Long, ByVal period As Double) As Long
LabVIEW: Not Supported

MCCL Reference
LC, NC, OC, OP

See Also
MCEnableCompare( ), **MCGetCount( )**

MCSetAcceleration

MCSetAcceleration( ) sets programmed acceleration value for the selected axis to *rate*, where *rate* is specified in the current units for *axis*.

```
void MCSetAcceleration(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double rate                    // new acceleration rate
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to change acceleration value of.
rate	New acceleration rate.

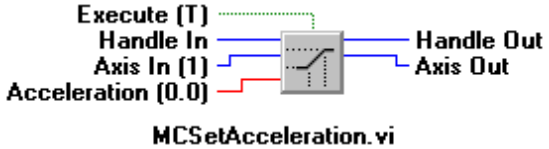
Returns
This function does not return a value.

Comments
The acceleration value for a particular *axis* may also be set using the **MCSetMotionConfigEx( )** function; **MCSetAcceleration( )** provides a short-hand method for setting just the acceleration value.

Compatibility
The DC2 stepper axes do not support ramping.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCSetAcceleration(hCtlr: HCTRLR; axis: Word; rate: Double); stdcall;
VB: Sub MCSetAcceleration Lib(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal rate As Double)
LabVIEW:



MCCL Reference
SA

See Also
MCGetAccelerationEx( ), **MCSetMotionConfigEx( )**

MCSetAuxEncPos

MCSetAuxEncPos( ) sets the current position of the auxiliary encoder.

```
void MCSetAuxEncPos(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   double position              // new position
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number of auxiliary encoder to set.
position	New encoder position.

Returns
This function does not return a value.

Comments
This command sets the current position of the auxiliary encoder to the value given by the *position* argument. A value of MC_ALL_AXES may be specified for *axis* to set the auxiliary encoders for all axes installed on a controller.

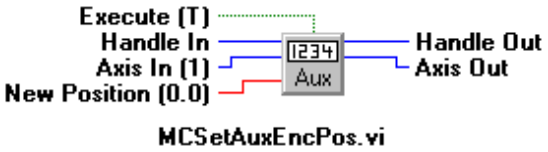
DCX-AT200 firmware version 3.5a or higher, or DCX-PC100 firmware version 4.9a or higher is required if you wish to set the position of the auxiliary encoder to a value other than zero. Earlier firmware versions ignore the value in the Position argument and zero the Auxiliary Encoder.

Compatibility
The DC2, DCX-PCI100 controllers, MC100, MC110, MC150, and MC320 modules do not support auxiliary encoders. Closed-loop steppers do not support auxiliary encoder functions, since the connected encoder is considered a primary encoder.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCSetAuxEncPos(hCtlr: HCTRLR; axis: Word; position: Double); stdcall;
VB: Sub MCSetAuxEncPos Lib(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal position As Double)

LabVIEW:



MCCL Reference
AH

See Also
MCGetAuxEncPosEx( )

MCSetCommutation

MCSetCommutation( ) sets the commutation settings for the MC320 module.

```
long int MCSetCommutation(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   MCCOMMUTATION* pCommutation    // pointer to commutation structure
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to which commutation parameters are to be set.
pCommutation	Points to an **MCCOMMUTATION** structure that contains commutation settings for *axis*.

Returns
MCSetCommutation( ) returns the value MCERR_NOERROR if the function completed without errors. If there was an error, one of the MCERR_xxxx error codes is returned.

Comments
See the section on commutation in your DCX-300 Series User's Guide for details on how to set use the commutation features of the MC320 module.

Compatibility
The DC2, DCX-PC100, DCX-PCI100, DCX-AT100, DCX-AT200, and MFX-PCI1000 controllers do not support onboard commutation. The MC300, MC302, MC360, and the MC362 modules do not support onboard commutation.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.2 or higher

Prototypes
Delphi: function MCSetCommutation(hCtlr: HCTRLR; axis: Word; var pCommutation: MCCOMMUTATION): LongInt; stdcall;

VB: Function MCSetCommutation(ByVal hCtrlr As Integer, ByVal axis As Integer, Commutation As MCCommutation) As Long
LabVIEW: Not Supported

MCCL Reference
LA, LB, LD, LE, LR

See Also
MCCOMMUTATION structure definition

MCSetContourConfig

MCSetContourConfig( ) sets contouring configuration for the specified axis.

```
short int MCConfigureDigitalIO(
   HCTRLR hCtlr,                 // controller handle
   WORD axis,                    // axis number
   MCCONTOUR* pContour           // address of contouring configuration
                                 // structure
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to set contouring configuration for.
pContour	Points to an **MCCONTOUR** structure that contains contouring configuration information for *axis*.

Returns
The return value is TRUE if the function is successful. A return value of FALSE indicates the function did not find the *axis* specified (*hCtlr* or *axis* incorrect).

Comments
Contouring configuration data should be setup prior to executing any contour motion. The field **CanDoContouring** in the **MCPARAMEX** structure will be set to TRUE, if the controller can process contour configuration data.

Compatibility
The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: function MCSetContourConfig(hCtlr: HCTRLR; axis: Word; var pContour: MCCONTOUR): SmallInt; stdcall;
VB: Function MCConfigureDigitalIO(ByVal hCtrlr As Integer, ByVal channel As Integer, ByVal mode As Integer) As Integer

LabVIEW: Not Supported

MCCL Reference
VA, VD, VO, VV

See Also
MCGetContourConfig( ), **MCCONTOUR** structure definition

MCSetDeceleration

MCSetDeceleration( ) sets programmed deceleration value for the selected axis to *rate*, where *rate* is specified in the current units for *axis*.

```
void MCSetDeceleration(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double rate                    // new deceleration rate
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to change acceleration value of.
rate	New deceleration rate.

Returns
This function does not return a value.

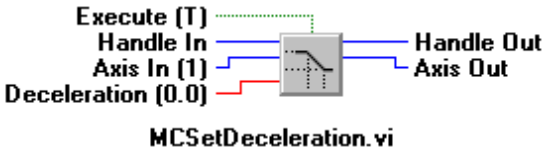
Comments
The deceleration value for a particular *axis* may also be set using the **MCSetMotionConfigEx( )** function; **MCSetDeceleration( )** provides a short-hand method for setting just the deceleration value. A value of MC_ALL_AXES may be specified for *axis* to set the deceleration for all axes installed on a controller.

Compatibility
The DCX-PCI100 controller, MC100, MC110, MC150, and MC160 modules do not support a separate deceleration value. Instead, the acceleration value will also be used as the deceleration value. The DC2 stepper axes do not support ramping.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCSetDeceleration(hCtlr: HCTRLR; axis: Word; rate: Double); stdcall;
VB: Sub MCSetDeceleration(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal rate As Double)

LabVIEW:



MCCL Reference
DS

See Also
MCGetDecelerationEx( ) , **MCSetMotionConfigEx( )**

MCSetDigitalFilter

MCSetDigitalFilter( ) sets the digital filter coefficients for the specified axis.

```
long int MCSetDigitalFilter(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* pCoeff,                // array of digital filter coefficients
   long int num                   // number of coefficients
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number.
pCoeff	Array of coefficients, must be *num* elements long (or longer). If the pointer is NULL the filter will be zeroed (overwriting any previous settings) but no new filter values will be stored.
num	Number of coefficients to retrieve, cannot be larger than the maximum digital filter size supported by the controller.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
This sets zero or more of the digital filter coefficients for the specified axis. The number of coefficients cannot exceed the maximum value supported by the axis, as reported by **MCGetCount( )**. Calling **MCSetDigitalFilter( )** overwrites any filter values previously downloaded to this axis.

Compatibility
The DC2, DCX-PC100, DCX-AT200, DCX-PCI100, MFX-PCI1000 controllers, MC360, and MC362 modules do not support digital filtering.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib

Version: MCAPI 3.1 or higher

Prototypes

Delphi: function MCSetDigitalFilter(hCtlr: HCTRLR; axis: Word; pCoeff: Array of Double; num: Longint):Longint; stdcall;

VB: Function MCSetDigitalFilter(ByVal hCtrlr As Integer, ByVal axis As Integer, coeff As Double, ByVal num As Integer) As Long

LabVIEW: Not Supported

MCCL Reference

FL, ZF

See Also

MCEnableDigitalFilter( ) , **MCGetCount( )**, **MCGetDigitalFilter( )**, **MCIsDigitalFilter( )**

MCSetFilterConfigEx

MCSetFilterConfigEx( ) configures the PID loop settings for a servo motor or the closed-loop settings for a stepper motor operating in closed-loop mode. Please see the online MCAPI Reference for the **MCSetFilterConfig( )** prototype.

```
long int MCSetFilterConfigEx(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   MCFILTEREX* pFilter          // pointer to PID filter structure
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number from which to retrieve PID information.
pFilter	Points to a **MCFILTEREX** structure that contains PID filter configuration information for *axis*.

Returns

MCSetFilterConfigEx( ) returns the value MCERR_NOERROR if the function completed without errors. If there was an error, one of the MCERR_xxxx error codes is returned.

Comments

The easiest way to change filter settings is to first call **MCGetFilterConfigEx( )** to obtain the current PID filter settings for *axis*, modify the values in the **MCFILTEREX** structure, and write the changed settings back to *axis* with **MCSetFilterConfigEx( )**.

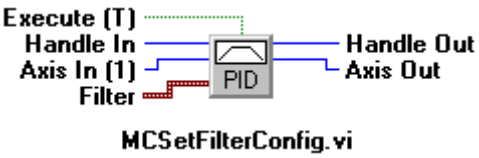
Closed-loop stepper operation requires firmware version 2.1a or higher on the DCX-PCI300 and firmware version 2.5a or higher on the DCX-AT300.

Compatibility

VelocityGain is not supported on the DCX-PCI100 controller, MC100, MC110 modules, or closed-loop steppers. **AccelGain** is not supported on the DC2, DCX-PC100, or DCX-PCI100 controllers. **DecelGain** is not supported on the DC2, DCX-PC100, or DCX-PCI100 controllers.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.2 or higher

Prototypes
Delphi: function MCSetFilterConfigEx(hCtlr: HCTRLR; axis: Word; var pFilter: MCFILTEREX): SmallInt; stdcall;
VB: Function MCSetFilterConfigEx(ByVal hCtrlr As Integer, ByVal axis As Integer, filter As MCFilterEx) As Integer
LabVIEW:



MCCL Reference
AG, DG, FR, IL, SD, SE, SI, VG

See Also
MCGetFilterConfigEx( ), **MCFILTEREX** structure definition

MCSetGain

MCSetGain( ) sets the proportional gain of a servo's feedback loop.

```
long int MCSetGain(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double gain                    // new gain setting
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to change gain of.
gain	New proportional gain.

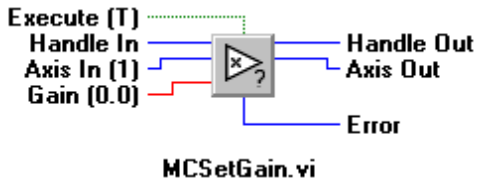
Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
The gain value for a particular *axis* may also be set using the **MCSetMotionConfigEx( )** function; **MCSetGain( )** provides a short-hand method for setting just the gain value and for updating gain settings on the fly when operating in gain mode.

Compatibility
MCSetGain( ) is not supported for open loop stepper axes.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes
Delphi: function MCSetGain(hCtlr: HCTRLR; axis: Word; gain: Double): Longint; stdcall;
VB: Function MCSetGain(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal gain As Double) As Long
LabVIEW:



MCCL Reference
SG

See Also
MCGetGain( ), **MCSetMotionConfigEx( )**

MCSetJogConfig

MCSetJogConfig( ) sets jog configuration for the specified axis.

```
short int MCSetJogConfig(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   MCJOG* pJog                    // address of jog configuration structure
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to configure jog information.
pJog	Points to a **MCJOG** structure that contains jog configuration information for *axis*.

Returns
The return value is TRUE if the function is successful. Otherwise it returns FALSE, indicating the function did not find the *axis* specified (hCtlr or *axis* incorrect).

Comments
It is important to set the jog configuration before enabling jogging if you will be using non-default parameters for the jog configuration.

Compatibility
The DCX-PCI controllers, MFX-PCI1000 controllers, DC2 stepper axes, MC150, and MC160 modules do not support jogging.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: function MCSetJogConfig(hCtlr: HCTRLR; axis: Word; var pJog: MCJOG): SmallInt; stdcall;
VB: Function MCSetJogConfig(ByVal hCtrlr As Integer, ByVal axis As Integer, jog As MCJog) As Integer
LabVIEW: Not Supported

MCCL Reference
JA, JB, JG, JO, JV

See Also
MCEnableJog( ), **MCGetJogConfig( )**, **MCJOG** structure definition

---

MCSetLimits

MCSetLimits( ) sets the current hard and soft limit settings for the specified axis.

```
long int MCSetLimits(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   short int hardMode,          // hard limit mode flags
   short int softMode,          // soft limit mode flags
   double limitMinus,           // soft negative limit value
   double limitPlus             // soft positive limit value
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to set the limits of .
hardMode Combination of the following limit mode flags for the hard limits:

Value	Description
MC_LIMIT_PLUS	Enables the positive limit.
MC_LIMIT_MINUS	Enables the negative limit.
MC_LIMIT_BOTH	Enables both the positive and negative limits.
MC_LIMIT_OFF	Sets the limit stopping mode to turn the motor off when a limit is tripped.
MC_LIMIT_ABRUPT	Sets the limit stopping mode to abrupt (target position is set to current position and PID loop stops axis as quickly as possible).
MC_LIMIT_SMOOTH	Sets the limit stopping mode to smooth (axis executes pre-programmed deceleration when limit is tripped).

Value	Description
MC_LIMIT_INVERT	Inverts the polarity of the hardware limit switch inputs. This value may not be used with soft limits.

softMode Combination of limit mode flags for the soft limits. See the values for *hardMode*, above.
limitMinus Positive limit value for soft limits, if supported by this controller.
limitPlus Negative limit value for soft limits, if supported by this controller.

Returns
MCSetLimits( ) returns the value MCERR_NOERROR if the function completed without errors. If there was an error, one of the MCERR_xxxx error codes is returned, and the limit settings will be left in an undetermined state.

Comments
The limit settings are the same as those that may be set by the **MCSetMotionConfigEx( )** function, however, this function provides a short-hand method for setting just the limit settings.

To disable limits (hard or soft) set the corresponding limit mode variable (*hardMode* and *softMode*) to zero (0). To disable a particular limit (plus or minus) DO NOT include its corresponding mode flag (MC_LIMIT_PLUS or MC_LIMIT_MINUS, respectively) in the combination of flags that make up the *hardMode* and *softMode* values.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

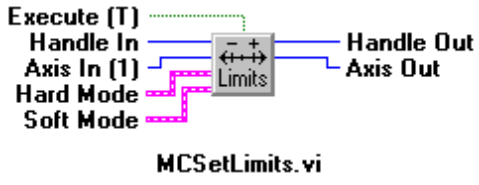
Compatibility
The DC2 and DCX-PC100 controllers do not support soft limits.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes
Delphi: function MCSetLimits(hCtlr: HCTRLR; axis: Word; hardMode, softMode: SmallInt; limitMinus, limitPlus: Double): Longint; stdcall;

VB: Function MCSetLimits(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal hardMode As Integer, ByVal SoftMode As Integer, ByVal limitMinus As Double, ByVal limitPlus As Double) As Long

LabVIEW:



MCCL Reference
HL, LF, LL, LM, LN

See Also

MCGetMotionConfigEx( ), **MCGetLimits( )**, **MCSetMotionConfigEx( )**

MCSetModuleInputMode

MCSetModuleInputMode( ) sets the current input mode for the specified axis.

```
long int MCSetModuleInputMode(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double mode                    // input mode value
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number of which to set input mode.
mode Input mode for the specified axis:

Value	Description
MC_IM_OPENLOOP	Sets stepper motor axis to open-loop mode.
MC_IM_CLOSEDLOOP	Sets stepper motor axis to closed-loop mode.

Returns

The return value is MCERR_NOERROR if no errors were detected. If there was an error, one of the MCERR_xxxx error codes is returned and the variable pointed to by *mode* is left unchanged.

Comments

You will need to issue **MCEnableAxis( )** twice, once FALSE and once TRUE, after calling this function to assure proper changing of modes.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

The DC2, DCX-PC100, DCX-PCI100, DCX-AT100, and DCX-AT200 controllers do not support a module which is capable of closed-loop stepper operation. The MC362 module is not capable of closed-loop stepper operation.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.2 or higher

Prototypes

Delphi: function MCSetModuleInputMode(hCtlr: HCTRLR; axis, mode: LongInt): LongInt; stdcall;
VB: Function MCSetModuleInputMode(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal mode As Long) As Long
LabVIEW: Not Supported

MCCL Reference

IM

See Also

MCGetModuleInputMode( )

MCSetModuleOutputMode

MCSetModuleOutputMode( ) configures the output of the specified servo or stepper axis.

```
void MCSetModuleOutputMode(
   HCTRLR hCtlr,                 // controller handle
   WORD axis,                    // axis number
   double mode                   // output mode selection
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to set output mode of.
mode Output mode, one of the following constants:

Value	Description
MC_OM_BIPOLAR	Sets servo axis to bipolar operation. (-10V to +10V)
MC_OM_UNIPOLAR	Sets servo axis to unipolar operation. (0V to +10V, with a separate direction signal)
MC_OM_PULSE_DIR	Sets stepper axis to pulse and direction output.
MC_OM_CW_CCW	Sets stepper axis to clockwise and counter-clockwise operation.

Returns

This function does not return a value.

Comments

Note that the function arguments will depend upon the type of axis being addressed - stepper or servo. Output phase settings are normally made at power up (before motors are energized) and then left unchanged. Incorrect settings can lead to unpredictable operation.

Compatibility

The DC2, DCX-PC100, DCX-PCI100 controllers, MC100, MC110, MC150, and MC160 modules do not support changing the output mode.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCSetModuleOutputMode(hCtlr: HCTRLR; axis, mode: Word); stdcall;
VB: Sub MCSetModuleOutputMode(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal mode As Integer)
LabVIEW: Not Supported

MCCL Reference
OM

See Also
MCGetServoOutputPhase( )

MCSetMotionConfigEx

MCSetMotionConfigEx( ) configures an axis for motion.

```
short int MCSetMotionConfigEx(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   MCMOTIONEX* pMotion            // address of motion configuration
                                  // structure
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to configure.
pMotion	Points to a **MCMOTIONEX** structure that contains motion configuration information for the specified axis.

Returns
The return value is TRUE if the function is successful. A return value of FALSE indicates the function could not configure the axis.

Comments
This function provides a way of setting all motion parameters for a given *axis* with a single function call using an initialized **MCMOTIONEX** structure. When you need to setup many of the parameters for an *axis* it is easier to call **MCGetMotionConfigEx( )**, update the **MCMOTIONEX** structure, and write the changes back using **MCSetMotionConfigEx( )**, rather than use a Get/Set function call for each parameter.

Note that some less often used parameters will only be accessible from this function and from **MCGetMotionConfigEx( )** - they do not have individual Get/Set functions.

Compatibility

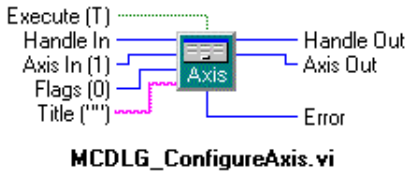
Acceleration is not supported on the DC2 stepper axes. **Deceleration** is not supported on the DCX-PCI100 controller, MC100, MC110, MC150, or MC160 modules. **MinVelocity** is not supported on the DCX-PCI100, DCX-PC100, or DC2 controllers. **Torque** is not supported on the DCX-PCI100 controller, MC100, or MC110 modules. **Deadband** is not supported on the DCX-PC100 controller, DC2 stepper axes, MC150, MC160, MC260, MC360, or MC362 modules. **DeadbandDelay** is not supported on the DCX-PC100 controller, DC2 stepper axes, MC150, MC160, MC260, MC360 or MC362 modules. **StepSize** is not supported on the DC2 or DCX-PCI100 controllers. **Current** is not supported on the DC2 or DCX-PCI100 controllers. **SoftLimitMode** is not supported on the DC2 or DCX-PC100 controllers. **SoftLimitLow** is not supported on the DC2 or DCX-PC100 controllers. **SoftLimitHigh** is not supported on the DC2 or DCX-PC100 controllers. **EnableAmpFault** is not supported on the DC2 controllers. **UpdateRate** is not supported on the DC2 or DCX-PCI100 controllers.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: function MCSetMotionConfigEx(hCtlr: HCTRLR; axis: Word; var pMotion: MCMOTIONEX): SmallInt; stdcall;
VB: Function MCSetMotionConfigEx(ByVal hCtrlr As Integer, ByVal axis As Integer, motion As MCMotionEx) As Integer
LabVIEW:



MCCL Reference

DB, DI, DT, FC, FF, FN, FR, HC, HS, LM, LS, MS, MV, SA, SD, SF, SG, SH, SI, SQ, SV

See Also

MCGetMotionConfigEx( ), **MCMOTIONEX** structure definition

MCSetOperatingMode

MCSetOperatingMode( ) sets the controller operating mode for *axis*.

```
void MCSetOperatingMode(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   WORD master,                   // master contouring axis
   WORD mode                      // new operating mode
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to configure.

master Contouring master axis (used for contour mode only).
mode New operating mode, can be any of the following:

Value	Description
MC_MODE_CONTOUR	Selects contouring mode (must also specify *master*).
MC_MODE_GAIN	Selects gain mode of operation.
MC_MODE_POSITION	Selects the position mode of operation (default).
MC_MODE_TORQUE	Selects torque mode operation.
MC_MODE_VELOCITY	Selects the velocity mode.

Returns
This function does not return a value.

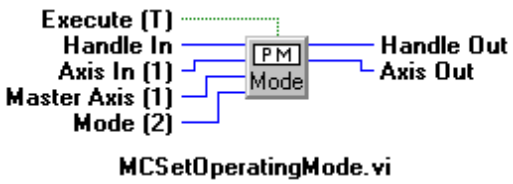
Comments
This function is used to switch between the main operating modes of the controller. All modes except MC_MODE_CONTOUR are supported by all controllers. Programs can check the field **CanDoContouring** of the **MCPARAMEX** structure for the value TRUE to determine if a controller can operate in MC_MODE_CONTOUR mode.

This function should not be called while *axis* is in motion.

Compatibility
The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers. Gain mode is not supported on stepper axes, MC100, or MC110 modules. Torque mode is not supported on stepper axes, DCX-PCI100 controller, MC100, or MC110 modules.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCSetOperatingMode(hCtlr: HCTRLR; axis, master, mode: Word); stdcall;
VB: Sub MCSetOperatingMode(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal master As Integer, ByVal mode As Integer)
LabVIEW:



MCCL Reference
CM, GM, PM, QM, VM

See Also
Controller hardware manual

MCSetPosition

MCSetPosition( ) sets the current position for *axis* to *position*.

```
void MCSetPosition(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double position                // new position
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to change position of.
position	New position value.

Returns
This function does not return a value.

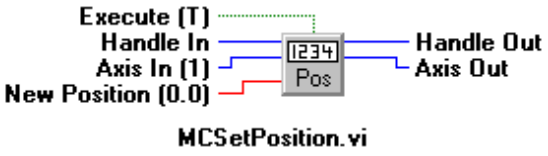
Comments
The current position of *axis* will be immediately updated to the value of *position*.

This function may be called with *axis* set to MC_ALL_AXES set the position of all axes at once. All axes will be set to the same value of *position*.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCSetPosition(hCtlr: HCTRLR; axis: Word; position: Double); stdcall;
VB: Sub MCSetPosition(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal position As Double)
LabVIEW:



MCCL Reference
DH

See Also
MCGetPositionEx( )

MCSetProfile

MCSetProfile( ) sets the velocity profile *axis*.

```
void MCSetPosition(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   WORD mode                      // new profile
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to change profile of.
position	Constant value specifying profile.

Returns
This function does not return a value.

Comments
Not all controllers can change their acceleration/deceleration profiles. The field CanChangeProfile in the MCPARAMEX data structure will be set to TRUE is the controller can change profiles.

This function may be called with *axis* set to MC_ALL_AXES to change the profile for all axes at once.

Compatibility
The DC2, DCX-PC100, and DCX-PCI100 controllers do not support S-curve or Parabolic profiles.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi:	procedure MCSetProfile(hCtlr: HCTRLR; wAxis, wMode: Word); stdcall;
VB:	Sub MCSetProfile(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal mode As Integer)
LabVIEW:	Notsupported

MCCL Reference
PP, PS, PT

See Also
MCGetConfiguration( ), MCPARAMEX

MCSetRegister

MCSetRegister( ) sets the value of the specified general purpose register.

```
long int MCSetRegister(
   HCTRLR hCtlr,                 // controller handle
   long int register,            // register number
   void* pValue,                 // pointer to variable with new register
                                 // value
   long int type                 // type of variable pointed to by pValue
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
register Register number to read from (0 to 255).
pValue Pointer to a variable that will has the new value for the register.
type Type of data pointed to by *pValue*:

Value	Description
MC_TYPE_LONG	Indicates *pValue* points to a variable of type long integer.
MC_TYPE_DOUBLE	Indicates *pValue* points to a variable of type double precision floating point.
MC_TYPE_FLOAT	Indicates *pValue* points to a variable of type single precision floating point.

Returns

The return value is MCERR_NOERROR, if no errors were detected. However, if there was an error, the return value is one of the MCERR_xxxx error codes, and the register value is unpredictable.

Comments

MCSetRegister( ) and **MCGetRegister( )** allow you to write to and read from, respectively, the general purpose registers on the motion controller. When running background tasks on a multitasking controller the only way to communicate with the background tasks is to pass parameters in the general purpose registers.

You cannot write to the local registers (registers 0 - 9) of a background task. When you need to communicate with a background task be sure to use one or more of the global registers (10 - 255).

To determine if your controller supports multi-tasking check the **MultiTasking** field of the **MCPARAMEX** structure returned by **MCGetConfigurationEx( )**.

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas

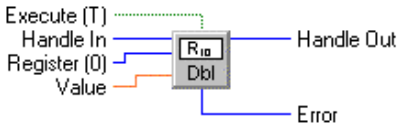
Library: use mcapi32.lib
Version: MCAPI 2.0 or higher

Prototypes

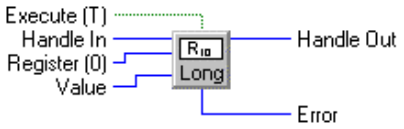
Delphi: function MCSetRegister(hCtlr: HCTRLR; register: Longint; var pValue: Pointer; type: Longint): Longint; stdcall;

VB: Function MCSetRegister(ByVal hCtrlr As Integer, ByVal register As Long, value As Any, ByVal argtype As Long) As Long

LabVIEW:



MCSetRegisterDouble.vi



MCSetRegisterLong.vi

MCCL Reference

AL, AR

See Also

MCGetRegister( )

MCSetScale

MCSetScale( ) sets scaling for the specified axis to the values contained in the **MCSCALE** structure.

```
short int MCSetScale(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   MCSCALE* pScale                // updated scaling settings
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to change scale of.
pScale	Pointer to structure with new scale values.

Returns

This function returns TRUE, if the functions completes successfully. A return value of FALSE indicates there was an error (*hCtlr* or *axis* is invalid).

Comments

Setting scaling factors allows application programs to talk to the controller in real world units, as opposed to arbitrary "encoder counts". You can determine if a controller can process scaling requests by testing the **CanDoScaling** flag in the **MCPARAMEX** structure for the controller.

This function may be called with *axis* set to MC_ALL_AXES to set the scaling of all axes at once. All axes will be set to the same value.

When **Scale** to a value other than one, **SoftLimitLow** and **SoftLimitHigh** should be changed to accommodate the new real world units.

Compatibility

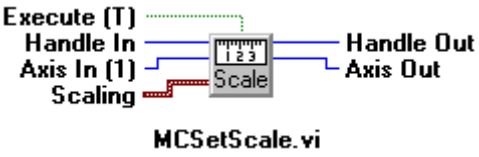
The DC2 and the DCX-PC100 do not support any scaling members. The DCX-PCI100 does not support **Offset** or **Constant**.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: function MCSetScale(hCtlr: HCTRLR; axis: Word; var pScale: MCSCALE): SmallInt; stdcall;
VB: Function MCSetScale(ByVal hCtrlr As Integer, ByVal axis As Integer, scale As MCScale) As Integer
LabVIEW:



MCCL Reference

UK, UO, UR, US, UT, UZ

See Also

MCGetConfigurationEx( ), **MCGetScale( )**, **MCPARAMEX** structure definition

MCSetServoOutputPhase

MCSetServoOutputPhase( ) sets the output phasing for the specified servo *axis*.

```
void MCSetServoOutputPhase(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   WORD phase                     // desired phasing
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to change servo phase of.
phase Desired phasing, one of the following:

Value	Description
MC_PHASE_STD	Selects standard or normal phasing. (default)
MC_PHASE_REV	Selects reverse phasing.

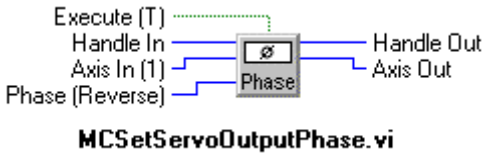
Returns
This function does not return a value.

Comments
This function may be called with *axis* set to MC_ALL_AXES set the phase of all axes at once. All axes will be set to the same value of *phase*.

Compatibility
The MC100 and MC110 modules do not support phase reverse.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCSetServoOutputPhase(hCtlr: HCTRLR; axis, phase: Word); stdcall;
VB: Sub MCSetServoOutputPhase(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal mode As Integer)
LabVIEW:



MCCL Reference
PH

See Also
MCGetServoOutputPhase( )

MCSetTorque

MCSetTorque( ) sets maximum output level for servos.

```
long int MCSetTorque(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double torque                  // new torque setting
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to change torque of.
torque	New torque.

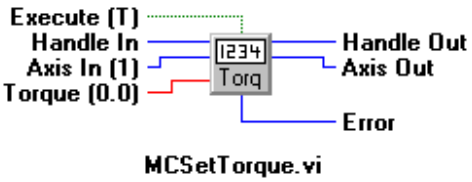
Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
The torque value for a particular *axis* may also be set using the **MCSetMotionConfigEx( )** function; **MCSetTorque( )** provides a short-hand method for setting just the torque value and for updating torque settings on the fly when operating in torque mode.

Compatibility
Torque mode is not supported on stepper axes, DCX-PCI100 controller, MC100, or MC110 modules.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes
Delphi: function MCSetTorque(hCtlr: HCTRLR; axis: Word; torque: Double): Longint; stdcall;
VB: Not Supported
LabVIEW:



MCCL Reference
SQ

See Also
MCGetTorque( ), **MCSetMotionConfigEx( )**

MCSetVectorVelocity

MCSetVectorVelocity( ) sets the vector velocity for the specified axis, in whatever units the axis is configured for.

```
long int MCSetVectorVelocity(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double velocity                // new vector velocity value
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to set vector velocity of.
velocity	New vector velocity value for the specified axis.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
The vector velocity value for a particular *axis* may also be set using **MCSetContourConfig( )**; **MCSetVectorVelocity( )** provides a short-hand method for setting just the vector velocity value and is most useful when updating vector velocity settings on the fly.

Compatibility
The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.0 or higher

Prototypes
Delphi: function MCSetVectorVelocity(hCtlr: HCTRLR; axis: Word; velocity: Double): Longint; stdcall;
VB: Function MCSetVectorVelocity(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal velocity As Double) As Long
LabVIEW: Not Supported

MCCL Reference
VV

See Also
MCGetVectorVelocity( ), **MCSetContourConfig( )**

MCSetVelocity

MCSetVelocity( ) sets programmed velocity for the selected *axis* to *rate*, where *rate* is specified in the current units for *axis*.

```
void MCSetVelocity(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double rate                    // new velocity
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to change velocity of.
rate	New velocity.

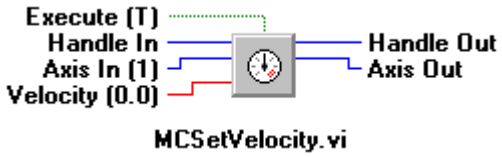
Returns
This function does not return a value.

Comments
The velocity value for a particular *axis* may also be set using the **MCSetMotionConfigEx( )** function; **MCSetVelocity( )** provides a short-hand method for setting just the velocity value and for updating velocity settings on the fly when operating in velocity mode.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCSetVelocity(hCtlr: HCTRLR; axis: Word; rate: Double); stdcall;
VB: Sub MCSetVelocity Lib(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal rate As Double)
LabVIEW:



MCCL Reference
SV

See Also
MCGetVelocityEx( ), **MCSetMotionConfigEx( )**

Chapter Contents

Chapter

6

Motion Functions

Motion functions range in use from allowing the program to commence or cease motion to permitting control of sequencing to altering operation of axes during motion.

A word of caution must be given regarding the use of board-level sequencing commands. Even though each of these functions includes a warning in this chapter, it should be stressed that once a command containing the word “Wait” or “Find” in the command name is called, the board will not accept another command nor will it respond to the calling program until the board has completed what it was initially told to do. This can lead to scenarios where the calling program has absolutely no control during potentially dangerous or otherwise expensive situations.

To see examples of how the functions in this chapter are used, please refer to the online Motion Control API Reference.

MCAbort

MCAbort( ) aborts any current motion for the specified axis or axes.

```
void MCAbort(
   HCTRLR hCtlr,                // controller handle
   WORD axis                    // axis number
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to abort motion.

Returns

This function does not return a value.

Comments

The selected *axis* will execute an emergency stop following this command. Issuing this command with *axis* set to MC_ALL_AXES will abort motion for all axes installed on the motion controller.

Servo axes will stop abruptly, and the servo control loop will remain energized.

For stepper motors, pulses from the motion controller will be disabled immediately. The state of the axis (enabled or disabled) following the call to **MCAbort( )** will depend upon the type of controller (see your controller hardware manual).

Following a call to **MCAbort( )**, verify that the axis has stopped using **MCIsStopped( )** or **MCWaitForStop( )**. Then call **MCEnableAxis( )** prior to issuing another motion command.

Following a call to **MCAbort( )** on the DCX-PC100 controller when in velocity mode, call **MCSetOperatingMode( )** prior to issuing another motion command.

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: procedure MCAbort(hCtlr: HCTRLR; axis: Word); stdcall;
VB: Sub MCAbort(ByVal hCtrlr As Integer, ByVal axis As Integer)
LabVIEW:

MCCL Reference

AB

See Also

MCEnableAxis( ), **MCSetOperatingMode( )**, **MCStop( )**, **MCIsStopped( )**, **MCWaitForStop( )**

MCArcCenter

MCArcCenter( ) specifies the center of an arc for contour path motion.

```
long int MCArcCenter(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   long int type,               // absolute or relative
   double position              // center position
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.

axis Axis number to specify arc center for.

type Flag to indicate if the center position is specified in absolute units or relative to the current position.

Value	Description
MC_ABSOLUTE	Center position is specified in absolute units.
MC_RELATIVE	Center position is specified relative to the current position of *axis*.

position Absolute or relative arc center position for *axis*.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

This function sets the center of an arc for contour path motion. Since arc motion is performed by two axes, this function should be called twice in a contour path block, once for each axis. To determine if a particular controller can process the **MCArcCenter( )** contouring function, check the **CanDoContouring** flag of the **MCPARAMEX** structure.

Compatibility

The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.0 or higher

Prototypes

Delphi: function MCArcCenter(hCtlr: HCTRLR; axis: Word; type: SmallInt; position: Double): Longint; stdcall;

VB: Function MCArcCenter (ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal arctype As Integer, ByVal position As Double) As Long

LabVIEW: Not Supported

MCCL Reference
CA, CR

See Also
MCArcEndAngle( ), **MCArcRadius( )**, **MCBlockBegin( )**, **MCSetOperatingMode( )**

MCArcEndAngle

MCArcEndAngle( ) specifies the ending angle of an arc for contour path motion.

```
long int MCArcEndAngle(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   long int type,                 // absolute or relative
   double angle                   // ending angle
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to specify arc ending angle for.
type Flag to indicate if the end angle is specified in absolute units or relative to the current position.

Value	Description
MC_ABSOLUTE	Center position is specified in absolute units.
MC_RELATIVE	Center position is specified relative to the current position of *axis*.

angle Absolute or relative arc ending angle for *axis*.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
This function sets the ending angle of an arc for contour path motion function should be called twice in a contour path block, once for each axis. To determine if a particular controller can process the **MCArcCenter( )** contouring function, check the **CanDoContouring** flag of the **MCPARAMEX** structure.

Compatibility
The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas

Library: use mcapi32.lib
Version: MCAPI 2.2 or higher

Prototypes

Delphi: function MCArcEndAngle(hCtlr: HCTRLR; axis: Word; type: SmallInt; angle: Double): Longint; stdcall;
VB: Function MCArcEndAngle (ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal arctype As Integer, ByVal angle As Double) As Long
LabVIEW: Not Supported

MCCL Reference

EA, ER

See Also

MCArcCenter( ), **MCArcRadius( )**, **MCBlockBegin( )**, **MCSetOperatingMode( )**

MCArcRadius

MCArcRadius( ) specifies the radius of an arc for contour path motion.

```
long int MCArcRadius(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   double radius                // arc radius
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to specify arc radius for.
radius	Arc radius for *axis*.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

This function sets the radius of an arc for contour path motion. To determine if a particular controller can process the **MCArcCenter( )** contouring function, check the **CanDoContouring** flag of the **MCPARAMEX** structure.

Compatibility

The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.2 or higher

Prototypes

Delphi: function MCArcRadius(hCtlr: HCTRLR; axis: Word; radius: Double): Longint; stdcall;
VB: Function MCArcRadius(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal radius As Double) As Long
LabVIEW: Not Supported

MCCL Reference

RR

See Also

MCArcCenter( ), **MCArcEndAngle( )**, **MCBlockBegin( )**, **MCSetOperatingMode( )**

MCCaptureData

MCCaptureData( ) configures a controller to perform data capture for the specified axis. Captured data includes actual position vs. time, optimal position vs. time, and following error vs. time.

```
long int MCCaptureData(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   long int points,               // number of data points to collect
   double period,                 // time period between data points
                                  // (seconds)
   double delay                   // delay prior to data capture (seconds)
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to capture data.
points	Number of data points to collect.
period	Time period between subsequent data point captures.
delay	Delay (dwell) before initial data collection.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

Captured position data is typically used to analyze servo motor performance and PID loop tuning parameters. PMC's Servo Tuning utility uses this function to analyze servo performance.

MCBlockBegin( ) may be used with **MCCaptureData( )** to bundle the capture data command with mode and move commands (see the example below).

Beginning with version 3.0 of the MCAPI users may use the **MCGetAxisConfiguration( )** function to determine the data capture capabilities of an axis.

Compatibility
The DC2 stepper axes, and the MC100, MC110, MC150, MC160 modules when installed on the DCX-PC100 controller do not support data capture. The DCX-PCI100 controller does not support torque mode nor do any stepper axes, which prevents the capture of torque values. For the DCX-AT200 period and delay are supported by MCAPI version 3.4.X or higher.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes
Delphi: function MCCaptureData(hCtlr: HCTRLR; axis: Word; points: Longint; period, delay: Double): Longint; stdcall;
VB: Function MCCaptureData(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal points As Long, ByVal period As Double, ByVal delay As Double) As Long
LabVIEW: Not Supported

MCCL Reference
PR

See Also
MCGetConfigurationEx( ), **MCGetCaptureData( )**, **MCBlockBegin( )**

MCContourDistance

MCContourDistance( ) sets the distance for user defined contour path motions.

```
long int MCContourDistance(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   double distance              // path distance
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number of controlling axis for contour motion.
distance	Path distance for user path.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
This function is used to specify the distance, as measured along the path, from the contour path starting point to the end of the next motion. It is required for user defined contour path motions.

Compatibility
The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.0 or higher

Prototypes
Delphi: function MCContourDistance(hCtlr: HCTRLR; axis: Word; distance: Double): Longint; stdcall;
VB: Function MCContourDistance(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal distance As Double) As Long
LabVIEW: Not Supported

MCCL Reference
CD

See Also
MCBlockBegin( )

MCDirection

MCDirection( ) sets the direction of motion when operating in velocity mode.

```
void MCDirection(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double dir                     // new direction
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to set the direction of.
dir New direction to move in, may be either of the following values:

Value	Description
MC_DIR_POSITIVE	Selects the positive direction for motion.
MC_DIR_NEGATIVE	Selects the negative direction for motion.

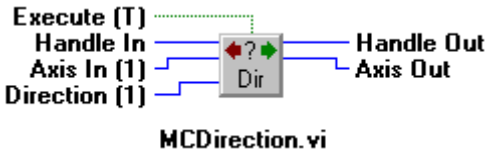
Returns
This function does not return a value.

Comments
This command may be used to change the direction of travel when an axis is operating in Velocity Mode. The actual direction of travel for MC_DIR_POSITIVE and MC_DIR_NEGATIVE will depend upon your hardware configuration.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCDirection(hCtlr: HCTRLR; axis, dir: Word); stdcall;
VB: Sub MCDirection(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal dir As Integer)
LabVIEW:



MCCL Reference
DI

See Also
MCSetOperatingMode( )

MCEdgeArm

MCEdgeArm( ) arms the edge capture function of an open-loop stepper axis.

```
long int MCEdgeArm(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double position                // new position for edge
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number for which to search for the home input signal.
position	The position where the home input signal is sensed for the axis will be properly set to *position* only after a call to **MCWaitForEdge( )** and **MCEnableAxis( )**.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
This function is used to initialize a stepper motor at a given position. The function remains pending until the home input of the module goes active. At that time you must call **MCWaitForEdge( )** followed by **MCEnableAxis( )** so that the position where the home signal is sensed will be set to the value of the *position* parameter. This function does not cause any motion to be started or stopped.

For the position where the home input signal is sensed to be set to the value of the *position* parameter, you must call **MCWaitForEdge( )** followed by **MCEnableAxis( )**. **MCIsEdgeFound( )** should be used to assure that the home input has latched prior to calling **MCWaitForEdge( )**.

Compatibility
This function is not supported by the DCX-AT200, DCX-PC, or DC2 controllers. When in closed-loop mode the MFX-PCI1000 and MC360 module do not support this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.2 or higher

Prototypes
Delphi: function MCEdgeArm(hCtlr: HCTRLR; axis: Word; position: Double): Longint; stdcall;
VB: Function MCEdgeArm(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal position As Double) As Long
LabVIEW: Not Supported

MCCL Reference
EL

See Also
MCFindEdge( ), **MCIsEdgeFound( )**, **MCWaitForEdge( )**

MCEnableAxis

MCEnableAxis( ) turns the specified axis on or off.

```
void MCEnableAxis(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   short int state                // Boolean flag for on/off setting of axis
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to turn on or off.
state Flag to indicate if this axis should be turned on or turned off:

Value	Description
TRUE	Turn on *axis*.
FALSE	Turn off *axis*.

Returns
This function does not return a value.

Comments
This function does much more than just enable or disable *axis*. However, as the name implies, the selected axis(axes) will be turned on or off depending upon the value of *state*. Note that an axis must be enabled before any motion will take place. Issuing this command with *axis* set to MC_ALL_AXES will enable or disable all axes installed on *hCtlr*.

state will accept any non-zero value as TRUE, and will work correctly with most programming languages, including those that define TRUE as a non-zero value other than one (one is the Windows default value for TRUE).

If *axis* is off and then turned on, the following events will occur.

- The target and optimal positions are set to the present encoder position.
- The offset from **MCFindEdge( )**, **MCFindIndex( )** or **MCIndexArm( )** is applied.
- The data passed by **MCSetScale( )** are applied.
- MC_STAT_AMP_ENABLE will be set.
- MC_STAT_AMP_FAULT, if present, will be cleared.
- MC_STAT_ERROR, if present, will be cleared.
- MC_STAT_FOLLOWING, if present, will be cleared.
- MC_STAT_MLIM_TRIP, if present, will be cleared.
- MC_STAT_MSOFT_TRIP, if present, will be cleared.
- MC_STAT_PLIM_TRIP, if present, will be cleared.
- MC_STAT_PSOFT_TRIP, if present, will be cleared.

If *axis* is on and then turned on again, the following events will occur.

- The offset from **MCFindEdge( )**, **MCFindIndex( )** or **MCIndexArm( )** is applied.
- The data passed by **MCSetScale( )** are applied.

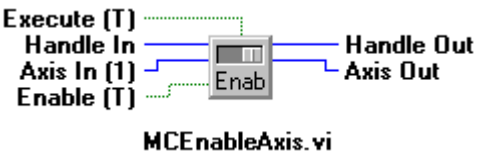
Calling this function to enable or disable an axis while it is in motion is not recommended. However, should it be done, *axis* will cease the current motion profile, and MC_STAT_AT_TARGET will be set.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCEnableAxis(hCtlr: HCTRLR; axis: Word; state: SmallInt); stdcall;
VB: Sub MCEnableAxis (ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal state As Integer)

LabVIEW:



MCCL Reference
MF, MN

See Also
MCAbort( ), **MCStop( )**

MCEnableBacklash

MCEnableBacklash( ) sets the backlash compensation distance and turns backlash compensation on or off, depending upon the value of *state*.

```
long int MCEnableBacklash(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double backlash,               // backlash compensation distance
   short int state                // enable state
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to control the backlash setting of.
backlash	Amount of backlash compensation to apply. This parameter is ignored, if *state* is FALSE.
state	Specifies whether the channel is to be turned on or turned off.

Value	Description
TRUE	Turns backlash compensation on.
FALSE	Turns backlash compensation off.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
In applications where the mechanical system is not directly connected to the motor, it may be required that the motor move an extra amount to take up gear backlash. The *backlash* parameter to this function sets the amount of this compensation, and should be equal to one half of the amount the axis must move to take up the backlash when it changes direction.

state will accept any non-zero value as TRUE, and will work correctly with most programming languages, including those that define TRUE as a non-zero value other than one (one is the Windows default value for TRUE).

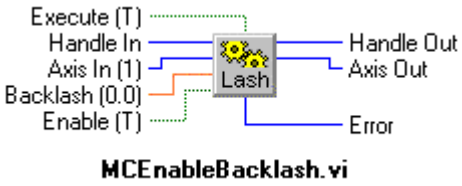
Compatibility
Stepper axes, the DC2, DCX-PC, and DCX-PCI100 controllers do not support backlash compensation.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.0 or higher

Prototypes
Delphi: function MCEnableBacklash(hCtlr: HCTRLR; axis: Word; backlash: Double; state: SmallInt): Longint; stdcall;

VB: Function MCEnableBacklash(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal backlash As Double, ByVal state As Integer) As Long

LabVIEW:



MCCL Reference
BD, BF, BN

MCEnableCapture

MCEnableCapture( ) begins position capture for the specified axis if *count* is greater than zero, or stops position capture if *count* is zero.

```
long int MCEnableCapture (
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   long int count                 // number of points to capture
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to begin or end position capture.
count	Set to zero to disable capture mode, or to a number greater than zero to capture that many positions.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
This functions enables the high-speed capture of count points (maximum 512) if count is greater than zero, or disables position capture if *count* is -1. The count of currently captured data points may be obtained using **MCGetCount( )**, and captured position values may be retrieved using **MCGetCaptureData( )**.

Compatibility
The DC2 stepper axes, and the MC100, MC110, MC150, MC160 modules when installed on the DCX-PC100 controller do not support data capture. The DCX-PCI100 controller does not support torque mode nor do any stepper axes, which prevents the capture of torque values.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.1 or higher

Prototypes
Delphi: function MCEnableCapture(hCtlr: HCTRLR; axis: Word; count: Longint): Longint; stdcall;
VB: Function MCEnableCapture(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal count As Long) As Long
LabVIEW: Not Supported

MCCL Reference
CB

See Also
MCGetCaptureData( ), **MCGetCount( )**

MCEnableCompare

MCEnableCompare( ) enables or disables high-speed compare mode for the specified axis.

```
long int MCEnableCompare(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   long int flag                  // flag to enable/disable compare state
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to enable high-speed compare.
flag Flag to indicate if this axis should be turned on or turned off:

Value	Description

Value	Description
MC_COMPARE_DISABLE	Disable high-speed compare for Axis.
MC_COMPARE_ENABLE	Enable high-speed compare for Axis.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
The high-speed compare function for *axis* is enabled or disabled by this function. High-speed compare mode must first be initialized by **MCConfigureCompare( )** before compare mode may be enabled. To determine how many compares have occurred use **MCGetCount( )**.

Compatibility
The DC2, DCX-PC100, DCX-AT200, and DCX-PCI100 controllers do not support high-speed position compare.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.1 or higher

Prototypes
Delphi: function MCEnableCompare(hCtlr: HCTRLR; axis: Word; flag: Longint): Longint; stdcall;
VB: Function MCEnableCompare(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal flag As Long) As Long
LabVIEW: Not Supported

MCCL Reference
BC

See Also
MCConfigureCompare( ), **MCGetCount( )**

MCEnableDigitalFilter

MCEnableDigitalFilter( ) enables or disables the digital filter capability of advanced motor modules, such as the MC300.

```
long int MCEnableDigitalFilter(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   long int state               // Boolean flag enables/disables digital
                                // filter
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.

axis Axis number to enable digital filter.
state Flag to indicate if digital filter should be enabled on or disabled:

Value	Description
TRUE	Enable digital filter for *axis*.
FALSE	Disable digital filter for *axis*.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
The digital filter function for *axis* is enabled or disabled by this function. Digital filter coefficients are loaded using **MCSetDigitalFilter( )** and may be read back from the controller using **MCGetDigitalFilter( )**. The function **MCIsDigitalFilter( )** will return a flag indicating the current enabled state of the digital filter, and **MCGetCount( )** may be used to determine the maximum filter size and the size of the currently loaded filter.

state will accept any non-zero value as TRUE, and will work correctly with most programming languages, including those that define TRUE as a non-zero value other than one (one is the Windows default value for TRUE).

Compatibility
The DC2, DCX-PC100, DCX-AT200, DCX-PCI100, MFX-PCI1000 controllers, MC360 and MC362 modules do not support digital filtering.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.1 or higher

Prototypes
Delphi: function MCEnableDigitalFilter(hCtlr: HCTRLR; axis: Word; state: Longint): Longint; stdcall;
VB: Function MCEnableDigitalFilter(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal state As Long) As Long
LabVIEW: Not Supported

MCCL Reference
NF, YF

See Also
MCGetCount( ), **MCGetDigitalFilter( )**, **MCIsDigitalFilter( )**, **MCSetDigitalFilter( )**

MCEnableEncoderFault

MCEnableEncoderFault( ) enables or disables encoder fault detection.

```
void MCEnableAxis(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   long int flag                  // flag to enable/disable fault detection
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Number of axis that is to have featured enabled or disabled.
flag Flags to indicate which encoders to detect faults for (or'ed togeather):

Value	Description
MC_ENC_FAULT_PRI	enable encoder fault detection for the primary encoder
MC_ENC_FAULT_AUX	enable encoder fault detection for the auxiliary encoder

Returns

This function returns MCERROR_NOERROR if there were no errors, or one of the MCERR_xxxx defined error codes if there was a problem.

Comments

Encoder fault detection must be enabled by **MCEnableEncoderFault( )** before the controller will detect and report an encoder fault.You may enable fault detection separately for the primary and the auxiliary encoder inputs, you should not enable fault detection for an encoder input that is not physically connected to an encoder (circuit noise would be interpreted as encoder failures). To disable call this function with flags set to zero.

Compatibility

Encoder fault detection is only supported on the MultiFlex family of motion controllers..

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.4 or higher

Prototypes

Delphi: procedure MCEnableEncoderFaul(hCtlr: HCTRLR; axis: Word; flag: LongInt); stdcall;
VB: Sub MCEnableEncoderFault(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal flag As Integer)
LabVIEW: Not Supported

MCCL Reference
EE

See Also
MCGetStatusEx( )

MCEnableGearing

MCEnableGearing( ) enables or disables electronic gearing for the specified *axis* / *master* pair.

```
void MCEnableGearing(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   WORD master,                 // master axis number
   double ratio,                // gearing ratio
   short int state              // enable state
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number for which to enable or disable gearing.
master	Master axis that *axis* is to follow.
ratio	Ratio at which *axis* is to reproduce *master*'s motions.
state	Specifies whether the gearing is to be enabled on or disabled.

Value	Description
TRUE	Enables gearing.
FALSE	Disables gearing.

Returns
This function does not return a value.

Comments
This function permits you to configure one axis to automatically reproduce the motions of a master axis. In addition, by using a ratio of other than 1.0, the reproduced motion can be scaled as desired.

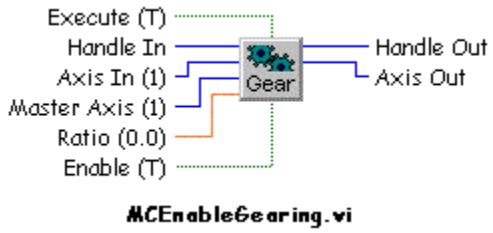
DC2 users should express the ratio as a floating point value (i.e. 0.5 for 2:1, 2.0 for 1:2, etc.). **MCEnableGearing( )** automatically converts this ratio to the 32 bit fixed point fraction the DC2 requires. The DCX-PC100 controller supports only a fixed ration of 1:1, the Ratio parameter is ignored for this controller.

state will accept any non-zero value as TRUE, and will work correctly with most programming languages, including those that define TRUE as a non-zero value other than one (one is the Windows default value for TRUE).

Compatibility
The DCX-PCI100 controller, DC2 stepper axes, the MC150, MC160, MC200, and MC260 modules when placed on the DCX-PC100 controller do not support gearing.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCEnableGearing(hCtlr: HCTRLR; axis, master: Word; ratio: Double; state: SmallInt); stdcall;
VB: Sub MCEnableGearing(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal master As Integer, ByVal ratio As Double, ByVal state As Integer)
LabVIEW:



MCCL Reference
SM, SS

MCEnableJog

MCEnableJog( ) function enables or disables jogging for the axis specified by *axis*.

```
void MCEnableJog(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   short int state                // enable state
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number for which to enable or disable synchronized motion.
state Specifies whether the synchronized motion is to be enabled on or disabled.

Value	Description
TRUE	Enables synchronized motion.
FALSE	Disables synchronized motion.

Returns
This function does not return a value.

Comments

The selected *axis* should be configured for jogging using the **MCSetJogConfig( )** function before being enabled by this function.

state will accept any non-zero value as TRUE, and will work correctly with most programming languages, including those that define TRUE as a non-zero value other than one (one is the Windows default value for TRUE).

Compatibility

The DCX-PCI controllers, MFX-PCI1000 controllers, DC2 stepper axes, MC150, and MC160 modules do not support jogging.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: procedure MCEnableJog(hCtlr: HCTRLR; axis: Word; state: SmallInt); stdcall;
VB: Sub MCEnableJog(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal state As Integer)
LabVIEW: Not Supported

MCCL Reference

JF, JN

See Also

MCGetJogConfig( ), **MCSetJogConfig( )**

MCEnableSync

MCEnableSync( ) enables or disables synchronized motion for contour path motion for the specified axis.

```
void MCEnableSync(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   short int state                // enable state
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number for which to enable or disable synchronized motion.
state Specifies whether the synchronized motion is to be enabled on or disabled.

Value	Description

Value	Description
TRUE	Enables synchronized motion.
FALSE	Disables synchronized motion.

Returns
This function does not return a value.

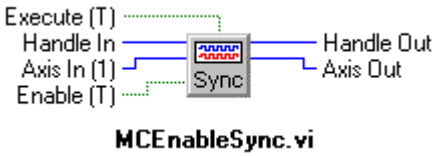
Comments
This function is issued to the controlling axis of a contour path motion, prior to issuing any contour path motions, to inhibit any motion until a call to **MCGoEx( )** is made.

state will accept any non-zero value as TRUE, and will work correctly with most programming languages, including those that define TRUE as a non-zero value other than one (one is the Windows default value for TRUE).

Compatibility
The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCEnableSync(hCtlr: HCTRLR; axis: Word; state: SmallInt); stdcall;
VB: Sub MCEnableSync(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal state As Integer)
LabVIEW:



MCCL Reference
NS, SN

See Also
MCGoEx( )

MCFindAuxEncIdx

MCFindAuxEncIdx( ) arms the auxiliary encoder index capture function of an axis.

```
long int MCFindAuxEncIdx(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double position                // reserved for future use
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number for which to search for the index signal.
position	This parameter is ignored by current motion controller firmware.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

This function arms the auxiliary encoder index capture function of an axis. The function remains pending until the auxiliary encoder index input of the module goes active, at which point, MC_STAT_INP_AUX will be latched. This function does not cause any motion to be started or stopped.

A homing routine may incorporate this function by using **MCDecodeStatusEx**( ) to determine when MC_STAT_INP_AUX latches. After making sure the axis has stopped, you may determine how far the current position is from where the auxiliary encoder index occurred. The difference between **MCGetAuxEncPosEx**( ) and **MCGetAuxEncIdxEx( )** should be used as the current position through a call to **MCSetAuxEncPos( )**.

At this time, the firmware does not support the *position* parameter. We advise you set *position* to zero, so that future firmware updates will not break your code.

Compatibility

The DC2, DCX-PCI100 controllers, MC100, MC110, MC150, and MC320 modules do not support auxiliary encoders. Closed-loop steppers do not support auxiliary encoder functions, since the connected encoder is considered a primary encoder.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.2 or higher

Prototypes

Delphi: function MCFindAuxEncIdx(hCtlr: HCTRLR; axis: Word; position: Double): Longint; stdcall;
VB: Function MCFindAuxEncIdx(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal position As Double) As Long
LabVIEW: Not Supported

MCCL Reference

AF

See Also

MCBlockBegin( ), **MCFindIndex( )**, **MCGetAuxEncIdxEx( )**

MCFindEdge

MCFindEdge( ) is used to initialize a motor at a given position, relative to the home or coarse home input.

```
long int MCFindEdge (
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double position                // new position for edge
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number for which to search for the edge signal.
position	The position where the edge signal is sensed for the axis will be set to *position* after a call to **MCEnableAxis( )**.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

This function is used to initialize a motor at a given position. The function remains pending until the home input of the module goes active. This function does not cause any motion to be started or stopped. See the example code in the online help for details of how to use **MCFindEdge( )**.

Once this command is issued, the calling program will not be able to communicate with the board until the home input is seen as high for *axis*. We recommend using **MCEdgeArm( )** and **MCIsEdgeFound( )** instead.

Only after an **MCEnableAxis( )** call will the position where the home input was seen as high for *axis* be set to the value of the *position* parameter.

The DC2 controllers, MC100, MC110, and MC260 modules use coarse home instead of home, but this still translates to MC_STAT_INP_HOME. In these cases, **MCDecodeStatusEx( )** should be used instead of this function.

Compatibility
The DC2 stepper axes, MC200 and MC210 when installed on the DCX-AT200, MC300, MC302, and MC320 modules do not support this command.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.0 or higher

Prototypes
Delphi: function MCFindEdge(hCtlr: HCTRLR; axis: Word; position: Double): Longint; stdcall;
VB: Function MCFindEdge Lib(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal position As Double) As Long
LabVIEW: Not Supported

MCCL Reference
FE

See Also
MCBlockBegin( ), **MCEdgeArm( )**, **MCFindIndex( )**, **MCIsEdgeFound( )**, **MCWaitForEdge( )**

MCFindIndex

MCFindIndex( ) is used to initialize a servo or closed-loop stepper motor at a given position, relative to the index input.

```
long int MCFindIndex(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   double position              // new position for index
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number for which to search for the index signal.
position The position where the encoder index pulse occurred for the axis will be set to *position* after a call to **MCEnableAxis( )**.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

This function is used to initialize a servo motor at a given position. The function remains pending until the index input of the module goes active. This function does not cause any motion to be started or stopped. See the example code in the online help for details of how to use **MCFindIndex( )**.

Once this command is issued, the calling program will not be able to communicate with the board until the *axis* captures the encoder index. We recommend instead using and confirming that **MCIndexArm( )** has captured the index through **MCIsIndexFound( )** before calling **MCWaitForIndex( )** to avoid this problem.

Only after an **MCEnableAxis( )** call will the position where the encoder index pulse occurred for *axis* be set to the value of the *position* parameter.

Compatibility

Open-loop stepper axes do not support this command, since the connected encoder is considered an auxiliary encoder.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.0 or higher

Prototypes

Delphi: function MCFindIndex(hCtlr: HCTRLR; axis: Word; position: Double): Longint; stdcall;
VB: Function MCFindIndex(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal position As Double) As Long
LabVIEW: Not Supported

MCCL Reference

FI

See Also

MCBlockBegin( ), **MCFindAuxEncIdx( )**, **MCFindEdge( )**, **MCIndexArm( )**, **MCWaitForEdge( )**, **MCWaitForIndex( )**

MCGoEx

MCGoEx( ) initiates a motion when operating in velocity mode.

```
long int MCGoEx(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double param                   // optional argument for the GO command
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to command.
param	Argument to the GO command.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

The axis must be configured for velocity mode operation before issuing a **MCGoEx( )** call. All axes may be instructed to move by setting the Axis parameter to MC_ALL_AXES.

To enable cubic splining while in contour mode on the DCX-AT200 or DCX-AT300 use **MCGoEx( )** with the value of *param* set to 1.0.

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes

Delphi: function MCGoEx(hCtlr: HCTRLR; axis: Word; param: Double): Longint; stdcall;
VB: Function MCGoEx(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal param As Double) As Long
LabVIEW:

MCCL Reference

GO

See Also

MCSetOperatingMode( ), **MCStop( )**

MCGoHome

MCGoHome( ) initiates a home motion for the specified axis or all axes.

```
void MCGoHome(
   HCTRLR hCtlr,                // controller handle
   WORD axis                    // axis number
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to command.

Returns

This function does not return a value.

Comments

The home or zero position is used that was last set by calling **MCSetPosition( )**. This command effectively executes a **MCMoveAbsolute( )** with a target position of 0.0.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi:
```
procedure MCGoHome( hCtlr: HCTRLR; axis: Word ); stdcall;
```
VB:
```
Sub MCGoHome Lib(ByVal hCtrlr As Integer, ByVal axis As Integer)
```
LabVIEW:

MCCL Reference

GH

See Also

MCMoveAbsolute( ), **MCSetPosition( )**

MCIndexArm

MCIndexArm( ) arms the index capture function of a servo or closed-loop stepper axis.

```
long int MCIndexArm(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double position                // new position for index
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number for which to search for the index signal.
position	The position where the encoder index pulse occurred for the axis will be set to *position* after a call to **MCEnableAxis( )**.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

This function is used to initialize a servo motor to a specified position where the encoder index pulse occurs. The function remains pending until the encoder index input of the module goes active, after which a call to **MCEnableAxis( )** sets the position where the encoder index pulse occurred to the value of the *position* parameter. This function does not cause any motion to be started or stopped.

For stepper axes this function performs in a similar fashion. The difference is that the stepper axis uses the home input signal in place of the encoder index input signal.

Only after an **MCEnableAxis( )** call will the position where the encoder index pulse occurred for *axis* be set to the value of the *position* parameter.

Compatibility

Open-loop stepper axes do not support this command, since the connected encoder is considered an auxiliary encoder.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.2 or higher

Prototypes

Delphi: function MCIndexArm(hCtlr: HCTRLR; axis: Word; position: Double): Longint; stdcall;
VB: Function MCIndexArm(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal position As Double) As Long
LabVIEW: Not Supported

MCCL Reference
IA

See Also
MCBlockBegin( ), **MCFindAuxEncIdx( )**, **MCFindIndex( )**, **MCWaitForIndex( )**

MCInterruptOnPosition

MCInterruptOnPosition( ) enables the breakpoint reached flag of the controller status word.

```
long int MCInterruptOnPosition(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   long int mode                // absolute / relative
   double position              // interrupt position
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to specify interrupt for
mode Flag to indicate if the interrupt position is specified in absolute units or relative to the current position

Value	Description
TRUE	Turn on *axis*.
FALSE	Turn off *axis*.

position Absolute or relative interrupt position for *axis*.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
This function configures an axis to set the breakpoint reached bit in the status word when an absolute or relative position is reached. By enabling status word interrupts from the controller with the **MCEnableInterrupt( )** the application program can be interrupted when the specified position is reached.

Compatibility
Only the MFX-PCI1000 series of motion controllers support status word interrupts.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.3 or higher

Prototypes

Delphi: function MCInterruptOnPosition(hCtlr: HCTRLR; axis: Word; mode: Long int, position: Double): Longint; stdcall;

VB: Function MCInterruptOnPosition(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal mode As Long int, ByVal position As Double) As Long

LabVIEW: Not Supported

MCCL Reference

IP, IR

See Also

MCEnableInterrupt( )

MCLearnPoint

MCLearnPoint( ) stores the current actual position or target position for the specified *axis* in point memory at location specified by *index*.

```
long int MCLearnPoint(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   WORD index,                  // point memory index
   WORD mode                    // type of position to store
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.

axis Axis number to store data for.

index Storage location for point data.

mode Determines if the actual position or the target position will be stored:

Value	Description
MC_LRN_POSITION	Learns the current actual position for the specified axis.
MC_LRN_TARGET	Learns the current target position for the specified axis.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

The actual position of an axis may be stored as it is moved; or, by disabling the axis, position commands may be issued to the axis, and the target positions stored, without actually moving the axis (see online help examples).

The number of points that may be stored will vary with the number of motor axes installed and the type of controller (see the compatibility section, below, for controller dependent limits). The first storage is location zero (not location 1).

The current position of all axes may be stored by setting the Axis parameter to MC_ALL_AXES.

Compatibility

The number of points that can be stored is dependent on the controller type and in some cases on the number of installed axes:

Controller	1	2	3	4	5	6	7	8
DCX-PCI300	256	256	256	256	256	256	256	256
MFX-PCI1000	256	256	256	256	256	256	256	256
DCX-PCI100	256	256	256	256	256	256	256	256
DCX-AT300	1536	768	512	384	307	256	n/a	n/a
DCX-AT200	1536	768	512	384	307	256	n/a	n/a
DCX-PC100	4096	2048	1365	1024	819	682	585	512
DC2-PC100	n/a	2048	n/a	n/a	n/a	n/a	n/a	n/a
DCX-PCI300	256	256	256	256	256	256	256	256
DCX-PCI100	256	256	256	256	256	256	256	256
DCX-AT300	1536	768	512	384	307	256	n/a	n/a
DCX-AT200	1536	768	512	384	307	256	n/a	n/a
DCX-PC100	4096	2048	1365	1024	819	682	585	512

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: function MCLearnPoint(hCtlr: HCTRLR; axis: Word; index: Longint; mode: Word): Longint; stdcall;
VB: Function MCLearnPoint Lib(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal index As Long, ByVal mode As Integer) As Long
LabVIEW: Not Supported

MCCL Reference

LP, LT

See Also

MCMoveToPoint( )

MCMoveAbsolute

MCMoveAbsolute( ) initiates an absolute position move for the specified axis.

```
void MCMoveAbsolute(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double position                // new absolute position
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to move.
position	Absolute position to move to.

Returns

This function does not return a value.

Comments

The axis must be enabled prior to executing a move (an exception to this is when the **MCMoveAbsolute( )** is used with **MCLearnPoint( )** in target mode).

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

There are no compatibility issues with this function.

Requirements

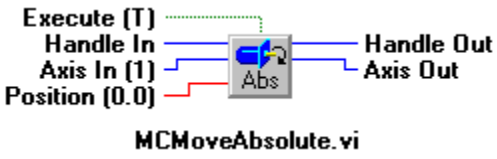
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: procedure MCMoveAbsolute(hCtlr: HCTRLR; axis: Word; position: Double); stdcall;

VB: Sub MCMoveAbsolute(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal position As Double)

LabVIEW:



MCCL Reference

MA

See Also
MCMoveRelative( ), **MCSetPosition( )**

MCMoveRelative

MCMoveRelative( ) initiates a relative position move for the specified axis or all axes.

```
void MCMoveRelative(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double distance                // distance to move from current position
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to move.
distance	Amount of distance to move.

Returns
This function does not return a value.

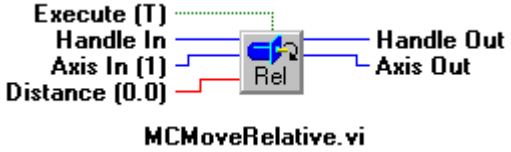
Comments
The axis must be enabled prior to executing a move (an exception to this is when the **MCMoveRelative( )** is used with **MCLearnPoint( )** in target mode).

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCMoveRelative(hCtlr: HCTRLR; axis: Word; distance: Double); stdcall;
VB: Sub MCMoveRelative(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal distance As Double)
LabVIEW:



MCCL Reference
MR

See Also
MCMoveAbsolute( ), **MCSetPosition( )**

MCMoveToPoint

MCMoveToPoint( ) initiates an absolute move to a stored location for the specified axis or all axes.

```
long int MCMoveToPoint(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   WORD index                   // index of point to move to
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to move.
index	Index of stored location to move to.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
The motor must be enabled prior to executing a **MCMoveToPoint( )** and the point specified by *index* must have been stored by a previous call to **MCLearnPoint( )**. All axes may be instructed to move by setting the *axis* parameter to MC_ALL_AXES.

Compatibility
The DC2 stepper axes do not support this command.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi:	function MCMoveToPoint(hCtlr: HCTRLR; axis: Word; index: Longint): Longint; stdcall;
VB:	Function MCMoveToPoint Lib(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal index As Long) As Long
LabVIEW:	Not Supported

MCCL Reference
MP

See Also
MCLearnPoint( )

MCReset

MCReset( ) performs a complete reset of the axis or controller, leaving the specified axis (or axes) in the disabled state.

```
void MCReset(
   HCTRLR hCtlr,                  // controller handle
   WORD axis                      // axis number
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to reset.

Returns
This function does not return a value.

Comments
Setting the *axis* parameter to MC_ALL_AXES will cause the specified controller to be reset.

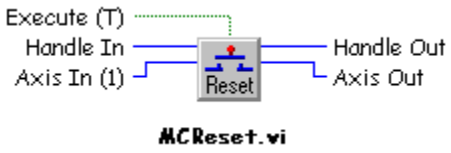
If you have enabled the hardware reset feature of the DCX-AT, or DCX-PC100 controllers **MCReset( )** will perform a hard reset when *axis* is equal to MC_ALL_AXES, or a soft reset when Axis specifies a particular axis. If this feature is off (the default state), **MCReset( )** issues the “RT” command to the board to perform any reset (this is a "soft" reset). On the DCX-AT200 and DCX-AT300 you must set jumper JP2 to connect pins 1 and 2 if Hard Reset is enabled, or connect pins 5 and 6 (factory default) if Hard Reset is disabled. On the DCX-PC100 you must set jumper JP4 to connect pins 1 and 2 if Hard Reset is enabled, or connect pins 5 and 6 (factory default) if Hard Reset is disabled. See the Motion Control Panel online help for how to enable the MCAPI Hardware Reset feature.

Compatibility
The DC2 series, DCX-PC100, DCX-AT100, and DCX-AT200 (prior to firmware version 1.2a) controllers do not support the resetting of individual axes. In these cases when this command is executed, the *axis* parameter is ignored and a controller reset is performed.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCReset(hCtlr: HCTRLR; axis: Word); stdcall;
VB: Sub MCReset Lib(ByVal hCtrlr As Integer, ByVal axis As Integer)

LabVIEW:



MCCL Reference
RT

See Also
MCAbort( ), **MCStop( )**

MCStop

MCStop( ) stops the specified axis or axes using the pre-programmed deceleration values.

```
void MCStop(
   HCTRLR hCtlr,                  // controller handle
   WORD axis                      // axis number
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to stop.

Returns
This function does not return a value.

Comments
This function initiates a controlled axis stop, as compared with **MCAbort( )** which stops the axis abruptly.

Following a call to **MCStop( )** verify that the axis has stopped using or **MCIsStopped( )** or **MCWaitForStop( )**. Then call **MCEnableAxis( )** prior to issuing another motion command.

Following a call to **MCStop( )** on the DCX-PC100 controller when in velocity mode, call **MCSetOperatingMode( )** prior to issuing another motion command.

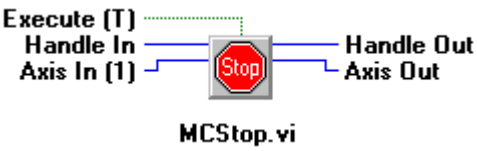
Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib

Version: MCAPI 1.0 or higher

Prototypes

Delphi: procedure MCStop(hCtlr: HCTRLR; axis: Word); stdcall;
VB: Sub MCStop(ByVal hCtrlr As Integer, ByVal axis As Integer)
LabVIEW:



MCCL Reference

ST

See Also

MCAbort( ), MCEnableAxis( ), **MCIsStopped( )**, **MCSetOperatingMode( )**, **MCWaitForStop( )**

MCWait

MCWait( ) waits the specified number of seconds before returning to the caller.

```
void MCWait(
   HCTRLR hCtlr,                  // controller handle
   double period                  // length of delay
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
period	Length of delay, in seconds.

Returns

This function does not return a value.

Comments

The delay is specified in seconds, unless **MCSetScale( )** has been called to change the time scale.

Once this command is issued, the calling program will not be able to communicate with the board until *period* elapses. We recommend creating your own time based looping structure.

Compatibility

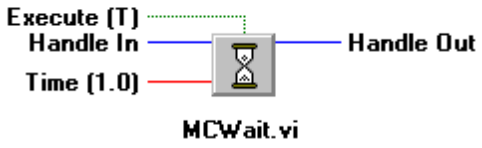
There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: procedure MCWait(hCtlr: HCTRLR; period: Double); stdcall;
VB: Sub MCWait(ByVal hCtrlr As Integer, ByVal period As Double)
LabVIEW:



MCCL Reference

WA

See Also

MCWaitForPosition( ), **MCWaitForRelative( )**, **MCWaitForStop( )**, **MCWaitForTarget( )**

MCWaitForEdge

MCWaitForEdge( ) waits for the coarse home input to go to the specified logic level for a servo, closed-loop stepper, or an MC260 open-loop stepper. When used with an open-loop stepper (excluding an MC260) this function completes a call to **MCEdgeArm( )**. Note that when used with an open-loop stepper (excluding an MC260), the parameter *state* has no effect.

```
long int MCWaitForEdge(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   short int state                // selects logic level to wait for
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to wait for.
state Selects the coarse home logic level to wait for:

Value	Description
TRUE	Wait for coarse home to go active.
FALSE	Wait for coarse home to go inactive.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

This function behaves differently depending on what type of module *axis* is and whether it is in open-loop or closed-loop mode. In both cases instruction processing is paused until the home or coarse home input, respectively, goes to the specified logic state. In open-loop mode, this function is one of three functions that must be called to set the home input signal transition to a predetermined position. In closed-loop mode, this function is used to find a home sensor to qualify an index pulse on servo or closed-loop stepper. However, using this function with a closed-loop system is discouraged.

In open-loop mode, exclusively stepper modules (excluding the MC260, see the closed-loop section for function behavior), this function should be called after **MCIsEdgeFound( )** confirms that the home input has latched from a previous call to **MCEdgeArm( )**. After this function returns control to the calling program, a call to **MCEnableAxis( )** will apply *position* defined in **MCEdgeArm( )** to the position where the home input first latched.

Once this command is issued, the calling program will not be able to communicate with the board until the home input signal is detected. We recommend calling **MCIsEdgeFound( )**, to confirm the home input is active prior to calling this function.

Note that when used with an open-loop stepper (excluding an MC260), the parameter *state* has no effect. Also, this function is only looking for an active signal state, not a transition.

When a module used in closed-loop mode or with an MC260, this function is called by itself to return when the home input state level defined by *state* is observed. To assure a leading or trailing edge, this function would have to be called twice with *state* different in both cases.

Once this command is issued, the calling program will not be able to communicate with the board until *state* matches the coarse home logic level. We recommend creating your own looping structure based on **MCDecodeStatusEx( )** and MC_STAT_INP_HOME instead of using this function.

state will accept any non-zero value as TRUE, and will work correctly with most programming languages, including those that define TRUE as a non-zero value other than one (one is the Windows default value for TRUE).

See the example code in the online help for details of how to use **MCWaitForEdge( )**.

Compatibility

The DC2 stepper axes, MC150, and MC160 modules do not support this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.0 or higher

Prototypes

Delphi: function MCWaitForEdge(hCtlr: HCTRLR; axis: Word; state: SmallInt): Longint; stdcall;
VB: Function MCWaitForEdge(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal state As Integer) As Long
LabVIEW: Not Supported

MCCL Reference
WE

See Also
MCEdgeArm( ), **MCFindEdge( )**, **MCFindIndex( )**, **MCIsEdgeFound( )**

MCWaitForIndex

MCWaitForIndex( ) waits until the index pulse has been observed on servo or closed-loop stepper axis.

```
long int MCWaitForIndex(
   HCTRLR hCtlr,                  // controller handle
   WORD axis                      // axis number
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to wait for.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
This function is used to initialize a motor to a given position relative to the index pulse. When called after **MCIndexArm( )**, it provides the exact same functionality as **MCFindIndex( )**. The benefit is that you may query the controller through **MCIsIndexFound( )** to see that the index has latched. Once the index has been seen, a call to **MCWaitForIndex( )** will not cause the board to stop communicating where **MCFindIndex( )** has the potential to cause the controller to stop communicating.

Once this command is issued, the calling program will not be able to communicate with the board until *axis* captures the encoder index. We recommend confirming that **MCIndexArm( )** has captured the index by using **MCIsIndexFound( )** before calling **MCWaitForIndex( )** to avoid this problem.

Only after an **MCEnableAxis( )** call will the position where the encoder index pulse occurred for *axis* be set to the value of the *position* parameter.

Compatibility
Open-loop stepper axes do not support this command, since the connected encoder is considered an auxiliary encoder.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.2 or higher

Prototypes
Delphi: function MCWaitForIndex(hCtlr: HCTRLR; axis: Word): Longint; stdcall;
VB: Function MCWaitForIndex(ByVal hCtrlr As Integer, ByVal axis As Integer) As Long
LabVIEW: Not Supported

MCCL Reference
WI

See Also
MCFindAuxEncIdx( ), **MCFindEdge( )**, **MCFindIndex( )**, **MCIndexArm( )**, **MCIsIndexFound( )**

MCWaitForPosition

MCWaitForPosition( ) waits for the *axis* to reach the specified *position* before allowing the next command to execute.

```
void MCWaitForPosition(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double position                // position to wait for
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to wait on to reach specified position.
position	Absolute position to wait for.

Returns
This function does not return a value.

Comments
You must start the specified *axis* moving, and make certain the motion will at least reach the wait position, in order for this function to return to the calling program.

Once this command is issued, the calling program will not be able to communicate with the board until *axis*' encoder reaches *position*.

Compatibility
The DC2 stepper axes, MC150, and MC160 modules do not support this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCWaitForPosition(hCtlr: HCTRLR; axis: Word; position: Double); stdcall;
VB: Sub MCWaitForPosition(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal position As Double)
LabVIEW: Not Supported

MCCL Reference
WP

See Also
MCWait( ), **MCWaitForRelative( )**, **MCWaitForStop( )**, **MCWaitForTarget( )**

MCWaitForRelative

MCWaitForRelative( ) waits for the *axis* to reach a position that is specified relative to the target position.

```
void MCWaitForRelative(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double distance                // relative position to wait for
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to wait on for to reach specified position.
distance	Position, relative to the current target position, to wait for.

Returns
This function does not return a value.

Comments
You must start the specified *axis* moving, and make certain the motion will at least reach the wait position, in order for this function to return to the calling program. The position argument is specified as a distance from the target position.

Once this command is issued, the calling program will not be able to communicate with the board until *axis*' encoder traverses *distance*.

Compatibility
The DC2 stepper axes, MC150, and MC160 modules do not support this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCWaitForRelative(hCtlr: HCTRLR; axis: Word; distance: Double); stdcall;
VB: Sub MCWaitForRelative(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal distance As Double)
LabVIEW: Not Supported

MCCL Reference
WR

See Also
MCWait( ), **MCWaitForPosition( )**, **MCWaitForStop( )**, **MCWaitForTarget( )**

MCWaitForStop

MCWaitForStop( ) waits for the specified *axis* or all axes to come to a stop. An optional dwell after the stop may be specified within this command to allow the mechanical system to come to rest.

```
void MCWaitForStop(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double dwell                   // dwell time after stop
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number function is waiting for to stop.
dwell	Delay time after stop has occurred.

Returns
This function does not return a value.

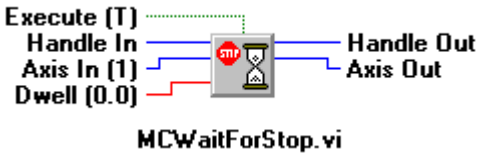
Comments
MCWaitForStop( ) is necessary for synchronizing motions, and for making certain that a prior motion has completed before beginning a new motion.

Once this command is issued, the calling program will not be able to communicate with the board until *axis*' encoder comes to rest. We recommend using **MCIsStopped( )** or **MCIsAtTarget( )** instead.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCWaitForStop(hCtlr: HCTRLR; axis: Word; dwell: Double); stdcall;
VB: Sub MCWaitForStop(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal dwell As Double)
LabVIEW:



MCCL Reference
WS

See Also
MCIsAtTarget( ), **MCIsStopped( )**, **MCWait( )**, **MCWaitForPosition( )**, **MCWaitForRelative( )**, **MCWaitForTarget( )**

MCWaitForTarget

MCWaitForTarget( ) waits for the specified *axis* to reach its target position. An optional dwell after the stop may be specified within this command to allow the mechanical system to come to rest.

```
void MCWaitForTarget(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double dwell                   // dwell time after stop
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number function is waiting for to reach the target position.
dwell	Delay time after stop has occurred.

Returns
This function does not return a value.

Comments
For a servo axis to be considered "at target" it must remain within the **Deadband** region for the **DeadbandDelay** period. **Deadband** and **DeadbandDelay** are specified in the **MCMOTIONEX** configuration structure.

Once this command is issued, the calling program will not be able to communicate with the board until *axis*' encoder settles within the **Deadband** region for the **DeadbandDelay** period. We recommend using **MCDecodeStatusEx( )** along with MC_STAT_AT_TARGET instead.

Compatibility

The DC2 and DCX-PC100 controllers do not support this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: procedure MCWaitForTarget(hCtlr: HCTRLR; axis: Word; dwell: Double); stdcall;
VB: Sub MCWaitForTarget(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal dwell As Double)
LabVIEW: Not Supported

MCCL Reference

WT

See Also

MCGetMotionConfigEx( ), **MCSetMotionConfigEx( )**, **MCWaitForPosition( )**, **MCWaitForRelative( )**, **MCWaitForStop( )**

Chapter Contents

Chapter 7

Reporting Functions

Reporting functions allow the calling program to query the board to determine how parameters have been configured, as well as getting information regarding the position and status of any given axis. Also included in this category are functions that allow the program to trap and decode errors.

To see examples of how the functions in this chapter are used, please refer to the online Motion Control API Reference.

MCDecodeStatusEx

MCDecodeStatusEx( ) permits you to test flags in the controller status word in a way that is independent of the type of controller being inspected.

```
long int MCDecodeStatus(
   HCTRLR hCtlr,                  // controller handle
   DWORD status,                  // status word data structure
   long int bit                   // status bit selection flag
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
status	Status value returned from a previous call to **MCGetStatusEx( )**.
bit	Status bit to decode. Over fifty different status bit flags (not all flags are supported by all controllers) are defined in the Constants section of this help file. Valid Bit constants begin with "MC_STAT_".

Returns

This function returns TRUE if the selected bit is set. Otherwise, FALSE is returned if the bit is not set or the bit does not apply to this controller type.

Comments

Using this function to test the status word returned by **MCGetStatusEx( )** isolates the program from controller dependent bit ordering of the status word. The sample programs include numerous examples of the **MCDecodeStatusEx( )** function.

To assist with proper constant selection two tables have been provided with the online help. The Status Word Lookup Table lists the constants in the same order as the status word bits they represent for each controller model, and has been included in Appendix C. A second table, The Status Word Cross Reference, lists the controller models supported by each constant, and will only be found in the online help.

Compatibility

There are no compatibility issues with this function.

Requirements

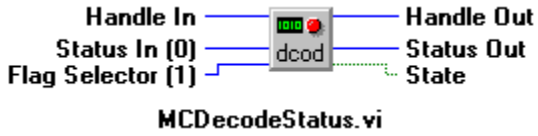
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.3 or higher

Prototypes

Delphi: function MCDecodeStatusEx(hCtlr: HCTRLR; status, bit: Longint): Longint; stdcall;

VB: Function MCDecodeStatusEx(ByVal hCtrlr As Integer, ByVal status As Long, ByVal bit As Long) As Long

LabVIEW:



MCCL Reference

None

See Also

MCGetStatusEx( ), online help sample programs

MCEnableInterrupt

MCEnableInterrupt( ) enables or disables the status word interrupt feature on motion controllers that support host interrupts. Users may elect to receive notification of interrupts via a message posted to a window message queue, or through a callback function.

```
long int MCEnableInterrupt(
   HWND hWnd,                    // window handle for notification messages
   HCTRLR hCtlr,                 // controller handle
   WORD axis                     // axis number
   DWORD mask                    // selects bits to use for interrupt
   MCINTERRUPTPROC lpIntFunc     // callback function pointer
);
```

Parameters

hWnd	Window handle to post notification messages to.
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number for which to enable or disable interrupt notifications.
mask	Bits set in this variable select the corresponding bits in the main status word of the controller which will cause an interrupt if set. Setting *mask* to zero disables interrupt notification for this axis.
lpIntFunc	Pointer to a callback function to be used for interrupt notification, or null to indicate that a message should be posted to the window function instead.

Returns

This function returns MCERR_NOERROR if there are no errors, or one of the **MCERR_xxxx** defined error codes if there was a problem.

Comments

This function permits you to configure an axis to notify your application program when any of the primary status word bits specified by *mask* go true.If the callback function agrument specifies a function the function will be called to notify the application of the status event. If the callback function agrument is NULL then a message will be posted to the window specified by *hWnd*.

Use the windows function **RegisterWindowsMessage( )** with the name "MCStatusNotify" to get the message identifier. The test for this message value in your window procedure. The WPARAM of the message will include the controller handle in the low word and the Axis number in the high word. The LPARAM value of this message will contain the status value.

The callback function must have the following signiture:

```
void CALLBACK MyIntProc( HWND hWnd, HCTRLR hCtlr, WORD waxis, DWORD status );
```

Where `MyIntProc` is any name you choose. If a windows handle was specified in the call to **MCEnableInterrupt( )** that will be the first argument to the callback function, the second agrument will be the controller handle, the third the axis number, and the forth a status word with bits set (true) for the bits that have just transitioned from false to true and are selected by the *mask*.

Only one notification window or callback function may be specified per axis per MCAPI handle at a time.

Compatibility

Only the MFX-PCI1000 series of motion controllers support the MCEnableInterrupt function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.3 or higher

Prototypes

Delphi: function MCEnableInterrupt(hWnd: HWND; hCtlr: HCTRLR; wAxis: WORD; mask:DWORD; lpIntFunc: MCInterruptProc): Longint; stdcall;

VB: Declare Function MCEnableInterrupt Lib "mcapi32.dll" (ByVal hWnd As Long, ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal mask As Long, lpIntFunc As Any)

MCCL Reference
EI

See Also
MCDecodeStatusEX( )), MCGetStatusEx( ), online help sample programs

---

MCErrorNotify

MCErrorNotify( ) registers with the MCAPI a specific window procedure that is to receive message based notification of API errors for this controller handle.

```
void MCErrorNotify(
   HWND hWnd,                    // error handling window procedure
   HCTRLR hCtlr,                 // controller handle
   DWORD errorMask               // mask to select error category
);
```

Parameters
hWnd	Handle of window procedure to receive error messages.
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
errorMask	Selects error categories to be notified about. Any combination of the MCERRMASK_xxxx constants may be OR'ed together to select errors to be reported. The constant MCERRMASK_STANDARD includes the most common error messages.

Returns
This function does not return a value.

Comments
Only one window procedure at a time may receive error messages for a controller handle. If another window procedure attempts to hook the error messages for a handle that already has an error handler, it will replace the current error handler. In practice, this is not a problem as applications have control of the handle. They can decide who to have hook the error notification mechanism.

The error notification message is a pre-agreed upon, inter-application message that goes by the name "MCErrorNotify". Application programs need to call the Windows function **RegisterWindowMessage( )** with the message name "MCErrorNotify" to obtain the numeric value if the message. The error message will have a numeric error code as its *wParam*, and a pointer to a null-terminated ASCII string representation of the name of the function that caused the error as its *lParam*. The CWDemo sample application includes an example of hooking the error notification loop and processing error messages.

In the event of a bad controller handle passed to an API function as part of an API call, an error message will be broadcast to every windows procedure. This is done because with a bad handle there is no way for the API to identify which window procedure should receive the error. Rather than quietly tell no one, the API plays it safe and tells everyone.

The standard Windows message queue is small and may be over-run if error messages occur in rapid succession. During application development, when errors are most likely, you may want to call the Windows function **SetMessageQueue( )** in your **WinMain** function to set the application queue to something larger than the default size of 8 messages.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.2 or higher

Prototypes
Delphi: procedure MCErrorNotify(hWnd: HWnd; hCtlr: HCTRLR; errorMask: Longint); stdcall;
VB: Sub MCErrorNotify(ByVal hWnd As Long, ByVal hCtrlr As Integer, ByVal errorMask As Long)
LabVIEW: Not Supported

MCCL Reference
None

See Also
MCGetError( ), **MCTranslateErrorEx( )**, CWDemo sample code

MCGetAccelerationEx

MCGetAccelerationEx( ) returns the current programmed acceleration value for the given axis, in whatever units the axis is configured for.

```
long int MCGetAccelerationEx(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* pAccel                 // acceleration return value
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query for acceleration
pAccel	Pointer to a double precision floating point variable that will hold the acceleration for the specified axis.

Returns
The acceleration value is placed in the variable specified by the pointer *pAccel* and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned and the variable pointed to by *pAccel* is left unchanged.

Comments

The acceleration value returned by this function is the same as the **Acceleration** field of the **MCMOTIONEX** structure returned by **MCGetMotionConfigEx( )**; **MCGetAccelerationEx( )** provides a short-hand method for obtaining just the acceleration value.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

The DC2 stepper axes do not support ramping.

Requirements

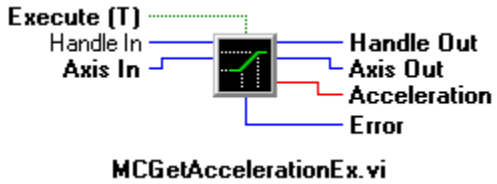
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes

Delphi: function MCGetAccelerationEx(hCtlr: HCTRLR; axis: Word; var pAccel: Double): Longint; stdcall;

VB: Function MCGetAccelerationEx(ByVal hCtrlr As Integer, ByVal axis As Integer, accel As Double) As Long

LabVIEW:



MCCL Reference

None

See Also

MCSetAcceleration( ), **MCGetMotionConfigEx( )**

MCGetAuxEncIdxEx

MCGetAuxEncIdxEx( ) returns the position where the auxiliary encoder's index pulse was observed.

```
long int MCGetAuxEncIdxEx(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* pIndex                 // index position return value
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pIndex	Pointer to a double precision floating point variable that will hold the auxiliary encoder index position for the specified axis.

Returns
The auxiliary encoder index position is placed in the variable specified by the pointer *pIndex* and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned and the variable pointed to by *pIndex* is left unchanged.

Comments
The auxiliary encoder's position may be set (to zero) using the **MCSetAuxEncPos( )** function. The index position reported will be relative to this zero position.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
The DC2, DCX-PCI100 controllers, MC100, MC110, MC150, and MC320 modules do not support auxiliary encoders. Closed-loop steppers do not support auxiliary encoder functions, since the connected encoder is considered a primary encoder.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes
Delphi: function MCGetAuxEncIdxEx(hCtlr: HCTRLR; axis: Word; var pIndex: Double): Longint; stdcall;
VB: Function MCGetAuxEncIdxEx(ByVal hCtrlr As Integer, ByVal axis As Integer, index As Double) As Long
LabVIEW: Not Supported

MCCL Reference
AZ

See Also
MCFindAuxEncIdx( ), **MCGetAuxEncPosEx( )**, **MCSetAuxEncPos( )**

MCGetAuxEncPosEx

MCGetAuxEncPosEx( ) returns the current position of the auxiliary encoder.

```
long int MCGetAuxEncPosEx(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   double* pPosition            // position return value
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.

axis	Axis number to query.
pPosition	Pointer to a double precision floating point variable that will hold the auxiliary encoder position for the specified axis.

Returns

The auxiliary encoder position is placed in the variable specified by the pointer *pPosition* and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned and the variable pointed to by *pPosition* is left unchanged.

Comments

The auxiliary encoder's position may be set using the **MCSetAuxEncPos( )** function.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

The DC2, DCX-PCI100 controllers, MC100, MC110, MC150, and MC320 modules do not support auxiliary encoders. Closed-loop steppers do not support auxiliary encoder functions, since the connected encoder is considered a primary encoder.

Requirements

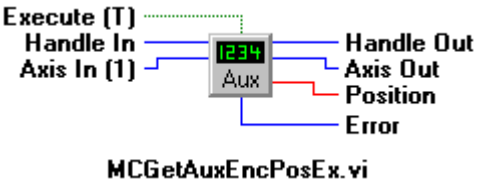
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes

Delphi: function MCGetAuxEncPosEx(hCtlr: HCTRLR; axis: Word; var pPosition: Double): Longint; stdcall;

VB: Function MCGetAuxEncPosEx(ByVal hCtrlr As Integer, ByVal axis As Integer, position As Double) As Long

LabVIEW:



MCCL Reference

AT

See Also

MCGetAuxEncIdxEx( ), **MCSetAuxEncPos ( ))**

MCGetAxisConfiguration

MCGetAxisConfiguration( ) obtains the configuration for the specified axis. Configuration information includes the axis type, servo motor update rates, stepper motor step rates, etc.

```
long int MCGetAxisConfiguration(
   HCTRLR hCtlr,                 // controller handle
   WORD axis,                    // axis number
   MCAXISCONFIG* pAxisCfg        // address of axis configuration structure
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pAxisCfg	Points to an **MCAXISCONFIG** structure that receives the configuration information.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

This function allows the application to query the driver about installed motor axis hardware and capabilities.

Before you call **MCGetAxisConfiguration( )** you must set the **cbSize** member to the size of the **MCAXISCONFIG** data structure. C/C++ programmers may use **sizeof( )**, Visual Basic and Delphi programmers will find current sizes for these data structures in the appropriate MCAPI.XXX header file.

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.0 or higher

Prototypes

Delphi: function MCGetAxisConfiguration(hCtlr: HCTRLR; axis: Word; var pAxisCfg: MCAXISCONFIG): Longint; stdcall;
VB: Function MCGetAxisConfiguration(ByVal hCtrlr As Integer, ByVal axis As Integer, axisCfg As MCAxisConfig) As Long
LabVIEW: Not Supported

MCCL Reference

Dual Port RAM

See Also
MCAXISCONFIG structure definition

MCGetBreakpointEx

MCGetBreakpointEx( ) returns the current breakpoint position as placed by the **MCWaitForPosition( )** or **MCWaitForRelative( )** command.

```
long int MCGetBreakpointEx(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* pBreakpoint            // breakpoint position return value
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pBreakpoint	Pointer to a double precision floating point variable that will hold the breakpoint position for the specified axis.

Returns
The breakpoint position is placed in the variable specified by the pointer *pBreakpoint* and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned and the variable pointed to by *pBreakpoint* is left unchanged.

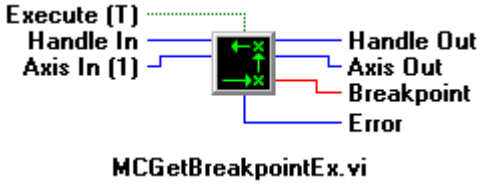
Comments

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
The DCX-PC100 controller and stepper axes do not support this command.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes
Delphi: function MCGetBreakpointEx(hCtlr: HCTRLR; axis: Word; var pBreakpoint: Double): Longint; stdcall;
VB: Function MCGetBreakpointEx(ByVal hCtrlr As Integer, ByVal axis As Integer, breakpoint As Double) As Long
LabVIEW:



MCCL Reference
TB

See Also
MCWaitForPosition( ), **MCWaitForRelative( )**

MCGetCaptureData

MCGetCaptureData( ) retrieves data collected following the most recent **MCCaptureData( )** call.

```
long int MCGetCaptureData(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number to get capture data from
   long int type,                 // type of capture data to retrieve
   long int start,                // index of starting point
   long int points,               // number of data points to retrieve
   double* pData                  // pointer to data array to for data
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to query.
type Specifies the type of data to retrieve:

Value	Description
MC_CAPTURE_ACTUAL	Retrieves the captured actual position data.
MC_CAPTURE_ERROR	Retrieves the following error (difference between actual and optimal positions).
MC_CAPTURE_OPTIMAL	Retrieves the captured optimal position data.
MC_CAPTURE_TORQUE	Retrieves the captured torque data.

start Index of the first data point to retrieve. The index is zero based, i.e. the first data point is 0, not 1.
points Total number of data points to retrieve.
pData Pointer to a double precision floating point variable that will hold the breakpoint position for the specified axis.

Returns
This function places one or more captured data values in the array specified by the pointer *pData*, and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned and state of the array pointed to by *pData* is undefined.

Comments
Capture data settings (number of points, delay, etc.) are set with the **MCCaptureData( )** function.

Beginning with version 3.0 of the MCAPI users may use the **MCGetAxisConfiguration( )** function to determine the data capture capabilities of an axis.

Compatibility
The DC2 stepper axes, and the MC100, MC110, MC150, MC160 modules when installed on the DCX-PC100 controller do not support data capture. The DCX-PCI100 controller does not support torque mode nor do any open loop stepper axes, which prevents the capture of torque values.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes
Delphi: function MCGetCaptureData(hCtlr: HCTRLR; axis: Word; type, start, points: Longint; var pData: Double): Longint; stdcall;

VB: Function MCGetCaptureData(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal start, ByVal argtype As Long, ByVal points As Long, data As Double) As Long

LabVIEW: Not Supported

MCCL Reference
DO, DR, DQ

See Also
MCCaptureData( ), **MCGetAxisConfiguration( )**

MCGetContourConfig

MCGetContourConfig( ) obtains the contouring configuration for the specified axis.

```
long int MCGetContourConfig(
   HCTRLR hCtlr,                 // controller handle
   WORD axis,                    // axis number
   MCCONTOUR* pContour           // structure to hold contour data
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pContour	Points to an **MCCONTOUR** structure that receives the configuration information for Axis.

Returns
The return value is TRUE if the function is successful. A return value of FALSE indicates the function did not find the Axis specified (*hCtlr* or *axis* incorrect).

Comments

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: function MCGetContourConfig(hCtlr: HCTRLR; axis: Word; var pContour: MCCONTOUR): SmallInt; stdcall;
VB: Function MCGetContourConfig Lib(ByVal hCtrlr As Integer, ByVal axis As Integer, contour As MCContour) As Integer
LabVIEW: Not Supported

MCCL Reference
Controller RAM Motor Tables

See Also
MCSetContourConfig( ), **MCCONTOUR** structure definition

MCGetContouringCount

MCGetContouringCount( ) obtains the current contour path motion that an axis is performing.

```
long int MCGetContouringCount(
   HCTRLR hCtlr,                  // controller handle
   WORD axis                      // axis number
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to query.

Returns
The return value is the number of linear or user defined contour path motions that have been completed.

Comments
This function allows the application to determine in what area of a continuous path motion an axis is at any given time.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: function MCGetContouringCount(hCtlr: HCTRLR; axis: Word): Longint; stdcall;
VB: Function MCGetContouringCount(ByVal hCtrlr As Integer, ByVal axis As Integer) As Long
LabVIEW: Not Supported

MCCL Reference
TX

See Also
MCGetContourConfig( ), **MCSetContourConfig( )**, **MCCONTOUR** structure definition

---

MCGetCount

MCGetCount( ) retrieves various count values from the specified *axis.*

```
long int MCGetCount(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   long int type,                 // type of count to retrieve
   long int* pCount               // variable to hold count value
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to query.
type Specifies the type of data to retrieve:

Value	Description
MC_COUNT_CAPTURE	Retrieves the number of captured positions in high-speed capture mode.
MC_COUNT_COMPARE	Retrieves the number of successful comparisons in high-speed compare mode.
MC_COUNT_CONTOUR	Retrieves the index of the currently executing contour move in contouring mode.
MC_COUNT_FILTER	Retrieves the number of digital filter coefficients currently loaded.
MC_COUNT_FILTERMAX	Retrieves the maximum number of digital filter coefficients supported.

pCount Variable to hold requested count value.

Returns
MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned.

Comments
MCGetCount( ) is a general purpose function for retrieving values related to high-speed capture mode, high-speed compare mode, contouring mode, and digital filter mode.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
The DC2 stepper axes, and the MC100, MC110, MC150, MC160 modules when installed on the DCX-PC100 controller do not support data capture. The DCX-PCI100 controller does not support torque mode nor do any stepper axes, which prevents the capture of torque values. The DC2, DCX-PC100, DCX-AT200, and DCX-PCI100 controllers do not support high-speed position compare. The MCAPI does not does not support contouring on the DC2, DCX-PC100, and DCX-PCI100 controllers. The DC2, DCX-PC100, DCX-AT200, DCX-PCI100, MFX-PCI1000 controllers, MC360, and MC362 modules do not support digital filtering.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.1 or higher

Prototypes
Delphi: function MCGetCount(hCtlr: HCTRLR; axis: Word; type: Longint; var pCount: Longint): Longint; stdcall;
VB: Function MCGetCount(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal argtype As Long, count As Long) As Long
LabVIEW: Not Supported

MCCL Reference
CG, GC, TX

See Also
MCGetContouringCount( )

MCGetDecelerationEx

MCGetDecelerationEx( ) returns the current programmed deceleration value for the given axis, in whatever units the axis is configured for.

```
long int MCGetDecelerationEx(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* pDecel                 // deceleration return value
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pDecel	Pointer to a double precision floating point variable that will hold the deceleration for the specified axis.

Returns

The deceleration is placed in the variable specified by the pointer *pDecel* and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned and the variable pointed to by *pDecel* is left unchanged.

Comments

The deceleration value is the same as that reported by the **MCGetMotionConfigEx( )** function, these functions provide a short-hand method for obtaining just the deceleration value.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

There are no compatibility issues with this function.

Requirements

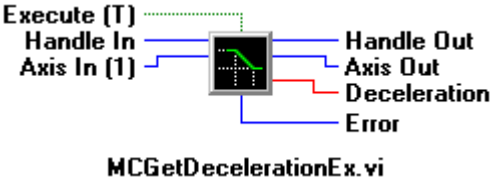
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes

Delphi: function MCGetDecelerationEx(hCtlr: HCTRLR; axis: Word; var pDecel: Double): Longint; stdcall;

VB: Function MCGetDecelerationEx(ByVal hCtrlr As Integer, ByVal axis As Integer, decel As Double) As Long

LabVIEW:



MCCL Reference
Controller RAM Motor Tables

See Also
MCSetDeceleration( ), **MCGetMotionConfigEx( )**

MCGetDigitalFilter

MCGetDigitalFilter( ) obtains the digital filter coefficients for the specified axis.

```
long int MCGetDigitalFilter(
   HCTRLR hCtlr                  // controller handle
   WORD axis,                    // axis number
   double* pCoeff,               // array to hold retrieved coefficients
   long int num,                 // number of coefficients to retrieve
   long int* pActual             // number of valid coefficients retrieved
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pCoeff	Array to hold retrieved coefficients, must be *num* elements long (or longer). If this pointer is NULL, no coefficients are retrieved.
num	Number of coefficients to retrieve, cannot be larger than the maximum digital filter size supported by the controller.
pActual	Points to long integer that will be set equal to the number of valid coefficients currently loaded for this axis. If this pointer is NULL, no value is returned.

Returns
MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned.

Comments
This function retrieves zero or more of the digital filter coefficients currently loaded in an axis. Optionally the actual number of loaded coefficients is also returned (this value is also available from **MCGetCount( )**).

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
The DC2, DCX-PC100, DCX-AT200, DCX-PCI100, MFX-PCI1000 controllers, MC360, and MC362 modules do not support digital filtering.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas

Library: use mcapi32.lib
Version: MCAPI 3.1 or higher

Prototypes

Delphi: function MCGetDigitalFilter(hCtlr: HCTRLR; axis: Word; coeff: Array of Double; num: Longint; var pActual: Longint): Longint; stdcall;

VB: Function MCGetDigitalFilter(ByVal hCtrlr As Integer, ByVal axis As Integer, coeff As Double, ByVal num As Long, actual As Long) As Long

LabVIEW: Not Supported

MCCL Reference

GF

See Also

MCEnableDigitalFilter( ), **MCGetCount( )**, **MCIsDigitalFilter( )**, **MCSetDigitalFilter( )**

MCGetError

MCGetError( ) returns the most recent error code for *hCtlr*.

```
short int MCGetError(
   HCTRLR hCtlr                  // controller handle
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.

Returns

The return value is a numeric error code (or MCERR_NOERROR if there is no error) for the most recent error detected for the specified controller.

Comments

The error is cleared (set equal to MCERR_NOERROR) after it has been read. Errors are maintained on a per-handle basis, such that calls to **MCGetError( )** only return errors that occurred during function calls that used the same handle.

A more flexible way to detect errors is to use the **MCErrorNotify( )**. This function delivers error messages directly to the window procedure of your choice.

Compatibility

There are no compatibility issues with this function.

Requirements

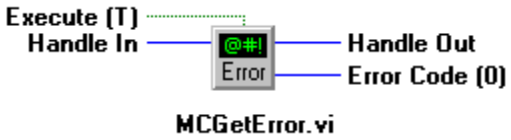
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.2 or higher

Prototypes

Delphi: function MCGetError(hCtlr: HCTRLR): SmallInt; stdcall;

VB: Function MCGetError(ByVal hCtrlr As Integer) As Integer

LabVIEW:



MCCL Reference

None

See Also

MCErrorNotify( ), **MCTranslateErrorEx( )**

MCGetFilterConfigEx

MCGetFilterConfigEx( ) obtains the current PID filter configuration for a servo motor or the closed-loop configuration for a stepper motor operating in closed-loop mode. Please see the online MCAPI Reference for the **MCGetFilterConfig( )** prototype.

```
long int MCGetFilterConfigEx(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   MCFILTEREX* pFilter            // address of filter configuration
                                  // structure
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pFilter	Points to an **MCFILTEREX** structure that receives the PID filter configuration information for *axis*.

Returns

MCGetFilterConfigEx( ) places the PID filter settings in the structure specified by the pointer *pFilter*. MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned.

Comments

This function must be used to obtain the current PID filter configuration for a servo motor or the closed-loop configuration for a stepper motor operating in closed-loop mode.

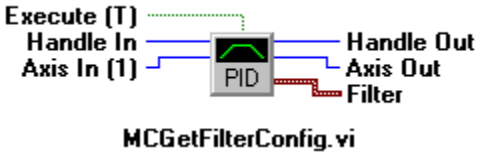
Closed-loop stepper operation requires firmware version 2.1a or higher on the DCX-PCI300 and firmware version 2.5a or higher on the DCX-AT300.

You may not set the *axis* parameter to MC_ALL_AXES for this command..

Compatibility
VelocityGain is not supported on the DCX-PCI100 controller, MC100, MC110 modules, or closed-loop steppers. **AccelGain** is not supported on the DC2, DCX-PC100, and DCX-PCI100 controllers. **DecelGain** is not supported on the DC2, DCX-PC100, and DCX-PCI100 controllers.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.2 or higher

Prototypes
Delphi: function MCGetFilterConfigEx(hCtlr: HCTRLR; axis: Word; var pFilter: MCFILTEREX): SmallInt; stdcall;
VB: Function MCGetFilterConfigEx(ByVal hCtrlr As Integer, ByVal axis As Integer, filter As MCFilterEx) As Integer
LabVIEW:



MCCL Reference
TD, TF, TG, TI, TL, Controller RAM Motor Tables

See Also
MCSetFilterConfigEx( ), **MCFILTEREX** structure definition

MCGetFollowingError

MCGetFollowingError( ) returns the current following error (difference between the actual and the optimal positions) for the specified axis.

```
long int MCGetFollowingError(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* pError                 // following error return value
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pError	Points to a double precision variable that will hold the following error.

Returns
This function places the following error in the variable specified by the pointer *pError*, and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned and the variable pointed to by pError is left unchanged.

Comments

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

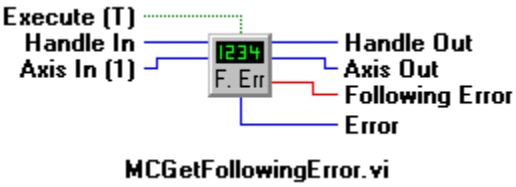
There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes

Delphi: function MCGetFollowingError(hCtlr: HCTRLR; axis: Word; var pError: Double): Longint; stdcall;
VB: Function MCGetFollowingError(ByVal hCtrlr As Integer, ByVal axis As Integer, error As Double) As Long
LabVIEW:



MCCL Reference

TF

See Also

MCGetOptimalEx( ), **MCGetPositionEx( )**

MCGetGain

MCGetGain( ) returns the current gain setting for the specified axis.

```
long int MCGetGain(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* pGain                  // gain return value
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pGain	Points to a double precision variable that will hold the gain value.

Returns

MCGetGain( ) places the gain value in the variable specified by the pointer *pGain* and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned and the variable pointed to by *pGain* is left unchanged.

Comments

The gain value is the same as that reported by the **MCGetMotionConfigEx( )** function, this function provide a short-hand method for obtaining just the gain value.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

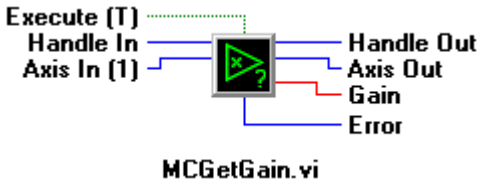
There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes

Delphi: function MCGetGain(hCtlr: HCTRLR; axis: Word; var pGain: Double): Longint; stdcall;
VB: Function MCGetGain(ByVal hCtrlr As Integer, ByVal axis As Integer, gain As Double) As Long
LabVIEW:



MCCL Reference

TG

See Also

MCGetMotionConfigEx( ) , **MCSetGain( )**

MCGetIndexEx

MCGetIndexEx( ) returns the position where the encoder index pulse was observed for the specified axis, in whatever units the axis is configured for.

```
long int MCGetIndexEx(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* pIndex                 // index position return value
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pIndex	Pointer to a double precision floating point variable that will hold the index position for the specified axis.

Returns

The index position is placed in the variable specified by the pointer *pIndex* and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned and the variable pointed to by pIndex is left unchanged.

Comments

Controller resets and the **MCSetPosition( )** function may be change the position reading of the primary encoder.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

The MC100, MC110 modules, and all stepper axes do not support this function.

Requirements

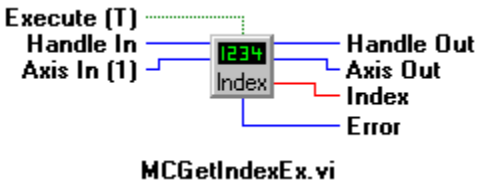
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes

Delphi: function MCGetIndexEx(hCtlr: HCTRLR; axis: Word; var pIndex: Double): Longint; stdcall;

VB: Function MCGetIndexEx(ByVal hCtrlr As Integer, ByVal axis As Integer, index As Double) As Long

LabVIEW:



MCCL Reference

TZ

See Also

MCGetAuxEncIdxEx( ), **MCSetPosition( )**

MCGetInstalledModules

MCGetInstalledModules( ) enumerates the types of modules installed on a motion controller.

```
long int MCGetInstalledModules(
   HCTRLR hCtlr,                  // controller handle
   long int* modules,             // pointer to an array for controller type
                                  // IDs
   long int size                  // size of Modules array
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
modules	Pointer to an array of long integers, filled with module types on return.
size	Size of *modules* array (number of integers).

Returns

MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned.

Comments

MCGetInstalledModules( ) fills the *modules* array with module type identifiers, where the type of module installed in position #1 on the controller is stored in Modules[0], the type of module installed in position #2 on the controller is stored in Modules[1], etc. In order to list all installed controllers the array must have a size at least equal to the value in the **MaximumModules** field of the **MCPARAMEX( )** data structure.

Compatibility

The DC2 and MFX-PCI1000 controllers do not support this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.0 or higher

Prototypes

Delphi:	function MCGetInstalledModules(hCtlr: HCTRLR; modules: Array of LongInt; size: LongInt): Longint; stdcall;
VB:	Function MCGetInstalledModules(ByVal hCtrlr As Integer, modules As Any, ByVal size As Long) As Long
LabVIEW:	Not Supported

MCCL Reference

None

See Also

MCGetConfigurationEx( )

MCGetJogConfig

MCGetJogConfig( ) obtains the current jog configuration block for the specified axis.

```
short int MCGetJogConfig(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   MCJOG* pJog                    // address of jog configuration structure
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number from which to retrieve jog information.
pJog	Points to a **MCJOG** structure that contains jog configuration information for *axis*.

Returns

The return value is TRUE if the function is successful. Otherwise it returns FALSE, indicating the function did not find the *axis* specified (hCtlr or *axis* incorrect).

Comments

This function must be used to obtain current jog configuration information for an axis.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

The DCX-PCI, MFX-PCI1000 controllers, DC2 stepper axes, MC150, and MC160 modules do not support jogging.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi:	function MCGetJogConfig(hCtlr: HCTRLR; axis: Word; var pJog: MCJOG): SmallInt; stdcall;
VB:	Function MCGetJogConfig(ByVal hCtrlr As Integer, ByVal axis As Integer, jog As MCJog) As Integer
LabVIEW:	Not Supported

MCCL Reference

Controller RAM Motor Tables

See Also

MCEnableJog( ), **MCGetJogConfig( )**, **MCJOG** structure definition

MCGetLimits

MCGetLimits( ) obtains the current hard and soft limit settings for the specified axis.

```
long int MCGetLimits(
   HCTRLR hCtlr,                // controller handle
   WORD axis,                   // axis number
   short int* pHardMode,        // hard limit mode flags
   short int* pSoftMode,        // soft limit mode flags
   double* pLimitMinus,         // soft low limit value
   double* pLimitPlus           // soft high limit value
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.

axis Axis number to query.

pHardMode Combination of limit mode flags for the hard limits. See description of *pSoftMode* for details.

pSoftMode Combination of the following limit mode flags for the soft limits:

Value	Description
MC_LIMIT_PLUS	Enables the positive limit.
MC_LIMIT_MINUS	Enables the negative limit.
MC_LIMIT_BOTH	Enables both the positive and negative limits.
MC_LIMIT_OFF	Limit stopping mode is set to turn the motor off when a limit is tripped.
MC_LIMIT_ABRUPT	Limit stopping mode is set to abrupt (target position is set to current position and PID loop stops axis as quickly as possible).
MC_LIMIT_SMOOTH	Limit stopping mode is set to smooth (axis executes pre-programmed deceleration when limit is tripped).
MC_LIMIT_INVERT	Inverts the polarity of the hardware limit switch inputs. This value may not be used with soft limits.

pLimitMinus Pointer to a variable where the negative limit value for soft limits, if supported by this controller, will be stored.

pLimitPlus Pointer to a variable where the positive limit value for soft limits, if supported by this controller, will be stored.

Returns

MCGetLimits( ) returns the value MCERR_NOERROR if the function completed without errors. If there was an error, one of the MCERR_xxxx error codes is returned, and the variables pointed to by the function pointers will be left in an undetermined state.

Comments

The limit settings are the same as those reported by the **MCGetMotionConfigEx( )** function, this function provide a short-hand method for obtaining just the limit settings.

Beginning with Version 2.23 of the Motion Control API you may pass a NULL pointer for *pHardMode*, *pSoftMode*, *pLimitMinus*, or *pLimitPlus*. This permits a program to easily ignore values it is not interested in. A program that needs to check the Hard Limit settings might set all the pointers for Soft Limit values to NULL to ignore those values, as opposed to having to create dummy variables to hold the values that will never be used. Because this feature is new in Version 2.23, only applications that do not require backward compatibility with an earlier MCAPI version should take advantage of it.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

The DC2 and DCX-PC100 controllers do not support soft limits.

Requirements

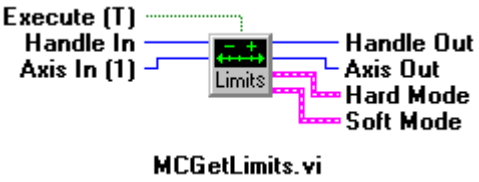
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes

Delphi: function MCGetLimits(hCtlr: HCTRLR; axis: Word; var pHardMode, pSoftMode: SmallInt; var pLimitMinus, pLimitPlus: Double): Longint; stdcall;

VB: Function MCGetLimits(ByVal hCtrlr As Integer, ByVal axis As Integer, hardMode As Integer, softMode As Integer, limitMinus As Double, limitPlus As Double) As Long

LabVIEW:



MCCL Reference

Controller RAM Motor Tables

See Also

MCGetMotionConfigEx( ), **MCSetLimits( )**, **MCSetMotionConfigEx( )**

MCGetModuleInputMode

MCGetModuleInputMode( ) returns the current input mode for the specified axis.

```
long int MCGetModuleInputMode(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   long int* mode                 // input mode value
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to query.
mode Pointer to a long integer variable that will hold the input mode for the specified axis:

Value	Description
MC_IM_OPENLOOP	Stepper motor axis is in open-loop mode.
MC_IM_CLOSEDLOOP	Stepper motor axis is in closed-loop mode.

Returns

The return value is MCERR_NOERROR if no errors were detected. If there was an error, one of the MCERR_xxxx error codes is returned and the variable pointed to by *mode* is left unchanged.

Comments

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

The DC2, DCX-PC100, DCX-PCI100, DCX-AT100, and DCX-AT200 controllers do not support a module which is capable of closed-loop stepper operation. The MC362 module is not capable of closed-loop stepper operation.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.2 or higher

Prototypes

Delphi: function MCGetModuleInputMode(hCtlr: HCTRLR; axis: Word; var mode: LongInt): Longint; stdcall;
VB: Function MCGetModuleInputMode(ByVal hCtrlr As Integer, ByVal axis As Integer, mode As Long) As Long
LabVIEW: Not Supported

MCCL Reference
IM

See Also
MCSetModuleInputMode( )

MCGetMotionConfigEx

MCGetMotionConfigEx( ) obtains the current motion configuration block for the specified axis.

```
short int MCGetMotionConfigEx(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   MCMOTIONEX* pMotion            // address of motion configuration
                                  // structure
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pMotion	Points to an **MCMOTIONEX** structure that receives motion configuration information for *axis*.

Returns
The return value is TRUE if the function is successful. A return value of FALSE indicates the function did not find the *axis* specified (*hCtlr* or *axis* incorrect).

Comments
This function provides a way of initializing a **MCMOTIONEX** structure with the current motion parameters for the given *axis*. When you need to setup many of the parameters for an axis it is easier to call **MCGetMotionConfigEx( )**, update the **MCMOTIONEX** structure, and write the changes back using **MCSetMotionConfigEx( )**, rather than use a Get/Set function call for each parameter.

Note that some less often used parameters will only be accessible from this function and from **MCSetMotionConfigEx( )** - they do not have individual Get/Set functions.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
Acceleration is not supported on the DC2 stepper axes. **Deceleration** is not supported on the DCX-PCI100 controller, MC100, MC110, MC150, or MC160 modules. **MinVelocity** is not supported on the DCX-PCI100, DCX-PC100, or DC2 controllers. **Torque** is not supported on the DCX-PCI100 controller, MC100, or MC110 modules. **Deadband** is not supported on the DCX-PC100 controller, DC2 stepper axes, MC150, MC160, MC260, MC360 or MC362 modules. **DeadbandDelay** is not supported on the DCX-PC100 controller, DC2 stepper axes, MC150, MC160, MC260, MC360 or MC362 modules. **StepSize** is not supported on the DC2 or DCX-PCI100 controllers. **Current** is not

supported on the DC2 or DCX-PCI100 controllers. **SoftLimitMode** is not supported on the DC2 or DCX-PC100 controllers. **SoftLimitLow** is not supported on the DC2 or DCX-PC100 controllers. **SoftLimitHigh** is not supported on the DC2 or DCX-PC100 controllers. **EnableAmpFault** is not supported on the DC2 controllers. **UpdateRate** is not supported on the DC2 or DCX-PCI100 controllers.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: function MCGetMotionConfigEx(hCtlr: HCTRLR; axis: Word; var pMotion: MCMOTIONEX): SmallInt; stdcall;
VB: Function MCGetMotionConfigEx(ByVal hCtrlr As Integer, ByVal axis As Integer, motion As MCMotionEx) As Integer
LabVIEW: Not Supported

MCCL Reference
TG, Controller RAM Motor Tables

See Also
MCSetMotionConfigEx( ), **MCMOTIONEX** structure definition

MCGetOperatingMode

MCGetOperatingMode( ) returns the current operating mode for the specified axis.

```
long int MCGetOperatingMode(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   long int* mode                 // operating mode value
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to query.
mode Pointer to a long integer variable that will hold the operating mode for the specified axis:

Value	Description
MC_MODE_CONTOUR	Contouring mode operation.
MC_MODE_GAIN	Gain mode operation.
MC_MODE_POSITION	Position mode operation.
MC_MODE_TORQUE	Torque mode operation.
MC_MODE_UNKNOWN	Unable to determine current mode of operation.
MC_MODE_VELOCITY	Velocity mode operation.

Returns
The return value is MCERR_NOERROR if no errors were detected. If there was an error, one of the MCERR_xxxx error codes is returned and the variable pointed to by *mode* is left unchanged.

Comments

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.2 or higher

Prototypes
Delphi: function MCGetOperatingMode(hCtlr: HCTRLR; axis: Word; var mode: LongInt): Longint; stdcall;
VB: Function MCGetOperatingMode(ByVal hCtrlr As Integer, ByVal axis As Integer, mode As Long) As Long
LabVIEW: Not Supported

MCCL Reference
None

See Also
MCSetOperatingMode( )

MCGetOptimalEx

MCGetOptimalEx( ) returns the current optimal position from the trajectory generator for the specified axis, in whatever units the axis is configured for.

```
long int MCGetOptimalEx(
   HCTRLR hCtlr,                 // controller handle
   WORD axis,                    // axis number
   double* pOptimal              // optimal return value
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pOptimal	Pointer to a double precision floating point variable that will hold the optimal position for the specified axis.

Returns

The optimal position is placed in the variable specified by the pointer *pOptimal* and a zero is returned, if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned and the variable pointed to by *pOptimal* is left unchanged.

Comments

The trajectory generator generates an optimal position based upon an ideal (i.e. error free) motor. The PID loop then compares the actual position to the optimal position to calculate a correction to the actual trajectory. The maximum difference allowed between the optimal and actual positions is set with the **FollowingError** member of an **MCFILTEREX** structure.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

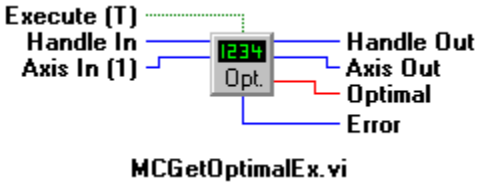
The DC2 stepper axes do not support this command.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes

Delphi: function MCGetOptimalEx(hCtlr: HCTRLR; axis: Word; var pOptimal: Double): Longint; stdcall;
VB: Function MCGetOptimalEx(ByVal hCtrlr As Integer, ByVal axis As Integer, optimal As Double) As Long
LabVIEW:



MCCL Reference

TO

See Also

MCGetFilterConfigEx( ), **MCSetFilterConfigEx( )**, **MCSetPosition( )**

MCGetPositionEx

MCGetPositionEx( ) returns the current position for the specified axis, in whatever units the axis is configured for.

```
void MCGetPositionEx(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* pPosition              // position return value
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pPosition	Pointer to a double precision floating point variable that will hold the position for the specified axis.

Returns

The position value is placed in the variable specified by the pointer *pPosition* and a zero is returned, if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned and the variable pointed to by *pPosition* is left unchanged.

Comments

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

There are no compatibility issues with this function.

Requirements

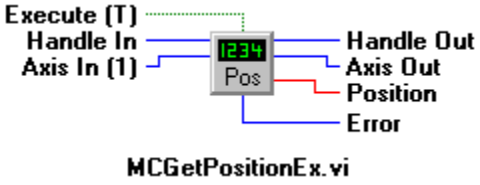
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes

Delphi: function MCGetPositionEx(hCtlr: HCTRLR; axis: Word; var pPosition: Double): Longint; stdcall;

VB: Function MCGetPositionEx(ByVal hCtrlr As Integer, ByVal axis As Integer, position As Double) As Long

LabVIEW:



MCCL Reference
TP

See Also
MCSetPosition( ), **MCSetScale( )**

MCGetProfile

MCGetProfile( ) returns the current acceleration / deceleration profile for the specified axis.

```
long int MCGetProfile(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   WORD* pProfile                 // profile return value
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to query.
pProfile Pointer to a WORD variable that will hold the profile for the specified axis:

Value	Description
MC_PROF_PARABOLIC	Indicates that a parabolic acceleration / deceleration profile has been selected.
MC_PROF_SCURVE	Indicates that an S-curve acceleration / deceleration profile has been selected.
MC_PROF_TRAPEZOID	Indicates that a trapezoidal acceleration / deceleration profile has been selected.
MC_PROF_UNKNOWN	This value is returned when **MCGetProfile( )** cannot determine the current profile setting.

Returns
The return value is MCERR_NOERROR, if no errors were detected. If there was an error, the return value is one of the MCERR_xxxx error codes is returned and the variable pointed to by *pProfile* is left unchanged.

Comments
To determine if the controller supports user configurable acceleration profiles check the **CanChangeProfile** field of the **MCPARAMEX** structure returned by **MCGetConfigurationEx( )**.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes
Delphi: function MCGetProfile(hCtlr: HCTRLR; axis: Word; var pProfile: Word): Longint; stdcall;
VB: Function MCGetProfile(ByVal hCtrlr As Integer, ByVal axis As Integer, profile As Integer) As Long
LabVIEW: Not Supported

MCCL Reference
Controller RAM Motor Tables

See Also
MCSetProfile( ), **MCPARAMEX** structure definition

MCGetRegister

MCGetRegister( ) returns the value of the specified general purpose register.

```
long int MCGetRegister(
   HCTRLR hCtlr,                  // controller handle
   long int register,             // register number
   void* pValue                   // pointer to variable to hold register
                                  // value
   long int type                  // type of variable pointed to by pValue
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
register Register number to read from (0 to 255).
pValue Pointer to a variable that will hold the register contents.
type Type of data pointed to by *pValue*:

Value	Description
MC_TYPE_LONG	Indicates *pValue* points to a variable of type long integer.
MC_TYPE_DOUBLE	Indicates *pValue* points to a variable of type double precision floating point.
MC_TYPE_FLOAT	Indicates *pValue* points to a variable of type single precision floating point.

Returns
The return value is MCERR_NOERROR, if no errors were detected. If there was an error, the return value is one of the MCERR_xxxx error codes is returned and the variable pointed to by *pValue* is left unchanged.

Comments

MCGetRegister( ) and **MCSetRegister( )** allow you to read from and write to, respectively, the general purpose registers on the motion controller. When running background tasks on a multitasking controller the only way to communicate with the background tasks is to pass parameters in the general purpose registers.

You cannot read from the local registers (registers 0 - 9) of a background task. When you need to communicate with a background task be sure to use one or more of the global registers (10 - 255).

To determine if your controller supports multi-tasking check the **MultiTasking** field of the **MCPARAMEX** structure returned by **MCGetConfigurationEx( )**.

Compatibility

There are no compatibility issues with this function.

Requirements

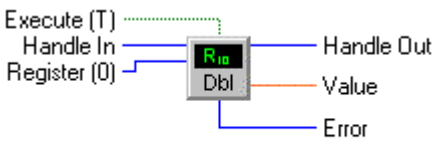
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.0 or higher

Prototypes

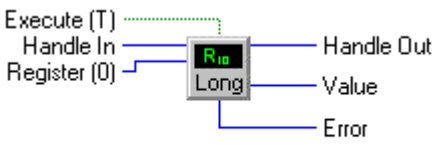
Delphi: function MCGetRegister(hCtlr: HCTRLR; register: Longint; var pValue: Pointer; type: Longint): Longint; stdcall;

VB: Function MCGetRegister(ByVal hCtrlr As Integer, ByVal register As Long, value As Any, ByVal argtype As Long) As Long

LabVIEW:



MCGetRegisterDouble.vi



MCGetRegisterLong.vi

MCCL Reference

TR

See Also

MCSetRegister( )

MCGetScale

MCGetScale( ) obtains the current scaling factors for the specified axis.

```
void MCGetScale(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   MCSCALE* pScale                // address of scale factors structure
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pScale	Pointer to a **MCSCALE** structure that will hold scaling information for *axis*.

Returns

The return value is TRUE if the function is successful. A return value of FALSE indicates the function did not find the *axis* specified (*hCtlr* or *axis* incorrect).

Comments

Scaling allows the application to communicate with the controller in real world units such as inches, meters, and radians; as opposed to low level (i.e. un-scaled) values such as raw encoder counts, etc.

In order to see if a controller supports scaling, an application can test the Boolean flag **CanDoScaling** in the **MCPARAMEX** structure returned by **MCGetConfigurationEx( )**.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

The DC2 and DCX-PC controllers do not support scaling.

Requirements

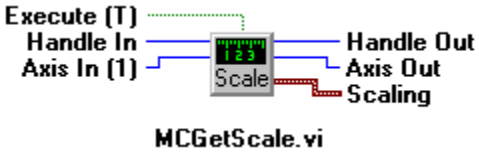
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: function MCGetScale(hCtlr: HCTRLR; axis: Word; var pScale: MCSCALE): SmallInt; stdcall;

VB: Function MCGetScale(ByVal hCtrlr As Integer, ByVal axis As Integer, scale As MCScale) As Integer

LabVIEW:



MCCL Reference

Controller RAM Motor Tables

See Also

MCGetConfigurationEx( ), **MCSetScale( )**, **MCSCALE** structure definition

MCGetServoOutputPhase

MCGetServoOutputPhase( ) returns the current servo output phasing for the specified axis.

```
long int MCGetServoOutputPhase(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   WORD* pPhase                   // phase return value
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.

axis Axis number to query for phase setting.

pPhase Pointer to a WORD variable that will hold the phase setting for the specified axis:

Value	Description
MC_PHASE_STD	Indicates that the axis is configured for standard phasing.
MC_PHASE_REV	Indicates that the axis is configured for reverse phasing.

Returns

The return value is MCERR_NOERROR if no errors were detected. If there was an error, the return value is one of the MCERR_xxxx error codes is returned, and the variable pointed to by *pPhase* is left unchanged.

Comments

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

The MC100 and MC110 modules do not support phase reverse.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes

Delphi: function MCGetServoOutputPhase(hCtlr: HCTRLR; axis: Word; var pPhase: Word): Longint; stdcall;

VB: Function MCGetServoOutputPhase(ByVal hCtrlr As Integer, ByVal axis As Integer, phase As Integer) As Long

LabVIEW: Not Supported

MCCL Reference
None

See Also
MCSetServoOutputPhase( )

MCGetStatusEx

MCGetStatusEx( ) returns the controller dependent status word for the specified axis.

```
long int MCGetStatus(
   HCTRLR hCtlr,                  // controller handle
   WORD axis                      // axis number
   MCSTATUSEX* status             // status words data structure
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
status	Pointer to an MCSTATUSEX data structure that will hold the retrieved status words.

Returns
The return value is the 32-bit status word for the selected axis.

Comments
Please refer to the hardware manual for your controller for specific information about meaning and location of the flags located within the status word. As an alternative, the MCAPI function **MCDecodeStatusEx( )** provides a controller-independent way to process the flags in the status word.

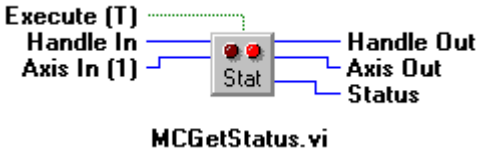
You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: function MCGetStatusEx(hCtlr: HCTRLR; axis: Word): Longint; stdcall;
VB: Function MCGetStatusEx(ByVal hCtrlr As Integer, ByVal axis As Integer) As Long

LabVIEW:



MCCL Reference
TS

See Also
MCDecodeStatusEx( ), Controller hardware reference manual

MCGetTargetEx

MCGetTargetEx( ) returns the move target position, as set by the most recent **MCMoveAbsolute( )** or **MCMoveRelative( )** function call, for the specified axis.

```
void MCGetTargetEx(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* pTarget                // target position return
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pTarget	Pointer to a double precision floating point variable that will hold the target position for the specified axis.

Returns
The target position value is placed in the variable specified by the pointer *pTarget* and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned, and the variable pointed to by *pTarget* is left unchanged.

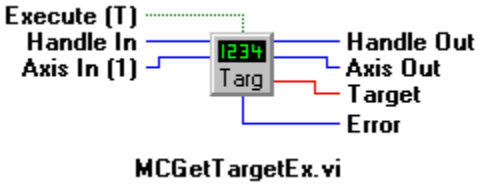
Comments
The API move functions **MCMoveAbsolute( )** and **MCMoveRelative( )** update the target position for an axis. The controller will then generate an optimal trajectory to the target position, and the PID loop will seek to minimize the error (difference between actual and optimal trajectories).

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: function MCGetTargetEx(hCtlr: HCTRLR; axis: Word; var pTarget: Double): Longint; stdcall;
VB: Function MCGetTargetEx(ByVal hCtrlr As Integer, ByVal axis As Integer, target As Double) As Long
LabVIEW:



MCCL Reference
TT

See Also
MCMoveAbsolute( ), **MCMoveRelative( )**

MCGetTorque

MCGetTorque( ) returns the current torque setting for the specified axis.

```
long int MCGetTorque(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* pTorque                // torque return value
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pTorque	Points to a double precision variable that will hold the torque.

Returns
MCGetTorque( ) places the torque setting in the variable specified by the pointer *pTorque* and a zero is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned, and the variable pointed to by *pTorque* is left unchanged.

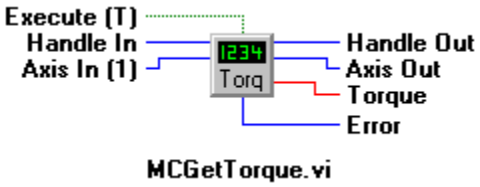
Comments

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
Torque mode is not supported on stepper axes, DCX-PCI100 controller, MC100, or MC110 modules.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes
Delphi: function MCGetTorque(hCtlr: HCTRLR; axis: Word; var pTorque: Double): Longint; stdcall;
VB: Function MCGetTorque(ByVal hCtrlr As Integer, ByVal axis As Integer, torque As Double) As Long
LabVIEW:



MCCL Reference
TQ

See Also
MCGetMotionConfigEx( ), **MCSetMotionConfigEx( )**, **MCSetTorque( )**, **MCMOTIONEX** structure definition

MCGetVectorVelocity

MCGetVectorVelocity( ) returns the current programmed velocity for the specified axis, in whatever units the axis is configured for.

```
long int MCGetVectorVelocity(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* pVelocity              // vector velocity return value
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pVelocity	Pointer to a double precision floating point variable that will hold the vector velocity value for the specified axis.

Returns
The position value is placed in the variable specified by the pointer *pVelocity* and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned, and the variable pointed to by *pVelocity* is left unchanged.

Comments
The vector velocity value for a particular *axis* may also be obtained using **MCGetContourConfig( )**. **MCGetVectorVelocity( )** provides a short-hand method for getting just the vector velocity value and is most useful when updating vector velocity settings on the fly.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.0 or higher

Prototypes
Delphi: function MCGetVectorVelocity(hCtlr: HCTRLR; axis: Word; var pVelocity: Double): Longint; stdcall;
VB: Function MCGetVectorVelocity(ByVal hCtrlr As Integer, ByVal axis As Integer, velocity As Double) As Long
LabVIEW: Not Supported

MCCL Reference
None

See Also
MCGetContourConfig( ), **MCSetVectorVelocity( )**

MCGetVelocityActual

MCGetVelocityActual( ) returns the current actual velocity for the specified axis, in whatever units the axis is configured for.

```
long int MCGetVelocityEx(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double* pVelocity              // velocity return value
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.
pVelocity	Pointer to a double precision floating point variable that will hold the velocity value for the specified axis.

Returns
The velocity value is placed in the variable specified by the pointer *pVelocity,* and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned, and the variable pointed to by *pVelocity* is left unchanged.

Comments
The actual velocity value for an *axis* is reported by most PMC controllers as the number of encoder counts during the most recent servo update period. See your motion controller's User's Manual for details.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.4 or higher

Prototypes
Delphi: function MCGetVelocityActual(hCtlr: HCTRLR; axis: Word; var pVelocity: Double): Longint; stdcall;
VB: Function MCGetVelocityActual(ByVal hCtrlr As Integer, ByVal axis As Integer, velocity As Double) As Long
LabVIEW: Not Supported

MCCL Reference
TV

See Also
MCSetVelocity( ), **MCSetMotionConfigEx( )**

MCGetVelocityEx

MCGetVelocityEx( ) returns the current programmed velocity for the specified axis, in whatever units the axis is configured for.

```
long int MCGetVelocityEx(
   HCTRLR hCtlr,                 // controller handle
   WORD axis,                    // axis number
   double* pVelocity             // velocity return value
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number to query.

pVelocity Pointer to a double precision floating point variable that will hold the velocity value for the specified axis.

Returns
The position value is placed in the variable specified by the pointer *pVelocity,* and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned, and the variable pointed to by *pVelocity* is left unchanged.

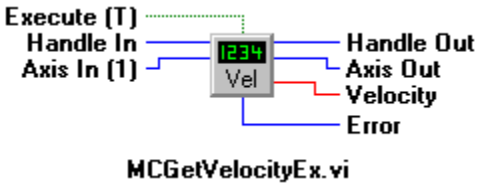
Comments
The programmed velocity value for a particular *axis* may also be obtained using the **MCGetMotionConfigEx( )** function. **MCGetVelocityEx( )** provides a short-hand method for getting just the velocity value and is most useful when updating velocity settings on the fly in velocity mode.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes
Delphi: function MCGetVelocityEx(hCtlr: HCTRLR; axis: Word; var pVelocity: Double): Longint; stdcall;
VB: Function MCGetVelocityEx(ByVal hCtrlr As Integer, ByVal axis As Integer, velocity As Double) As Long
LabVIEW:



MCCL Reference
Controller RAM Motor Tables

See Also
MCSetVelocity( ), **MCSetMotionConfigEx( )**

MCIsAtTarget

MCIsAtTarget( ) waits for the "At Target" condition to go true for the specified axis. Use it to determine when motion has completed for an axis.

```
long int MCIsAtTarget(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double timeout                 // timeout, in seconds
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number for which to wait for the "At Target" condition.
timeout	Time to wait, in seconds, for the At Target condition to go true.

Returns

This function returns TRUE, if the axis is "At Target." A return value of FALSE indicates the specified axis is not "At Target" by the end of *timeout*. If MC_ALL_AXES is specified for Axis, TRUE will be returned only if all axes are "At Target."

Comments

This function waits for up to *timeout* seconds for the At Target status of the axis to be TRUE. It returns as soon as the status goes TRUE or when *timeout* expires. Set *timeout* to zero to check the At Target status only once and return immediately (i.e. no wait is performed).

Compatibility

The DC2, DCX-PC, and DCX-PCI100 do not support the At Target status bit and should use **MCIsStopped( )** instead.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.2 or higher

Prototypes

Delphi:	function MCIsAtTarget(hCtlr: HCTRLR; axis: Word; timeout: Double): Longint; stdcall;
VB:	Function MCIsAtTarget(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal timeout As Double) As Long
LabVIEW:	Not Supported

MCCL Reference

None

See Also

MCIsStopped( )

MCIsDigitalFilter

MCIsDigitalFilter( ) is used to determine the enabled state of the digital filter mode.

```
long int MCIsDigitalFilter(
   HCTRLR hCtlr,                  // controller handle
   WORD axis                      // axis number
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number to query.

Returns

This function returns TRUE if the digital filter for the specified axis is enabled, or it returns FALSE if the digital filter is disabled.

Comments

This function is used to determine the enabled state of the digital filter mode supported by advanced motion control modules, such as the MC300.

You may not set the *axis* parameter to MC_ALL_AXES for this command.

Compatibility

The DC2, DCX-PC100, DCX-AT200, DCX-PCI100, MFX-PCI1000 controllers, MC360 and MC362 modules do not support digital filtering.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.1 or higher

Prototypes

Delphi:	function MCIsDigitalFilter(hCtlr: HCTRLR; axis: Word): Longint; stdcall;
VB:	Function MCIsDigitalFilter(ByVal hCtrlr As Integer, ByVal axis As Integer) As Long
LabVIEW:	Not Supported

MCCL Reference

None

See Also

MCEnableDigitalFilter( ), **MCGetCount( )**, **MCGetDigitalFilter( )**, **MCSetDigitalFilter( )**

MCIsEdgeFound

MCIsEdgeFound( ) waits for the "Edge Found" condition to go true for the specified axis. Use it to determine when an open-loop stepper motor homing sequence has detected the edge sensor.

```
long int MCIsEdgeFound(
   HCTRLR hCtlr,                 // controller handle
   WORD axis                     // axis number
   double timeout                // timeout, in seconds
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number for which to wait for the "Edge Found" condition.
timeout	Time to wait, in seconds, for the “Edge Found” condition to go true.

Returns

This function returns TRUE if the stepper axis has detected the edge input or FALSE if the axis has not detected the edge input by the end of *timeout*.

Comments

This function waits for up to *timeout* seconds for the Edge Found status of a stepper motor axis to go TRUE. It returns as soon as the status goes TRUE or when *timeout* expires. Set *timeout* to zero to check the edge found status only once and return immediately (i.e. no wait is performed). This function uses **MCDecodeStatusEx( )** internally to test the MC_STAT_EDGE_FOUND status bit.

Compatibility

The DC2, DCX-PC100, and DCX-AT200 controllers do not support this function. Stepper modules when run in closed-loop mode do not support this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.2 or higher

Prototypes

Delphi:	function MCIsEdgeFound(hCtlr: HCTRLR; axis: Word; timeout: Double): Longint; stdcall;
VB:	Function MCIsEdgeFound(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal timeout As Double) As Long
LabVIEW:	Not Supported

MCCL Reference

TS

See Also

MCDecodeStatusEx( ), **MCEdgeArm( )**, **MCWaitForEdge( )**

MCIsIndexFound

MCIsIndexFound( ) waits for the "Index Found" condition to go true for the specified axis. Use it to determine when a servo or closed-loop stepper motor homing sequence has detected the encoder index.

```
long int MCIsIndexFound(
   HCTRLR hCtlr,                 // controller handle
   WORD axis                     // axis number
   double timeout                // timeout, in seconds
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number for which to wait for the "Index Found" condition.
timeout	Time to wait, in seconds, for the “Index Found” condition to go true.

Returns

This function returns TRUE if the servo axis has detected the encoder index or FALSE if the axis has not detected the encoder index by the end of *timeout*.

Comments

This function waits for up to *timeout* seconds for the Index Found status of a servo motor axis to go TRUE. It returns as soon as the status goes TRUE or when *timeout* expires. Set *timeout* to zero to check the encoder index status only once and return immediately (i.e. no wait is performed). This function uses **MCDecodeStatusEx( )** internally to test the MC_STAT_INDEX_FOUND status bit.

Compatibility

The DC2, DCX-PC100, and DCX-AT200 controllers do not support this function. Stepper modules when run in open-loop mode with an auxiliary encoder do not support primary encoder functions such as this.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 3.2 or higher

Prototypes

Delphi:	function MCIsIndexFound(hCtlr: HCTRLR; axis: Word; timeout: Double): Longint; stdcall;
VB:	Function MCIsIndexFound(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal timeout As Double) As Long
LabVIEW:	Not Supported

MCCL Reference

TS

See Also

MCDecodeStatusEx( ), **MCIndexArm( )**, **MCWaitForIndex( )**

MCIsStopped

MCIsStopped( ) waits for the "Trajectory Complete" condition to go true for the specified axis. Use it to determine when motion has completed for an axis.

```
long int MCIsStopped(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // axis number
   double timeout                 // timeout, in seconds
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
axis	Axis number for which to wait for the "Trajectory Complete" condition.
timeout	Time to wait, in seconds, for the Trajectory Complete condition to go true.

Returns

This function returns TRUE if the axis is "Trajectory Complete." A return value of FALSE indicates the specified axis is not "Trajectory Complete" by the end of *timeout*. If MC_ALL_AXES is specified for Axis, TRUE will be returned only if all axes are "Trajectory Complete."

Comments

This function waits for up to *timeout* seconds for the Trajectory Complete status of the axis to be TRUE. It returns as soon as the status goes TRUE or when *timeout* expires. Set *timeout* to zero to check the Trajectory Complete status only once and return immediately (i.e. no wait is performed).

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.2 or higher

Prototypes

Delphi:	function MCIsStopped(hCtlr: HCTRLR; axis: Word; timeout: Double): Longint; stdcall;
VB:	Function MCIsStopped(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal timeout As Double) As Long
LabVIEW:	Not Supported

MCCL Reference

None

See Also

MCIsAtTarget( )

MCTranslateErrorEx

MCTranslateErrorEx( ) translates numeric error codes into ASCII text messages.

```
long int MCTranslateErrorEx(
   short int error,              // error code to translate
   char* buffer,                 // character buffer for message
   long int length               // length of Buffer, in bytes
);
```

Parameters

error	Numeric error code to translate.
buffer	String buffer to hold ASCII error message.
length	Length of string buffer (in bytes).

Returns

This function returns a pointer to the ASCII error message corresponding to Error. If Error does not correspond to a valid error message, a NULL pointer is returned. It will work with errors returned from **MCGetError( )** and **MCErrorNotify( )** error messages.

Comments

Beginning with version 2.1 of the MCAPI this function is included as a native MCAPI function (previously it was contained in a separate module). Incorporating **MCTranslateErrorEx( )** into the MCAPI DLL will facilitate future updates, but has required changes from how It previously worked. The string buffer and buffer length have been added to the argument list. These changes make it possible to call **MCTranslateErrorEx( )** from a much wider range of programming languages.

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes

Delphi: function MCTranslateErrorEx(error: SmallInt; buffer: PChar; length: Longint): Longint; stdcall;

VB: Function MCTranslateErrorEx(ByVal error As Integer, ByVal buffer As String, ByVal length As Long) As Long

LabVIEW:

MCCL Reference

None

See Also

MCErrorNotify( ), **MCGetError( )**

Chapter Contents

Chapter

8

I/O Functions

Digital I/O functions allow configuration of high or low “true” states, reading of inputs, sequencing based on input, and setting outputs. Analog I/O functions control the input and output of analog values through A/D and D/A ports installed on the controller.

A word of caution must be given regarding the use of board-level sequencing commands. Even though a warning is included with **MCWaitForDigitalIO( ),** it should be stressed that once this command is called, the board will not accept another command nor will it respond to the calling program until the board has completed what it was initially told to do. This can lead to scenarios where the calling program has absolutely no control during potentially dangerous or otherwise expensive situations.

To see examples of how the functions in this chapter are used, please refer to the online Motion Control API Reference.

MCConfigureDigitalIO

MCConfigureDigitalIO( ) configures a specific digital I/O channel for input or output and for high or low true logic.

```
short int MCConfigureDigitalIO(
   HCTRLR hCtlr,                  // controller handle
   WORD channel,                  // channel number
   WORD mode                      // configuration flags
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
channel	Digital channel number to configure.

mode Specifies how the channel is to be configured. This parameter may be any one of the digital I/O flags listed below. An input/output flag and a logic level flag may be OR'ed together.

Value	Description
MC_DIO_INPUT	Configures the channel for input.
MC_DIO_OUTPUT	Configures the channel for output.
MC_DIO_LOW	Configures the channel for negative logic level.
MC_DIO_HIGH	Configures the channel for positive logic level.
MC_DIO_LATCH	Configures the (input) channel for latched operation.

Returns

The return value is TRUE if the function is successful. A return value of FALSE indicates **MCConfigureDigitalIO( )** was unable to configure the channel as requested.

Comments

Each digital I/O channel may be configured for input or for output. The logic level maps the logical "on" and "off" states of the channel to the physical input and output voltages for that channel. If the channel is set to MC_DIO_LOW (negative logic) the "on" state of a channel will represent a low voltage (<0.4VDC) and "off" a high voltage (>2.4VDC). When set to MC_DIO_HIGH (positive logic) the "on" state of a channel will represent a high voltage (>2.4VDC) and "off" a low voltage (<.0.4VDC).

On the DC2-STN controller, beginning with firmware release 1.2a, it is possible to configure an input channel to "latch" input events (see the controller manual for details of signal hold time, etc.). Configure an input channel using the MC_DIO_LATCH constant to enable latching or clear the latched state. Configure an input channel using the MC_DIO_INPUT constant to disable latching.

The DCX-PCI motherboard has 16 general I/O, consisting of 8 fixed inputs and 8 fixed outputs. Since these digital I/O are fixed, they may not be configured for input or output. A program may verify the functionality (input or output) of a channel by using **MCGetDigitalIOConfig( )** to check the current configuration.

Under the MCAPI, the DC2-STN controller's input channels are numbered 1 - 8, and the output channels are numbered 9 - 16 (the MCAPI requires that each channel have a unique channel number).

Compatibility

MC_DIO_INPUT and MC_DIO_OUTPUTare not supported by MFX-PCI1000.

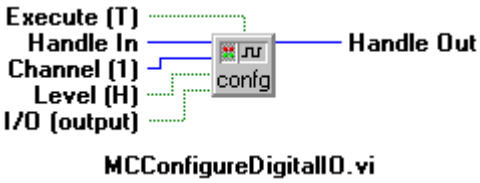
Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: function MCConfigureDigitalIO(hCtlr: HCTRLR; channel, mode: Word): SmallInt;
VB: Function MCConfigureDigitalIO (ByVal hCtrlr As Integer, ByVal channel As Integer, ByVal mode As Integer) As Integer

LabVIEW:



MCCL Reference
CH, CI, CL, CT

See Also
MCEnableDigitalIO( ), **MCGetDigitalIO( )**, **MCGetDigitalIOConfig( )**

MCEnableDigitalIO

MCEnableDigitalIO( ) turns the specified digital I/O channel on or off.

```
void MCEnableDigitalIO(
   HCTRLR hCtlr,                  // controller handle
   WORD channel,                  // channel number
   short int state                // enable state
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
channel Digital channel number to enable.
state Specifies whether the channel is to be turned on or turned off.

Value	Description
TRUE	Turns the channel on.
FALSE	Turns the channel off.

Returns
This function does not return a value.

Comments
The I/O channel selected by *hCtlr* and *channel* must have previously been configured for output using the **MCConfigureDigitalIO( )** command. Note that depending upon how a channel has been configured "on" (and conversely "off") may represent either a high or a low voltage level.

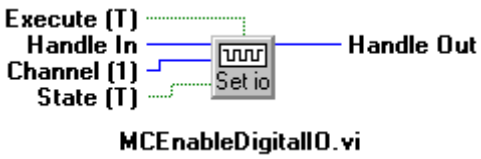
state will accept any non-zero value as TRUE, and will work correctly with most programming languages, including those that define TRUE as a non-zero value other than one (one is the Windows default value for TRUE).

Under the MCAPI, the DC2-STN controller's input channels are numbered 1 - 8, and the output channels are numbered 9 - 16 (the MCAPI requires that each channel have a unique channel number).

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCEnableDigitalIO(hCtlr: HCTRLR; channel: Word; state: SmallInt); stdcall;
VB: Sub MCEnableDigitalIO(ByVal hCtrlr As Integer, ByVal channel As Integer, ByVal state As Integer)
LabVIEW:



MCCL Reference
CF, CN

See Also
MCConfigureDigitalIO( ), **MCEnableDigitalIO( )**, **MCGetDigitalIOConfig( )**, **MCPARAMEX** structure definition

---

MCGetAnalogEx

MCGetAnalog( ) reads the current input state of the specified input Channel.

```
WORD MCGetAnalog(
   HCTRLR hCtlr,                  // controller handle
   WORD channel                   // channel number
   DWORD value                    // channel number
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
channel	Analog channel number to read from.
value	Pointer to a variable that will contain the analog reading when **MCGetAnalogEX( )** returns.

Returns
This function returns the current A/D reading for *channel*.

Comments
PMC motion controllers typically include four undedicated analog input channels. On older controllers these inputs are 8-bit, the newer Multiflex series of controllers is typically configured with 14-bit inputs. By default these channels are assigned channel numbers 1 to 4.

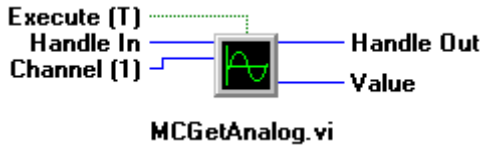
Additional analog input/output channels supplied by MC500 modules will occupy sequential channel numbers beginning with channel 5. The fields **AnalogInput** and **AnalogOutput** in the **MCPARAMEX** structure contain the number of input and output channels the controller is configured for.

MCGetAnalogEx( ) should be used for new designs.

Compatibility
There are no compatibility issues with this function, however, please note that the DCX-PCI controllers have no built-in analog inputs and for the MFX-PCI1000 analog inputs are an option.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.0 or higher (3.4 or higher for **MCGetAnalogEX( )**)

Prototypes
Delphi: function MCGetAnalogEx(hCtlr: HCTRLR; channel: Word): Word; stdcall;
VB: Function MCGetAnalogEx(ByVal hCtrlr As Integer, ByVal channel As Integer) As Integer
LabVIEW:



MCCL Reference
TA

See Also
MCSetAnalog( ), **MCPARAMEX** structure definition

MCGetDigitalIO

MCGetDigitalIO( ) returns the current state of the specified digital I/O channel.

```
short int MCGetDigitalIO(
   HCTRLR hCtlr,                  // controller handle
   WORD channel                   // channel number
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
channel Digital channel number to get state of.

Returns
The return value is TRUE if the channel is "on." A return value of FALSE indicates the channel is "off".

Comments
This function will read the current state of both input and output digital I/O channels. Note that this function simply reports if the channel is "on" or "off"; depending upon how a channel has been configured "on" (and conversely "off") may represent either a high or a low voltage level.

The field DigitalIO in the **MCPARAMEX** structure contains the total number of digital I/O channels the controller is configured for.

Under the MCAPI, the DC2-STN controller's input channels are numbered 1 - 8, and the output channels are numbered 9 - 16 (the MCAPI requires that each channel have a unique channel number).

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: function MCGetDigitalIO(hCtlr: HCTRLR; channel: Word): SmallInt; stdcall;
VB: Function MCGetDigitalIO(ByVal hCtrlr As Integer, ByVal channel As Integer) As Integer
LabVIEW:

MCCL Reference
TC

See Also
MCEnableDigitalIO( ), **MCGetDigitalIO( )**, **MCGetDigitalIOConfig( )**

MCGetDigitalIOConfig

MCGetDigitalIOConfig( ) returns the current configuration (in / out / high / low) of the specified digital I/O channel.

```
short int MCGetDigitalIO(
   HCTRLR hCtlr,                // controller handle
   WORD channel,                // channel number
   WORD* pMode                  // variable to hold the channel settings
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.

channel Digital channel number to get configuration of.

pMode Pointer to a variable to hold the current configuration settings of the specified channel. This variable will contain one or more of the following flags on return:

Value	Description
MC_DIO_INPUT	Channel configured for input.
MC_DIO_OUTPUT	Channel configured for output.
MC_DIO_LOW	Channel configured for low true logic level.
MC_DIO_HIGH	Channel configured for high true logic level.
MC_DIO_LATCH	Input channel configured for latched operation.
MC_DIO_FIXED	Channel is a fixed input or output and cannot be changed using **MCConfigureDigitalIO( )**.
MC_DIO_LATCHABLE	Input channel is capable of latched operation.
MC_DIO_STEPPER	Input channel has been dedicated to driving a stepper motor (DC2-PC or DC2-STN).

Returns

The current configuration of the specified digital I/O channel is placed in the variable specified by the pointer *pMode*, and MCERR_NOERROR is returned if there were no errors. If there was an error, one of the MCERR_xxxx error codes is returned, and the variable pointed to by *pMode* is left unchanged.

Comments

The configuration of the specified channel is returned as one or more of the MC_DIO_xxx constants OR'ed together. This value is identical to the value you would create to configure the channel using **MCConfigureDigitalIO( )**, with the exception of the MC_DIO_FIXED, MC_DIO_LATCHABLE, and MC_DIO_STEPPER which are read-only (i.e. **MCGetDigitalIOConfig( )** only) parameters.

Currently none of the motion controllers supported by the MCAPI allow you to read back the configuration of the digital I/O. To implement **MCGetDigitalIOConfig( )** the MCAPI "remembers" any changes made to the digital I/O using **MCConfigureDigitalIO( )**. When the MCAPI DLL is loaded into memory (at application run time), it assumes the default state power-on state for all the installed digital I/O. Therefore, this function is most useful within a single application, after you have explicitly configured each I/O channel.

The field **DigitalIO** in the **MCPARAMEX** structure contains the total number of digital I/O channels the controller is configured for.

Under the MCAPI, the DC2-STN controller's input channels are numbered 1 - 8, and the output channels are numbered 9 - 16 (the MCAPI requires that each channel have a unique channel number).

Compatibility
MC_DIO_INPUT and MC_DIO_OUTPUTare not supported by MFX-PCI1000.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes
Delphi: function MCGetDigitalIOConfig(hCtlr: HCTRLR; channel: Word; var pMode: Word): LongInt; stdcall;
VB: Function MCGetDigitalIOConfig(ByVal hCtrlr As Integer, ByVal channel As Integer, mode As Integer) As Long
LabVIEW: Not Supported

MCCL Reference
None

See Also
MCConfigureDigitalIO( ), **MCEnableDigitalIO( )**, **MCPARAMEX** structure definition

MCSetAnalogEx

MCSetAnalogEx( ) sets the voltage level of the specified general purpose analog output Channel.

```
void MCSetAnalog(
   HCTRLR hCtlr,                  // controller handle
   WORD channel,                  // channel number
   DWORD value                    // new output value
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
channel	Analog output channel number to set
value	New output value.

Returns
MCSetAnalogEx( ) returns the value MCERR_NOERROR if the function completed without errors. If there was an error one of the MCERR_xxxx error codes is returned.

Comments

Analog output ports on MC500 and MC520 Analog Modules accept values in the range of 0 to 4095 counts (12 bits). This range of values corresponds to an output voltage of 0 to 5V or -10 to +10V, depending upon how the output is configured (see your controller's hardware manual). Each digital bit corresponds to a voltage level as follows:

Output Used	Volts per Count
0 to 5V	0.0012V
-10 to +10V	0.0049V

Compatibility

Analog output channels are not supported by the DC2-PC100 dedicated 2 axis controllers or the MultiFlex family of controllers.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 3.4 or higher

Prototypes

Delphi: procedure MCSetAnalogEx(hCtlr: HCTRLR; channel, value: DWord); stdcall;
VB: Sub MCSetAnalogEx(ByVal hCtrlr As Integer, ByVal channel As Integer, ByVal value As Integer)
LabVIEW: Not Supported

MCCL Reference

OA

See Also

MCGetAnalogEx( )

MCWaitForDigitalIO

MCWaitForDigitalIO( ) waits for the specified digital I/O channel to go on or off, depending upon the value of *state*.

```
void MCWaitForDigitalIO(
   HCTRLR hCtlr,                // controller handle
   WORD channel,                // digital I/O channel to watch
   short int state              // state of channel to watch for
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
channel Digital channel number to wait for.
state Selects state of channel to wait for:

Value	Description

TRUE	Wait for channel to go "on."
FALSE	Wait for channel to go "off."

Returns
This function does not return a value.

Comments
Digital channels 1 to 16 are built into each controller. Additional digital channels, beginning with channel 17, may be added in blocks of 16 channels using MC400 Digital I/O Modules. The field **DigitalIO** in the **MCPARAMEX** structure contains the total number of digital channels installed on the controller.

Once this command is issued, the calling program will not be able to communicate with the board until the digital I/O is equal to *state*. We recommend creating your own looping structure based on **MCGetDigitalIO( )** instead.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCWaitForDigitalIO(hCtlr: HCTRLR; channel: Word; state: SmallInt); stdcall;
VB: Sub MCWaitForDigitalIO(ByVal hCtrlr As Integer, ByVal channel As Integer, ByVal state As Integer)
LabVIEW:

MCCL Reference
WF, WN

See Also
MCConfigureDigitalIO( ), **MCEnableDigitalIO( )**, **MCGetDigitalIO( )**, **MCPARAMEX** structure definition

Chapter Contents

Chapter

9

Macro’s and Multi-Tasking Functions

Macro and multi-tasking functions provide access to the motion controllers on-board macro capability, as well as the multitasking features of advanced controllers.

To see examples of how the functions in this chapter are used, please refer to the online Motion Control API Reference.

MCCancelTask

MCCancelTask( ) cancels an executing task on a multi-tasking controller. The task should have been previously started with an **MCBlockBegin( )** / **MCBlockEnd( )** pair.

```
long int MCCancelTask(
   HCTRLR hCtlr,                // controller handle
   long int taskID              // ID of task to cancel
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
taskID	Task ID value for the task to be stopped. This value was returned by the **MCBlockEnd( )** function when the task was generated.

Returns

This function returns MCERR_NOERROR if there were no errors. One of the MCERR_xxxx defined error codes will be returned if there was a problem.

Comments

MCCancelTask( ) is the only way to stop tasks that are not programmed to stop themselves (i.e. infinite loop tasks).

See the description of **MCBlockBegin( )** for more information and reference the online help for examples.

Compatibility
The DC2 and DCX-PC100 controllers do not support background tasks.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes
Delphi: function MCCancelTask(hCtlr: HCTRLR; taskID: Longint): Longint; stdcall;
VB: Function MCCancelTask(ByVal hCtrlr As Integer, ByVal taskID As Long) As Long
LabVIEW: Not Supported

MCCL Reference
ET

See Also
MCBlockBegin( ), **MCCancelTask( )**

MCMacroCall

MCMacroCall( ) causes a previously loaded macro to be executed.

```
void MCMacroCall(
   HCTRLR hCtlr,                  // controller handle
   WORD macro                     // macro number
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
macro Macro number to execute.

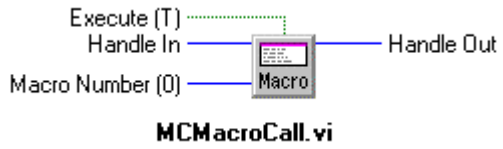
Returns
This function does not return a value.

Comments
Macros are normally downloaded using the **pmcputs( )** ASCII interface command, using the Motion Control Command Language (MCCL); or by converting the MCAPI functions to a macro with the **MCBlockBegin( )** / **MCBlockEnd( )** functions. These controller level macros are often the only efficient way to implement hardware specific sequences, such as special homing routines, initializing encoder positions, etc.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: procedure MCMacroCall(hCtlr: HCTRLR; macro: Word); stdcall;
VB: Sub MCMacroCall(ByVal hCtrlr As Integer, ByVal macro As Integer)
LabVIEW:



MCCL Reference
MC

See Also
MCBlockBegin( ), **MCBlockEnd( )**, **pmcputs( )**, Controller hardware manual

---

MCRepeat

MCRepeat( ) inserts a repeat command into a block command - task, compound command, or macro.

```
long int MCRepeat(
   HCTRLR hCtlr,                  // controller handle
   long int count                 // repeat count
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
count Repeat count. Commands that precede the **MCRepeat( )** in the block command will be repeated *count* more times (for a total execution of *count* + 1).

Returns
MCRepeat( ) returns the value MCERR_NOERROR if the function completed without errors. If there was an error, one of the MCERR_xxxx error codes is returned.

Comments
This function may only be used within an **MCBlockBegin( )** / **MCBlockEnd( )** command pair.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib

Version: MCAPI 1.3 or higher

Prototypes

Delphi: function MCRepeat(hCtlr: HCTRLR; count: Longint): Longint; stdcall;
VB: Function MCRepeat(ByVal hCtrlr As Integer, ByVal count As Long) As Long
LabVIEW: Not Supported

MCCL Reference

RP

See Also

MCBlockBegin( ), **MCBlockEnd( )**

Chapter Contents

Chapter 10

MCAPI Driver Functions

Driver functions handle driver related housekeeping, and as such do not directly affect the controller.

To see examples of how the functions in this chapter are used, please refer to the online Motion Control API Reference.

MCBlockBegin

MCBlockBegin( ) initiates a block command sequence. All commands up to the subsequent **MCBlockEnd( )** will be included in the block. Block commands include compound commands, macro definition commands, contour path motions, and tasks on multitasking controllers.

```
long int MCBlockBegin(
   HCTRLR hCtlr,                  // controller handle
   long int mode,                 // block mode type
   long int num                   // macro / task number / controlling axis
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
mode Type of block command to begin:

Value	Description
MC_BLOCK_COMPOUND	Specifies that this block is a compound command.
MC_BLOCK_TASK	Specifies this block as an individual task on multitasking controllers. *num* should be set to the desired task number.
MC_BLOCK_MACRO	Specifies this block as a macro definition. *num* should be set to the desired macro number for this macro.

Value	Description
MC_BLOCK_RESETM	Resets macro memory. Setting *num* to zero resets all macros (and works with all controllers), *num* may also be set to 1 or 2 on the DCX AT200 to selectively delete ram or flash based macros.
MC_BLOCK_CANCEL	Cancels a block command without sending any commands to the controller.
MC_BLOCK_CONTR_USER	Specifies that this block is a user defined contour path motion. *num* should be set to the controlling axis number.
MC_BLOCK_CONTR_LIN	Specifies that this block is a linear contour path motion. *num* should be set to the controlling axis number.
MC_BLOCK_CONTR_CW	Specifies that this block is a clockwise arc contour path motion. *num* should be set to the controlling axis number.
MC_BLOCK_CONTR_CCW	Specifies that this block is a counter clockwise arc contour path motion. *num* should be set to the controlling axis number.

num Specifies the macro number for macro blocks, the task number for task blocks, the controlling axis for contour blocks, or the macro types for macro reset.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

The **MCBlockBegin( )** and **MCBlockEnd( )** commands are used to bracket other API commands in order to affect how those commands are executed. While the high level MCAPI is function based (as are most Windows APIs), PMC's motion control cards are command based. They are capable of accepting single commands or blocks of commands, depending upon the complexity of the motion. To provide the same block functionality to the MCAPI the **MCBlockBegin( )** and **MCBlockEnd( )** functions where created. These functions may be used to bracket one or more MCAPI function calls to create function blocks.

One use is to create a compound command block - where multiple commands are sent to the controller as a single block. This is useful for data capture sequences, homing sequences, or anywhere you want to synchronize a complex group of commands.

For multi-tasking controllers, the block commands can be used to group individual commands as separate tasks. Multi-tasking permits multiple user programs to run in parallel on PMC's advanced motion control cards. Multi-tasking also permits you to run command sequences that would normally lock-up the controller's command interpreter in the background, thus leaving the command interpreter unaffected.

A third use of the block commands is to store the bracketed command sequence as a macro. Macros may be replayed at any time using the **MCMacroCall( )** function. Please note that API commands that read data from a controller, such as any of the **MCGet…** functions, should not be included in macros. Macro memory may be reset (cleared) by calling **MCBlockBegin( )** with Mode set to

MC_BLOCK_RESETM. If your controller allows you to reset selected blocks of macros you may specify this by setting *num* to 1 for RAM-based macros or 2 for Flash memory macros.

All calls to **MCBlockBegin( )**, except those with a *mode* of MC_BLOCK_RESETM or MC_BLOCK_CANCEL require a corresponding call to **MCBlockEnd( )**. Calls to **MCBlockBegin( )** may not be nested, except that **MCBlockBegin( )** calls with an Mode of MC_BLOCK_CANCEL may be included within other **MCBlockBegin( )** blocks (this call terminates the outer **MCBlockBegin( )**, so no **MCBlockEnd( )** is needed in this case).

Beginning with version 2.0 of the MCAPI, blocks are also used for multi-axis contouring. Contouring requires first that the selected axes be placed in contouring mode and a controlling axis specified. This is done with the **MCSetOperatingMode( )** function. Then blocks of contour path moves are issued. Under the MCAPI, these contour path blocks are specified by bracketing **MCArcCenter( )**, **MCGoHome( )**, **MCMoveAbsolute( )**, **MCMoveRelative( )**, or **MCSetVectorVelocity( )** with block commands that are one of the MC_BLOCK_CONTR_xxx types.

Block commands may be canceled prior to issuing an **MCBlockEnd( )** by calling **MCBlockBegin( )** with Mode set to MC_BLOCK_CANCEL.

Compatibility

The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers. The DC2 and DCX-PC100 controllers do not support background tasks.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes

Delphi: function MCBlockBegin(hCtlr: HCTRLR; mode, num: Longint): Longint; stdcall;
VB: Function MCBlockBegin(ByVal hCtrlr As Integer, ByVal mode As Long, ByVal num As Long) As Long
LabVIEW: Not Supported

MCCL Reference

CP, GT, MD, RM

See Also

MCBlockEnd( ), **MCCancelTask( )**, **MCMacroCall( )**, **MCRepeat( )**

MCBlockEnd

MCBlockEnd( ) ends a block command and transmits the compound command, task, macro, or contour path to the controller.

```
long int MCBlockEnd(
   HCTRLR hCtlr,                 // controller handle
   long int* pTaskID             // task ID for MC_BLOCK_TASK blocks
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.

pTaskID Pointer to variable to hold the Task ID value for MC_BLOCK_TASK blocks, this parameter is ignored and may be set to NULL for MC_BLOCK_COMPOUND or MC_BLOCK_MACRO blocks. Setting this parameter to NULL for MC_BLOCK_TASK will cause the function to not return the Task ID for this task.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

The **MCBlockBegin( )** and **MCBlockEnd( )** commands are used to bracket other API commands in order to affect how those commands are executed.

See the description of **MCBlockBegin( )** for more information.

Compatibility

The MCAPI does not does not support contouring on the DC2, DCX-PC100, or DCX-PCI100 controllers. The DC2 and DCX-PC100 controllers do not support background tasks.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes

Delphi: function MCBlockEnd(hCtlr: HCTRLR; var pTaskID: LongInt): Longint; stdcall;
VB: Function MCBlockEnd(ByVal hCtrlr As Integer, taskID As Long) As Long
LabVIEW: Not Supported

MCCL Reference

None

See Also

MCBlockBegin( ), **MCCancelTask( )**

MCClose

MCClose( ) closes the specified motion controller handle, and is typically called at the end of a program.

```
short int MCClose(
   HCTRLR hCtlr                  // controller handle
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

Following a call to **MCClose( )**, no further calls should be made to the Motion Control API functions with this handle (the exception being **MCOpen( )**, which may be called to open or reopen the API at any time).

By calling **MCClose( )** you notify Windows that you are done with the controller and device driver. When the last user has closed the driver Windows is then free to unload the driver from memory. Failure to call close leaves the handle open, reducing the number of available controller handles for other applications.

Compatibility

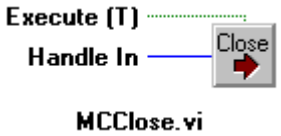
There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: function MCClose(hCtlr: HCTRLR): SmallInt; stdcall;
VB: Function MCClose(ByVal hCtrlr As Integer) As Integer
LabVIEW:



MCCL Reference

None

See Also

MCOpen( )

MCGetConfigurationEx

MCGetConfigurationEx( ) obtains the configuration for the specified controller. Configuration information includes the controller type, number and type of installed motor modules, and if the controller supports scaling, contouring, etc.

```
long int MCGetConfigurationEx(
   HCTRLR hCtlr,                 // controller handle
   MCPARAMEX* pParam             // address of extended configuration
                                 // structure
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
pParam	Points to an **MCPARAMEX** structure that receives the configuration information for *hCtlr*.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

This function allows the application to query the driver about installed controller hardware and capabilities. Included are the number and type of axes, digital and analog IO channels, scaling, and contouring.

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 3.0 or higher

Prototypes

Delphi: function MCGetConfigurationEx(hCtlr: HCTRLR; var pParam: MCPARAMEX): LongInt; stdcall;
VB: Function MCGetConfigurationEx(ByVal hCtrlr As Integer, param As MCParamEx) As Long
LabVIEW: Not Supported

MCCL Reference

Dual Port RAM

See Also

MCPARAMEX structure definition

MCGetVersion

MCGetVersion( ) returns version information about the MCAPI.DLL and, optionally, about the device driver in use for a particular controller.

```
DWORD MCGetVersion(
   HCTRLR hCtlr                  // controller handle
);
```

Parameters

hCtlr Controller handle, selects which motion controller to obtain device driver version info from. May be NULL (if NULL **MCGetVersion( )** version number info is returned for the MCAPI DLL only).

Returns

The return version number for the MCAPI DLL and, if *hCtlr* is not NULL, the version number for the device driver in use for the controller. If *hCtlr* is NULL, device driver version info will be zero.

Comments

The DLL version number is contained in the low order word of the return value. The major version number is stored as the low order byte of this word, while the release number is multiplied by 10, added to the revision number, and stored as the high order byte.

If the controller handle is not NULL, the version information for the device driver that is associated with this controller will be placed in the high order word of the return value, using the same format as was used for the DLL version information.

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.2 or higher

Prototypes

Delphi: function MCGetVersion(hCtlr: HCTRLR): Longint; stdcall;
VB: Function MCGetVersion(ByVal hCtrlr As Integer) As Long
LabVIEW: Not Supported

MCCL Reference

None

MCOpen

MCOpen( ) returns a handle to a particular controller for use with subsequent API calls.

```
HCTRLR MCOpen(
   short int id,                  // controller ID
   WORD mode,                     // open mode - ASCII / binary
   char* pName                    // not used
);
```

Parameters

id Controller ID, selects the controller to open.

mode I/O mode to open controller in:

Value	Description
MC_OPEN_ASCII	Open controller for ASCII (character) I/O.
MC_OPEN_BINARY	Open the binary command interface of the controller.
MC_OPEN_EXCLUSIVE	May be OR'ed with MC_OPEN_ASCII or MC_OPEN_BINARY to request exclusive access to the controller.

pName Should be set to NULL for the present

Returns

This function returns handle to the specified controller for use in subsequent API calls. The handle will be greater than zero if the open call succeeds or less than zero if there is an error. Standard error codes (see the file MCERR.H) will be multiplied by -1 to make their values negative and returned in place of a handle, if there is an error:

Value	Description
MCERR_ALLOC_MEM	Unable to allocate memory for handle.
MCERR_CONSTANT	The constant value supplied for *mode* is invalid.
MCERR_INIT_DRIVER	Unable to initialize device driver.
MCERR_MODE_UNAVAIL	The requested mode (ASCII or binary) is unavailable. Typically due to the fact that another process has an open handle to the controller in the opposite mode.
MCERR_NO_CONTROLLER	No controller is installed at this ID, run MCSETUP.
MCERR_NOT_PRESENT	The specified controller hardware is missing or not responding.
MCERR_OPEN_EXCLUSIVE	Unable to open controller for exclusive use - another process must already have an open handle to this controller.

MCERR_OUT_OF_HANDLES	The driver is out of handles, try closing unused handles first.
MCERR_RANGE	Specified *id* is out of range.
MCERR_UNSUPPORTED_MODE	The requested open mode (ASCII or binary) is not supported for this controller.

Please note that the error codes in the table above, when an error has occurred, will returned as a negative value.

Comments

Always save the handle returned by **MCOpen( )** and use that value in subsequent calls to the API. **MCOpen( )** must be called before any other API calls are attempted. If a call is made to any other API function with a bad handle, a handle error message (MCERR_CONTROLLER) will be broadcast to all windows. Everyone is notified in the case of a bad handle because the MCAPI normally uses the handle to route error messages, and obviously can't do this if the handle is invalid.

If it is necessary that no one else gains access to a controller while you are using it, you may combine the open mode with MC_OPEN_EXCLUSIVE:

```
if ((hCtlr = MCOpen( 7, MC_OPEN_ASCII | MC_OPEN_EXCLUSIVE, NULL )) > 0)
{
   // got an exclusive handle
}
```

will only return a valid handle if no other process has an open handle to this controller already, and will prevent any one else from opening the controller while the exclusive handle is open.

The name argument in the **MCOpen( )** function call is for future enhancements to the API and should be set to NULL for the present.

If you are using an DCX-AT or DCX-PCI configured for multi-interface, you may open binary and ASCII handles simultaneously. Exclusive handles are interface based, not controller based, in this case (i.e. you may have one exclusive ASCII handle and one exclusive binary handle open at the same time).

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: function MCOpen(id: SmallInt; mode: Word; pName: PChar): HCTRLR; stdcall;
VB: Function MCOpen(ByVal id As Integer, ByVal mode As Integer, ByVal name As String) As Integer

LabVIEW:

MCCL Reference
None

See Also
MCClose( ), **MCErrorNotify( )**

MCReopen

MCReopen( ) may be used to change the mode of an existing handle.

```
long int MCReopen(
   HCTRLR hCtlr,                  // controller handle
   WORD mode                      // new mode
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
mode New mode flags:

Value	Description
MC_OPEN_ASCII	Open controller for ASCII (character) I/O.
MC_OPEN_BINARY	Open the binary command interface of the controller.
MC_OPEN_EXCLUSIVE	May be combined with MC_OPEN_ASCII or MC_OPEN_BINARY using the binary or operator '|' to request exclusive access.

Returns
MCReopen( ) returns the value MCERR_NOERROR, if the function completed without errors. If there was an error, one of the MCERR_xxxx error codes is returned.

Comments
The most likely cause for failure is that another open handle exists for the same controller. **MCReopen( )** cannot change a controller's open mode if there are multiple handles, as there is no way to notify the owners of those other handles that a mode switch has occurred. If you plan on using this function in an application, it is suggested that you open the controller in exclusive mode to prevent any additional handles from being opened.

If you are using a DCX-PCI or DCX-AT in multi-interface mode, the above restrictions do not apply.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes
Delphi: function MCReopen(hCtlr: HCTRLR; mode: Word): Longint; stdcall;
VB: Function MCReopen(ByVal hCtrlr As Integer, ByVal mode As Integer) As Long
LabVIEW: Not Supported

MCCL Reference
None

See Also
MCClose( ), **MCOpen( )**

MCSetTimeoutEx

MCSetTimeoutEx( ) sets the timeout period for I/O to a particular controller.

```
long int MCSetTimeoutEx(
   HCTRLR hCtlr,                // controller handle
   double timeout,              // new timeout value
   double* pOldTimeout          // old timeout value
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
timeout	New timeout period, in seconds.
pOldTimeout	Pointer to a double precision floating point variable that will hold the old timeout setting for the specified axis. If the pointer is NULL, no value is returned.

Returns
If there were no errors, the previous timeout setting is placed in the variable specified by the pointer *pOldTimeout,* and MCERR_NOERROR is returned. If there was an error, one of the MCERR_xxxx error codes is returned, and the variable pointed to by *pOldTimeout* is left unchanged. If the pointer *pOldTimeout* is NULL, the old timeout value is not returned.

Comments
The timeout period is the maximum amount of time, in seconds, that the MCAPI device driver will wait to send a command and/or receive a reply. The default setting for timeout for all controllers is zero seconds. A timeout setting of zero will cause the controller to wait forever (i.e. no timeout) for I/O to complete.

Note that a timeout value that is acceptable for most functions may fail (i.e. timeout) if the controller is asked to perform a lengthy operation (a long wait, a reset, etc.). One option in these cases is to change the timeout value for the duration of the long operation, then change the timeout value back.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.3 or higher

Prototypes
Delphi: function MCSetTimeoutEx(hCtlr: HCTRLR; timeout: Double; var pOldTimeout: Double): Longint; stdcall;
VB: Function MCSetTimeoutEx(ByVal hCtrlr As Integer, ByVal timeout As Double, oldTimeout As Double) As Long
LabVIEW: Not Supported

MCCL Reference
None

Chapter Contents

Chapter

11

OEM Low Level Functions

The OEM low level commands provide direct access to controller functionality. The functions in this group are not part of the formal Motion Control API.

These functions have been implemented in a way that is consistent with DOS mode libraries for these controllers. This consistency is designed to simplify the task of porting existing DOS applications to Windows.

To see examples of how the functions in this chapter are used, please refer to the online Motion Control API Reference.

pmccmd

pmccmd( ) downloads a formatted binary command buffer directly to the PMC controller. Programmers should use the more advanced **pmccmdex( )** instead of this function when possible.

```
long int pmccmd(
   HCTRLR hCtlr,                // controller handle
   short int bytes,             // length of buffer
   void* pBuffer                // pointer to command buffer
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
bytes	Length of buffer, in bytes.
pBuffer	Pointer to command buffer.

Returns

The return value from this function is the actual number of bytes downloaded. Because of the nature of the binary interface, the return value will be equal to the buffer size (value of the *bytes* argument),

indicating the command buffer was successfully downloaded, or zero, indicating a problem communicating with the controller.

Comments

The binary interface is described in detail in the hardware manual that accompanied your controller. The user of this function is responsible for correctly formatting the buffer - no checking is performed by the function. To send binary commands to the motion controller the *hCtlr* handle must have opened in binary mode.

This function may be used within an **MCBlockBegin( )** / **MCBlockEnd( )** pair to create Macros, Compound commands, or Tasks.

This command function may also be used in ASCII mode; in this case the command buffer should contain a correctly formatted ASCII command (including the terminating carriage return "\r").

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcapi.h and mccl.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: function pmccmd(hCtlr: HCTRLR; bytes: SmallInt; pBuffer: PChar): SmallInt; stdcall;
VB: Function pmccmd(ByVal hCtrlr As Integer, ByVal bytes As Integer, ByVal buffer As String) As Integer
LabVIEW: Not Supported

MCCL Reference

None

See Also

pmcrdy( ), **pmcrpy( )**

pmccmdex

pmccmdex( ) downloads a formatted binary command buffer directly to the PMC controller.

```
long int pmccmdex(
   HCTRLR hCtlr,                  // controller handle
   WORD axis,                     // Axis number for this command
   WORD cmd,                      // MCCL command
   void* pArgument,               // pointer to command argument
   long int type                  // type of argument
);
```

Parameters

hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
axis Axis number for this command.
cmd MCCL command to execute - see MCCL.H and the User's Manual for your motion controller.
pArgument Pointer to a variable that has the argument for this command.
type Type of data pointed to by *pArgument*:

Value	Description
MC_TYPE_LONG	Indicates *pArgument* points to a variable of type long integer.
MC_TYPE_DOUBLE	Indicates *pArgument* points to a variable of type double precision floating point.
MC_TYPE_FLOAT	Indicates *pArgument* points to a variable of type single precision floating point.
MC_TYPE_REG	Indicates *pArgument* points to a variable of the format of a 32 bit integer with register number.
MC_TYPE_NONE	Indicates *pArgument* points to a variable of type which is NULL.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments

The binary interface is described in detail in the hardware manual that accompanied your controller. To send binary commands to the motion controller the *hCtlr* handle must have opened in binary mode.

This function may be used within an **MCBlockBegin( )** / **MCBlockEnd( )** pair to create Macros, Compound commands, or Tasks.

Compatibility

There are no compatibility issues with this function.

Requirements
Header: include mcapi.h and mccl.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.2 or higher

Prototypes
Delphi: function pmccmdex(hCtlr: HCTRLR; axis: Word; cmd: Word; var pArgument: Pointer; type: Longint): Longint; stdcall;
VB: Function pmccmdex(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal cmd As Integer, argument As Any, ByVal argtype As Long) As Long
LabVIEW: Not Supported

MCCL Reference
None

See Also
pmcrdy( ), **pmcrpyex( )**

pmcgetc

pmcgetc( ) reads a single character from the controller ASCII interface.

```
short int pmcgetc(
   HCTRLR hCtlr                  // controller handle
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.

Returns
The return value from this function is number of bytes actually read from the controller (1 or 0).

Comments
This function will return immediately if there is no character available. Use the string get command, **pmcgets( )**, if you want to wait for a character, or place **pmcgetc( )** in a loop.

You must open the controller in ASCII mode (MC_OPEN_ASCII) in order to use this command.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes

Delphi: function pmcgetc(hCtlr: HCTRLR): SmallInt; stdcall;
VB: Function pmcgetc(ByVal hCtrlr As Integer) As Integer
LabVIEW: Not Supported

MCCL Reference

None

See Also

pmcgetc( ), **pmcputc( )**, **pmcputs( )**

---

pmcgetramex

pmcgetramex( ) reads *bytes* from controller memory beginning at location *offset*.

```
short int pmcgetram(
   HCTRLR hCtlr,                  // controller handle
   WORD offset,                   // memory offset to read from
   void* pBuffer,                 // buffer to hold ram value
   DWORD size                     // number of bytes of memory to read
);
```

Parameters

hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
offset	Starting memory location, relative to the beginning of controller dual ported ram, to read from.
pBuffer	Buffer to hold read in controller memory, must be at least *bytes* long.
size	Number of bytes of memory to read.

Returns

The return value will be MCERR_NOERROR if there were no errors, or one of the MCERR_xxxx defined error codes if there was a problem.

Comments

No range checking is performed on Offset or Bytes - it is the caller's responsibility to supply valid values for these arguments. Consult the controller hardware manual for details on the controller memory map. The extended version of this function supports 32-bit offsets and buffer sizes to better support PMC's newest motion controllers.

These functions use the mccl read commands to access data from the controllers viewpoint. The original version of pmcgetram, **pmcgetram( )**, applied an internal offset to the caller's offset parameter to make addresses seem more natural (e.g. 1000 hex was added to adresses on ISA-bus controllers so that the addresses matched the dual port ram as seen from the PC). **pmcgetramex( )** does not apply any offset.

Do not use this command within an **MCBlockBegin( )** / **MCBlockEnd( )** block.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 3.4 or higher

Prototypes
Delphi: procedure pmcgetramex(hCtlr: HCTRLR; offset: Word; pBuffer: PChar; bytes: SmallInt); stdcall;
VB: Sub pmcgetramex(ByVal hCtrlr As Integer, ByVal offset As Integer, ByVal buffer As String, ByVal bytes As Integer)
LabVIEW: Not Supported

MCCL Reference
None

See Also
pmcputramex( )

---

pmcgets

pmcgets( ) reads a null-terminated ASCII string of up to *bytes* characters from the controller ASCII interface.

```
short int pmcgets(
   HCTRLR hCtlr,                  // controller handle
   void* pBuffer,                 // pointer to buffer
   short int bytes                // length of buffer
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
pBuffer	Pointer to reply buffer.
bytes	Length of *buffer*, in bytes.

Returns
The return value from this function is number of bytes actually read from the controller.

Comments
This function will wait for a reply for as long as the controller is busy processing command. A zero will be returned when the controller is idle and there are no reply characters. However, a non-zero timeout value will force the function to return the number of characters it has received prior to the timeout.

You must open the controller in ASCII mode (MC_OPEN_ASCII) in order to use this command.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: function pmcgets(hCtlr: HCTRLR; pBuffer: PChar; bytes: SmallInt): SmallInt; stdcall;
VB: Function pmcgets(ByVal hCtrlr As Integer, ByVal buffer As String, ByVal bytes As Integer) As Integer
LabVIEW: Not Supported

MCCL Reference
None

See Also
MCSetTimeoutEx( ), **pmcgetc( )**, **pmcputc( )**, **pmcputs( )**

---

pmcputc

pmcputc( ) writes a single character to the controller ASCII interface.

```
short int pmcputc(
   HCTRLR hCtlr,                  // controller handle
   short int char                 // output char
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
char Character to output.

Returns
This function returns a one if the character is successfully written or a zero if it is unable to write to the controller.

Comments
Remember to terminate all command strings with a carriage return "\r" in order for the command to be executed. This command does not wait for the controller - if it is unable to write the character it returns immediately with a return value of zero.

You must open the controller in ASCII mode (MC_OPEN_ASCII) in order to use this command.

Do not use this command within an **MCBlockBegin( )** / **MCBlockEnd( )** block. This function attempts to write immediately to the motion controller.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: function pmcputc(hCtlr: HCTRLR; char: SmallInt): SmallInt; stdcall;
VB: Function pmcputc(ByVal hCtrlr As Integer, ByVal char As Integer) As Integer
LabVIEW: Not Supported

MCCL Reference
None

See Also
pmcgetc( ), **pmcgets( )**, **pmcputs( )**

---

pmcputramex

pmcputramex( ) writes *bytes* directly into the controller's memory beginning at location *offset*.

```
void pmcputram(
   HCTRLR hCtlr,                  // controller handle
   WORD offset,                   // memory offset to write to
   void* pBuffer,                 // buffer to hold ram value
   DWORD size                     // number of bytes of memory to write
);
```

Parameters
hCtlr	Controller handle, returned by a successful call to **MCOpen( )**.
offset	Starting memory location, relative to the beginning of controller dual ported ram, to write to.
pBuffer	Buffer of data to write into controller memory.
size	Number of bytes of memory to write.

Returns
The return value will be MCERR_NOERROR if there were no errors, or one of the MCERR_xxxx defined error codes if there was a problem.

Comments

No range checking is performed on *offset* or *bytes*. It is the caller's responsibility to supply valid values for these arguments. Writing directly to dual ported ram can cause unpredictable results. **USE THIS FUNCTION WITH EXTREME CAUTION!**

This function uses the mccl write commands to access data from the controllers viewpoint. The original version of pmcputram, **pmcputram( )**, applied an internal offset to the caller's offset parameter to make addresses seem more natural (e.g. 1000 hex was added to adresses on ISA-bus controllers so that the addresses matched the dual port ram as seen from the PC). **pmcputramex( )** does not apply any offset.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 3.4 or higher

Prototypes
Delphi: procedure pmcputramex(hCtlr: HCTRLR; offset: Word; pBuffer: PChar; bytes: SmallInt); stdcall;
VB: Sub pmcputramex(ByVal hCtrlr As Integer, ByVal offset As Integer, ByVal buffer As String, ByVal bytes As Integer)
LabVIEW: Not Supported

MCCL Reference
None

See Also
pmcgetramex( )

---

pmcputs

pmcputs( ) writes a NULL terminated command string to the controller ASCII interface.

```
short int pmcputs(
   HCTRLR hCtlr,                // controller handle
   char* pBuffer                // output string
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
pBuffer Output string.

Returns
This function returns the number of characters actually written to the controller. This number may be less than the length of the string if the controller becomes busy and stops accepting characters.

Comments
Remember to terminate all command strings with a carriage return "\r" in order for the command to be executed. This function consumes any reply characters from the controller while it is writing (this may change in future implementations).

You must open the controller in ASCII mode (MC_OPEN_ASCII) in order to use this command.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: function pmcputs(hCtlr: HCTRLR; pBuffer: PChar): SmallInt; stdcall;
VB: Function pmcputs(ByVal hCtrlr As Integer, ByVal buffer As String) As Integer
LabVIEW: Not Supported

MCCL Reference
None

See Also
pmcgetc( ), **pmcgets( )**, **pmcputs( )**

---

pmcrdy

pmcrdy( ) checks the specified controller to see if it is ready to accept a binary command buffer.

```
short int pmcrdy(
   HCTRLR hCtlr                  // controller handle
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.

Returns
The return value from this function is TRUE (+1) if the controller is ready to accept commands. The controller will return FALSE if it is busy. For the AT200 controller, a value of -1 is returned if the controller is ready to accept data in file download mode.

Comments
Basic language users are cautioned that Visual Basic defines TRUE as -1, while Windows defines TRUE to be +1 (the API uses the Windows value for TRUE and returns a +1 if the controller is ready). Therefore, code such as:

```
   if pmcrdy( hCtlr ) = True then
```

will not work as expected in Visual Basic.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h, mcapi.pas, or mcapi32.bas
Library: mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: function pmcrdy(hCtlr: HCTRLR): SmallInt; stdcall;
VB: Function pmcrdy(ByVal hCtrlr As Integer) As Integer
LabVIEW: Not Supported

MCCL Reference
None

See Also
pmccmd( ), **pmcrpy( )**

---

pmcrpy

pmcrpy( ) reads a binary reply of up to *bytes* bytes from the controller. Programmers should use the more advanced **pmcrpyex( )** instead of this function when possible.

```
long int pmcrpy(
   HCTRLR hCtlr,                // controller handle
   short int bytes,             // length of buffer
   void* pBuffer                // pointer to buffer
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
bytes Length of buffer, in bytes.
pBuffer Pointer to reply buffer.

Returns
The return value from this function is the actual number of bytes read. This value may be less than the argument *bytes*, but will never exceed *bytes*. If the controller has no reply ready, the return value will be zero.

Comments
This function waits for a reply for as long as the controller is busy - it returns with a return value of zero if no reply is (or will be) available.

You must open the controller in ASCII mode (MC_OPEN_ASCII) in order to use this command.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h and mccl.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 1.0 or higher

Prototypes
Delphi: function pmcrpy(hCtlr: HCTRLR; bytes: SmallInt; pBuffer: PChar): SmallInt; stdcall;
VB: Function pmcrpy(ByVal hCtrlr As Integer, ByVal bytes As Integer, ByVal buffer As String) As Integer
LabVIEW: Not Supported

MCCL Reference
None

See Also
pmccmd( ), **pmcrdy( )**, **pmcrpyex( )**

---

pmcrpyex

pmcrpyex( ) reads a binary reply of up to *bytes* bytes from the controller.

```
long int pmcrpyex(
   HCTRLR hCtlr,                  // controller handle
   void* pReply,                  // pointer to command reply
   long int type                  // type of argument
);
```

Parameters
hCtlr Controller handle, returned by a successful call to **MCOpen( )**.
pReply Pointer to a variable to hold the reply value.
type Type of data pointed to by *pReply*:

Value	Description
MC_TYPE_LONG	Indicates *pReply* points to a variable of type long integer.

MC_TYPE_DOUBLE	Indicates *pReply* points to a variable of type double precision floating point.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was a problem.

Comments
The binary interface is described in detail in the hardware manual that accompanied your controller.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcapi.h and mccl.h, mcapi.pas, or mcapi32.bas
Library: use mcapi32.lib
Version: MCAPI 2.2 or higher

Prototypes
Delphi: function pmcrpyex(hCtlr: HCTRLR; var pReply: Pointer; type: Longint): Longint; stdcall;
VB: Function pmcrpyex(ByVal hCtrlr As Integer, reply As Any, ByVal argtype As Long) As Long
LabVIEW: Not Supported

MCCL Reference
None

See Also
pmccmdex( ), **pmcrdy( )**, **pmcrpy( )**

Chapter Contents

Chapter

12

Common Motion Dialog Functions

The Common Motion Dialog library includes easy-to-use high-level functions for the control and configuration of your motion controller. By combining these functions in a single library we've made it easy for programmers to include the Common Motion Dialog functionality in their application programs. Functions are provided for the configuration of servo and stepper axes, scaling setup, controller selection, file download, and save/restore of motor settings.

To see examples of how the functions in this chapter are used, please refer to the online Motion Control API Reference.

MCDLG_AboutBox

MCDLG_AboutBox( ) displays a simple About dialog box that includes version information about both the application and the Motion Control API.

```
long int MCDLG_AboutBox(
   HWND hWnd,                    // handle to parent window
   LPCSTR title,                 // title string for the dialog box
   long int bitmapID             // bitmap ID for the dialog box
);
```

Parameters

hWnd	Handle to parent window of About Box. This handle is used by **MCDLG_AboutBox( )** to retrieve VERSIONINFO strings from the application.
title	An optional title string for the About dialog box. If this pointer is NULL or points to a zero length string the default title of “About” is used.
bitmapID	An optional Bitmap resource identifier. If greater than zero, the specified bitmap will be displayed in the About dialog box. If zero, **MCDLG_AboutBox( )** will display the default bitmap. Bitmaps should be no larger than 240 (width) by 80 (height) pixels, 16 colors.

Returns
This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was an error creating the dialog box.

Comments
Version information is obtained by retrieving VERSIONINFO values from the executable module. The specific strings queried for are “CompanyName”, “FileDescription”, “FileVersion”, and “LegalCopyright”. It is a good idea to include a VERSIONINFO resource in any application as it permits Windows to accurately determine the version of any executable file or DLL. Applications and DLLs supplied with the Motion Control API include a VERSIONINFO resource.

The dialog box displays a default logo bitmap above the version information. By specifying a valid bitmap resource ID for the *bitmapID* parameter you may change the bitmap displayed. If this parameter is greater than zero the new bitmap will replace the default in the About dialog box. Bitmaps should be no larger than 240 (width) by 80 (height) pixels, 16 colors.

If a NULL pointer or a pointer to a zero length string is passed as the *title* argument the default title will be used. Acceptance of a pointer to a zero length string was included to support programming languages that have difficulty with NULL pointers (e.g. Visual Basic). To eliminate the title pass a pointer to a string with a single space (i.e. " ").

Note that **MCDLG_AboutBox( )** uses the HWND argument passed to it to identify the executable file from which to read the VERSIONINFO information. In some development environments, such as Visual Basic, window handles are owned by a DLL supplied by the author of the development system, not the user's EXE file. In these situations, **MCDLG_AboutBox( )** is unable to correctly perform its VERSIONINFO query and should not be used.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes
Delphi: function MCDLG_AboutBox(hWnd: HWnd; title: PChar; bitmapID: Longint): Longint; stdcall;
VB: Function MCDLG_AboutBox(ByVal hWnd As Long, ByVal title As String, ByVal bitmapID As Long) As Long
LabVIEW: Not Supported

MCDLG_CommandFileExt

MCDLG_CommandFileExt( ) returns the file extension for MCCL command files for a particular motion controller type.

```
long int MCDLG_CommandFileExt(
   long int type,                // controller type identifier
   long int flags,               // flags
   LPCSTR buffer,                // buffer for file extension string
   long int length               // length of string buffer, in bytes
);
```

Parameters

type	Motion Controller type, must be equal to one of the predefined motion controller types (see MCAPI.H).
flags	Reserved for future use (set to zero).
buffer	Pointer to a string buffer that will hold the file extension (should be _MAX_FILE long).
length	Size of buffer, in bytes.

Returns

This function returns a pointer to the file extension string for the specified motion controller type. It returns NULL if *type* does not specify a valid controller type.

Comments

The Motion Control API registers a separate file extension for each controller type. The MCAPI tools, such as Win Control, use these file extensions when they open MCCL command files. You can use this function to get the registered file extension for any controller type.

See the MCAPI sample program Win Control for an example.

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 3.0 or higher

Prototypes

Delphi:	function MCDLG_CommandFileExt(type: LongInt; flags: LongInt; buffer: PChar; length: Longint): PChar; stdcall;
VB:	Function MCDLG_CommandFileExt(ByVal argtype As Long, ByVal flags As Long, ByVal buffer As String, ByVal length As Long) As String
LabVIEW:	Not Supported

MCDLG_ConfigureAxis

MCDLG_ConfigureAxis( ) displays a servo or stepper axis setup dialog that permits user configuration of the axis.

```
long int MCDLG_ConfigureAxis(
   HWND hWnd,                    // handle to parent window
   HCTRLR hCtlr,                 // handle to a motion controller
   WORD axis,                    // axis number to configure
   long int flags,               // configuration flags
   LPCSTR title                  // optional axis title for the dialog box
);
```

Parameters

hWnd Handle to parent window. May be NULL.
hCtlr Motion Controller handle, returned by a successful call to MCOpen( ).
axis Axis number of axis to be configured.
flags Flags to control the operation (multiple flags may be OR'ed together):

Value	Description
MCDLG_CHECKACTIVE	Checks if an axis is moving before the new settings are written to the controller and skips if the axis is moving. Combine with MCDLG_PROMPT to prompt user whether or not to proceed.
MCDLG_PROMPT	Combine with MCDLG_CHECKACTIVE to prompt user whether or not to proceed if a motor is moving and the user has dismissed the dialog box with OK.

title An optional title string for the axis. If this pointer is NULL or points to a zero length string the default title, which includes the axis number and a description of the axis type is used.

Returns

This function returns MCERR_NOERROR if the user pressed OK button to dismiss the dialog box. It returns MCERR_CANCEL if the user pressed the CANCEL button to dismiss the dialog box. It returns one of the other MCERR_xxxx error codes if there was an error creating the dialog box.

Comments

This function provides comprehensive, ready-to-use setup dialogs for stepper and servo motor axis types. The motion controller is queried for the current axis settings to initialize this dialog box. Any changes the user makes are sent to the motion controller if the user dismisses the dialog by pressing the OK button.

Changing the parameters of an axis while it is moving may result in erratic behavior (such as when you choose to include the motor position in the changed parameters). The flag MCDLG_CHECKACTIVE forces this function to check the axis to see if it is active before it proceeds. By default MCDLG_CHECKACTIVE will skip the changing of an active axis, but if you also include the

flag MCDLG_PROMPT the user will be prompted for how to proceed. The programming samples are all built with MCDLG_CHECKACTIVE and MCDLG_PROMPT set.

If a NULL pointer or a pointer to a zero length string is passed as the *title* argument, the default title will be used. Acceptance of a pointer to a zero length string was included to support programming languages that have difficulty with NULL pointers (e.g. Visual Basic). To eliminate the title pass a pointer to a string with a single space (i.e. " ").

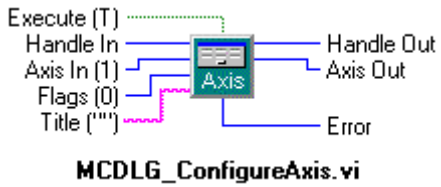
Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes
Delphi: function MCDLG_ConfigureAxis(hWnd: HWnd; hCtlr: HCTRLR; axis: Word; flags: Longint; title: PChar): Longint; stdcall;

VB: Function MCDLG_ConfigureAxis(ByVal hWnd As Long, ByVal hCtlr As Integer, ByVal axis As Integer, ByVal flags As Long, ByVal title As String) As Long

LabVIEW:



MCDLG_ControllerDescEx

MCDLG_ControllerDescEx( ) returns a descriptive string for the specified motion controller type.

```
LPCSTR MCDLG_ControllerDescEx(
   long int type,                // controller type identifier
   long int flags,               // flags
   LPSTR buffer,                 // buffer for descriptive string
   long int length               // size of buffer, in bytes
);
```

Parameters
type Motion Controller type, must be equal to one of the predefined motion controller types (see MCAPI.H).

flags Flags to control the operation:

Value	Description
MCDLG_NAMEONLY	Resulting string will contain only the name portion (no description).

MCDLG_DESCONLY	Resulting string will contain only the name portion (no name).

buffer Pointer to a string buffer that will hold the descriptive string.
length Size of *buffer*, in bytes.

Returns
This function returns a pointer to the descriptive string buffer for the specified motion controller type, or it returns NULL if *type* does not specify a valid controller type.

Comments
This extended version of **MCDLG_ControllerDesc( )** includes by default the controller name and a description of the controller in the output string. Use the *flags* parameter to control the information included in the string.

You may use this function to provide a descriptive string for a motion controller by passing the function the **ControllerType** member of an **MCPARAMEX** structure following a call to **MCGetConfigurationEx( )**. As an example, the MCDLG function **MCDLG_ControllerInfo( )** uses this function to produce its Controller Information dialog.

Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 3.0 or higher

Prototypes
Delphi: function MCDLG_ControllerDescEx(type: LongInt; flags: LongInt; buffer: PChar; length: Longint): PChar; stdcall;
VB: Function MCDLG_ControllerDescEx(ByVal argtype As Long, ByVal flags As Long, ByVal buffer As String, ByVal length As Long) As String
LabVIEW: Not Supported

MCDLG_ControllerInfo

MCDLG_ControllerInfo( ) displays configuration information about the specified motion controller.

```
long int MCDLG_ControllerInfo(
   HWND hWnd,                    // handle to parent window
   HCTRLR hCtlr,                 // handle to a motion controller
   long int flags,               // configuration flags
   LPCSTR title                  // title for the dialog box
);
```

Parameters
hWnd Handle to parent window. May be NULL.

hCtlr	Motion Controller handle, returned by a successful call to **MCOpen( )**.
flags	Currently no flags are defined for **MCDLG_ControllerInfo( )**, and this argument should be set to zero.
title	An optional title string for the dialog box. If this pointer is NULL or points to a zero length string, a default title is used.

Returns

This function returns MCERR_NOERROR if there were no errors, or it returns one of the MCERR_xxxx defined error codes if there was an error creating the dialog box.

Comments

This function displays a read only dialog providing information on the current motion controller configuration and capabilities (this information is typically used by programs to control execution for example can the controller multi-task? Is contouring supported?).

If a NULL pointer or a pointer to a zero length string is passed as the *title* argument the default title will be used. Acceptance of a pointer to a zero length string was included to support programming languages that have difficulty with NULL pointers (e.g. Visual Basic). To eliminate the title pass a pointer to a string with a single space (i.e. " ").

Compatibility

There are no compatibility issues with this function.

Requirements

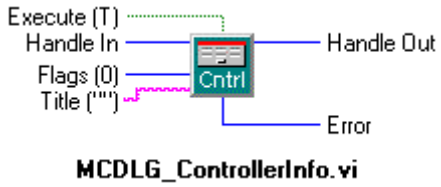
Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes

Delphi: function MCDLG_ControllerInfo(hWnd: HWnd; hCtlr: HCTRLR; flags: Longint; title: PChar): Longint; stdcall;

VB: Function MCDLG_ControllerInfo(ByVal hWnd As Long, ByVal hCtlr As Integer, ByVal flags As Long, ByVal title As String) As Long

LabVIEW:



MCDLG_DownloadFile

MCDLG_DownloadFile( ) downloads an ASCII command file to the specified motion controller.

```
long int MCDLG_DownloadFile(
   HWND hWnd,                    // handle of window to echo download to
   HCTRLR hCtlr,                 // handle of motion controller
   long int flags,               // configuration flags
   LPCSTR fileName               // path/filename of file to download
);
```

Parameters

hWnd	Handle of window to echo downloaded characters to. May be NULL.
hCtlr	Motion Controller handle, returned by a successful call to **MCOpen( )**.
flags	Currently no flags are defined for **MCDLG_ConfigureAxis( )**, and this field should be left blank.
fileName	Path / filename of file to download.

Returns

This function returns MCERR_NOERROR if the file was successfully downloaded, or it returns one of the other MCERR_xxxx error codes if there was an error downloading the file.

Comments

MCDLG_DownloadFile( ) opens the specified file and downloads the contents to the specified controller. If a valid (non-NULL) window handle is given for *hWnd*, downloaded characters (and replies from the controller) are sent to the window via WM_CHAR messages. This feature allows you to use **MCDLG_DownloadFile( )** with a terminal interface application, such as Win Control, that displays the file while it is being downloaded.

Compatibility

There are no compatibility issues with this function.

Requirements

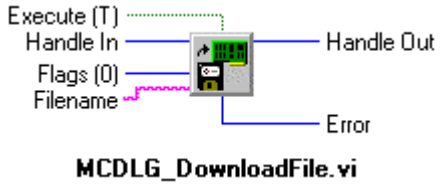
Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes

Delphi: function MCDLG_DownloadFile(hWnd: HWnd; hCtlr: HCTRLR; flags: Longint; fileName: PChar): Longint; stdcall;

VB: Function MCDLG_DownloadFile(ByVal hWnd As Long, ByVal hCtlr As Integer, ByVal flags As Long, ByVal fileName As String) As Long

LabVIEW:



MCDLG_Initialize

MCDLG_Initialize( ) must be called before any other MCDLG functions are called or any of the MCDLG window classes are used.

```
long int MCDLG_Initialize(
   void
);
```

Returns

This function returns MCERR_NOERROR if the MCDLG library was successfully initialized, or it returns one of the other MCERR_xxxx error codes if there was an error initializing the library.

Comments

Calling **MCDLG_Initialize( )** ensures that internal MCDLG data structures are correctly initialized and that MCDLG window classes are registered.

Compatibility

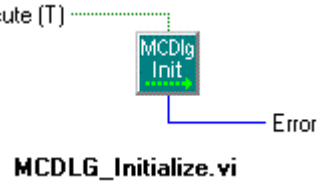
There are no compatibility issues with this function.

Requirements

Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes

Delphi: function MCDLG_Initialize: Longint; stdcall;
VB: Function MCDLG_Initialize( ) As Long
LabVIEW:



MCDLG_ListControllers

MCDLG_ListControllers( ) enumerates the types of motion controllers installed.

```
long int MCDLG_ListControllers(
   short int idArray[ ],         // pointer to an array for controller type
                                 // IDs
   short int size                // size of idArray[]
);
```

Parameters

idArray	Pointer to an array of short integers, filled with controller types on return.
size	Size of *idArray[ ]* (number of integers).

Returns

The return value is the number of installed controllers found.

Comments

MCDLG_ListControllers( ) fills *idArray[ ]* with controller type identifiers, where the type of the controller configured at ID 0 is stored in idArray[0], the type of the controller configured at ID 1 is stored in idArray[1], etc. In order to list all installed controllers the array must have a size of at least MC_MAX_ID + 1 (the constant MC_MAX_ID is defined in the MCAPI header files).

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes

Delphi:	function MCDLG_ListControllers(idArray: Array of SmallInt; size: SmallInt): Longint; stdcall;
VB:	Function MCDLG_ListControllers Lib "mcdlg32.dll" (idArray As Any, ByVal size As Integer) As Long
LabVIEW:	Not Supported

MCDLG_ModuleDescEx

MCDLG_ModuleDescEx( ) returns a descriptive string for the specified module/axis type.

```
LPCSTR MCDLG_ModuleDescEx(
   long int type,                // axis type identifier
   long int flags,               // flags
   LPSTR buffer,                 // buffer for descriptive string
   long int length               // size of buffer, in bytes
);
```

Parameters

type Module type, must be equal to one of the predefined module types (see MCAPI.H).

flags Flags to control the operation:

Value	Description
MCDLG_NAMEONLY	Resulting string will contain only the name portion (no description).
MCDLG_DESCONLY	Resulting string will contain only the description portion (no name).

buffer Pointer to a string buffer that will hold the descriptive string.

length Size of *buffer*, in bytes.

Returns

This function returns pointer to the descriptive string buffer for the specified axis type, or it returns NULL if *type* does not specify a valid axis type.

Comments

This extended version of **MCDLG_ModuleDesc( )** includes by default the module name and a description of the module in the output string. Use the *flags* parameter to control the information included in the string.

You may use this function to provide a descriptive string for an axis by passing the function the **ModuleType** member of an **MCAXISCONFIG** structure following a call to **MCGetAxisConfiguration( )**. As an example, the MCDLG function **MCDLG_ConfigureAxis( )** uses this function to produce its default axis description string.

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 3.0 or higher

Prototypes

Delphi: function MCDLG_ModuleDescEx(type: LongInt; flags: LongInt; buffer: PChar; length: Longint): PChar; stdcall;

VB: Function MCDLG_ModuleDescEx(ByVal argtype As Long, ByVal flags As Long, ByVal buffer As String, ByVal length As Long) As String

LabVIEW: Not Supported

MCDLG_RestoreAxis

MCDLG_RestoreAxis( ) restores the settings of the given axis to a previously saved state.

```
long int MCDLG_RestoreAxis(
   HCTRLR hCtlr,                // handle to a motion controller
   WORD axis,                   // axis number to configure
   long int flags,              // configuration flags
   LPCSTR privateIniFile        // optional INI file to read from
);
```

Parameters

hCtlr Motion Controller handle, returned by a successful call to **MCOpen( )**.

axis Axis number of axis to be restored.

flags Flags to control the restore operation (multiple flags may be OR'ed together):

Value	Description
MCDLG_CHECKACTIVE	Checks if an axis is moving before the settings are restored and skips if the axis is moving. Combine with MCDLG_PROMPT to prompt user whether or not to proceed.
MCDLG_NOMOTION	Do not restore **MCMOTIONEX** structure settings.
MCDLG_NOFILTER	Do not restore **MCFILTEREX** structure settings.
MCDLG_NOPHASE	Do not restore phase setting.
MCDLG_NOPOSITION	Do not restore axis position.
MCDLG_PROMPT	If the stored data doesn't match the type of the axis being restored to a Message Box will be displayed. Also affects the behavior of MCDLG_CHECKACTIVE (see above).

privateIniFile Name, optionally with path and drive, of the INI file in which to save the axis settings. If NULL **MCDLG_RestoreAxis( )** will use MCAPI.INI.

Returns

This function returns MCERR_NOERROR if there were no problems, or it returns one of the other MCERR_xxxx error codes if there was an error. The most common reason for a return value of FALSE is supplying an invalid or non-existent filename for *privateIniFile*.

Comments

MCDLG_SaveAxis( ) encodes the motion controller type and module type into signature that is saved with the axis settings. **MCDLG_RestoreAxis( )** checks for a valid signature before restoring the axis settings. If you make changes to your hardware configuration (i.e. change module types or controller type) **MCDLG_RestoreAxis( )** will refuse to restore those settings.

You may specify the constant MC_ALL_AXES for the *axis* parameter in order to restore the parameters for all axes installed on a motion controller with a single call to this function.

Restoring the parameters to an axis while it is moving may result in erratic behavior (such as when you choose to include the motor position in the restored parameters). The flag MCDLG_CHECKACTIVE forces this function to check each restored axis to see if it is active before it proceeds. By default MCDLG_CHECKACTIVE will skip the restore of an active axis, but if you also include the flag MCDLG_PROMPT the user will be prompted for how to proceed. The programming samples are all built with MCDLG_CHECKACTIVE and MCDLG_PROMPT set.

Note that this function writes a lot of information to the motion controller for each axis saved, and should be used sparingly over slow interfaces such as the RS232.

If a NULL pointer or a pointer to a zero length string is passed as the *privateIniFile* argument the default file (MCAPI.INI) will be used. Most applications should use the default file so that configuration data may be easily shared among applications. Acceptance of a pointer to a zero length string was included to support programming languages that have difficulty with NULL pointers (e.g. Visual Basic).

Compatibility

There are no compatibility issues with this function.

Requirements

Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes

Delphi: function MCDLG_RestoreAxis(hCtlr: HCTRLR; axis: Word; flags: Longint; privateIniFile: PChar): Longint; stdcall;

VB: Function MCDLG_RestoreAxis(ByVal hCtlr As Integer, ByVal axis As Integer, ByVal flags As Long, ByVal privateIniFile As String) As Long

LabVIEW:

See Also

MCDLG_SaveAxis( )

MCDLG_RestoreDigitalIO

MCDLG_RestoreDigitalIO( ) restores the settings of the all the digital I/O channels between *startChannel* and *endChannel* (inclusive) to their previously saved states.

```
long int MCDLG_RestoreDigitalIO(
   HCTRLR hCtlr,                // handle to a motion controller
   WORD startChannel,           // starting channel number to restore
   WORD endChannel,             // ending channel number to restore
   LPCSTR privateIniFile        // optional INI file to read from
);
```

Parameters

hCtlr	Motion Controller handle, returned by a successful call to **MCOpen( )**.
startChannel	Number of the first digital I/O channel axis to be restored. If set to zero the first available channel on the controller will be used.
endChannel	Number of the last digital I/O channel axis to be restored. If set to zero the last available channel on the controller will be used.
privateIniFile	Name, optionally with path and drive, of the INI file in which to save the axis settings. If NULL **MCDLG_RestoreDigitalIO( )** will use MCAPI.INI.

Returns

This function returns MCERR_NOERROR if the settings were restored correctly, or it returns MCERR_RANGE if either StartChannel or EndChannel is out of range.

Comments

By setting *startChannel* and *endChannel* both to zero this function will automatically restore all the digital I/O channels on a motion controller.

If a NULL pointer or a pointer to a zero length string is passed as the *privateIniFile* argument, the default file (MCAPI.INI) will be used. Most applications should use the default file so that configuration data may be easily shared among applications. Acceptance of a pointer to a zero length string was included to support programming languages that have difficulty with NULL pointers (e.g. Visual Basic).

Under the MCAPI, the DC2-STN controller's input channels are numbered 1 - 8, and the output channels are numbered 9 - 16 (the MCAPI requires that each channel have a unique channel number).

Compatibility

There are no compatibility issues with this function.

Requirements

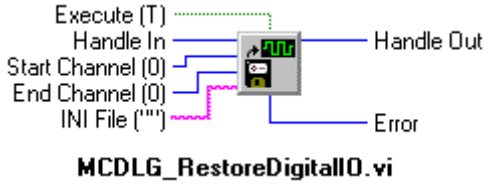
Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes

Delphi: function MCDLG_RestoreDigitalIO(hCtlr: HCTRLR; startChannel: Word; endChannel: Word; privateIniFile: PChar):Longint; stdcall;

VB: Function MCDLG_RestoreDigitalIO(ByVal hCtlr As Integer, ByVal startChannel As Integer, ByVal endChannel As Integer, ByVal privateIniFile As String) As Long

LabVIEW:



See Also

MCDLG_SaveDigitalIO( )

MCDLG_SaveAxis

MCDLG_SaveAxis( ) saves the settings of the given axis to an initialization file for later use.

```
long int MCDLG_SaveAxis(
   HCTRLR hCtlr,                  // handle to a motion controller
   WORD axis,                     // axis number to configure
   long int flags,                // configuration flags
   LPCSTR privateIniFile          // optional INI file to write to
);
```

Parameters

hCtlr Motion Controller handle, returned by a successful call to **MCOpen( )**.

axis Axis number of axis to be restored.

flags Flags to control the restore operation (multiple flags may be OR'ed together):

Value	Description
MCDLG_NOMOTION	Do not restore **MCMOTIONEX** structure settings.
MCDLG_NOFILTER	Do not restore **MCFILTEREX** structure settings.
MCDLG_NOPHASE	Do not restore phase setting.
MCDLG_NOPOSITION	Do not restore axis position.

privateIniFile Name, optionally with path and drive, of the INI file in which to save the axis settings. If NULL **MCDLG_RestoreAxis( )** will use MCAPI.INI.

Returns

This function returns MCERR_NOERROR if there were no problems, or it returns one of the other MCERR_xxxx error codes if there was an error. The most common reason for a return value of FALSE is supplying an invalid or non-existent filename for *privateIniFile*.

Comments

MCDLG_SaveAxis( ) encodes the motion controller type and module type into signature that is saved with the axis settings. **MCDLG_RestoreAxis( )** checks for a valid signature before restoring the axis settings. If you make changes to your hardware configuration (i.e. change module types or controller type) **MCDLG_RestoreAxis( )** will refuse to restore those settings.

You may specify the constant MC_ALL_AXES for the *axis* parameter in order to save the parameters for all axes installed on a motion controller with a single call to this function. Setting *axis* to -1 will cause **MCDLG_SaveAxis( )** to delete all of the stored axis information for this controller.

Note that this function reads a lot of information from the motion controller for each axis saved, and should be used sparingly over slow interfaces such as the RS232.

If a NULL pointer or a pointer to a zero length string is passed as the *privateIniFile* argument the default file (MCAPI.INI) will be used. Most applications should use the default file so that configuration data may be easily shared among applications. Acceptance of a pointer to a zero length string was included to support programming languages that have difficulty with NULL pointers (e.g. Visual Basic).

Compatibility

There are no compatibility issues with this function.

Requirements

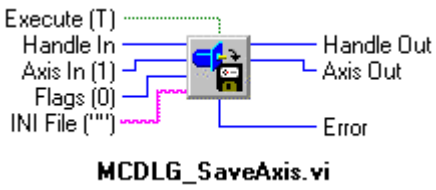
Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes

Delphi: function MCDLG_SaveAxis(hCtlr: HCTRLR; axis: Word; flags: Longint; privateIniFile: PChar): Longint; stdcall;

VB: Function MCDLG_SaveAxis(ByVal hCtlr As Integer, ByVal axis As Integer, ByVal flags As Long, ByVal privateIniFile As String) As Long

LabVIEW:



MCDLG_SaveDigitalIO

MCDLG_SaveDigitalIO( ) saves the settings of the all the digital I/O channels between *startChannel* and *endChannel* (inclusive) to an INI file.

```
long int MCDLG_SaveDigitalIO(
   HCTRLR hCtlr,                 // handle to a motion controller
   WORD startChannel,            // starting channel number to save
   WORD endChannel,              // ending channel number to save
   LPCSTR privateIniFile         // optional INI file to write to
);
```

Parameters

hCtlr	Motion Controller handle, returned by a successful call to **MCOpen( )**.
startChannel	Number of the first digital I/O channel axis to be restored. If set to zero the first available channel on the controller will be used.
endChannel	Number of the last digital I/O channel axis to be restored. If set to zero, the last available channel on the controller will be used.
privateIniFile	Name, optionally with path and drive, of the INI file in which to save the axis settings. If NULL **MCDLG_SaveDigitalIO( )** will use MCAPI.INI.

Returns

MCERR_NOERROR if the settings were saved correctly or MCERR_RANGE if either *startChannel* or *endChannel* is out of range.

Comments

By setting *startChannel* and *endChannel* both to zero this function will automatically save all the digital I/O channels on a motion controller.

If a NULL pointer or a pointer to a zero length string is passed as the *privateIniFile* argument the default file (MCAPI.INI) will be used. Most applications should use the default file so that configuration data may be easily shared among applications. Acceptance of a pointer to a zero length string was included to support programming languages that have difficulty with NULL pointers (e.g. Visual Basic).

Under the MCAPI, the DC2-STN controller's input channels are numbered 1 - 8, and the output channels are numbered 9 - 16 (the MCAPI requires that each channel have a unique channel number).

Compatibility

There are no compatibility issues with this function.

Requirements

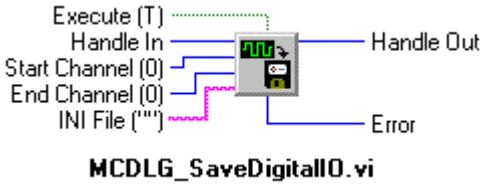
Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes

Delphi: function MCDLG_SaveDigitalIO(hCtlr: HCTRLR; startChannel: Word; endChannel: Word; privateIniFile: PChar):Longint; stdcall;

VB: Function MCDLG_SaveDigitalIO(ByVal hCtlr As Integer, ByVal startChannel As Integer, ByVal endChannel As Integer, ByVal privateIniFile As String) As Long

LabVIEW:



MCDLG_Scaling

MCDLG_Scaling( ) displays a scaling setup dialog and, if the motion controller supports scaling, allows the user to change the scaling parameters.

```
long int MCDLG_Scaling(
   HWND hWnd,                    // handle to parent window
   HCTRLR hCtlr,                 // handle to a motion controller
   WORD axis,                    // axis number to configure
   long int flags,               // configuration flags
   LPCSTR title                  // optional title for the dialog box
);
```

Parameters

hWnd Handle to parent window. May be NULL.

hCtlr Motion Controller handle, returned by a successful call to **MCOpen( )**.

axis Axis number of axis to be scaled.

flags Flags to control scaling:

Value	Description
MCDLG_PROMPT	If user clicks OK to dismiss dialog display a message warning that scaling changes will take effect following the next motor on command.

title An optional title string for the About dialog box. If this pointer is NULL or points to a zero length string the default title of “About” is used.

Returns

This function returns MCERR_NOERROR if the user pressed OK button to dismiss the dialog box. It returns MCERR_CANCEL if the user pressed the CANCEL button to dismiss the dialog box, or it returns one of the other MCERR_xxxx error codes if there was an error creating the dialog box.

Comments

For controllers that don't support scaling the Motion Control API will fill in the **MCSCALE** data structure with default values (zero for offsets, one for factors). **MCDLG_Scaling( )** will display these

defaults as read-only. For advanced controllers such as the DCX-AT and the DCX-PCI **MCDLG_Scaling( )** will display the current scale factors and allow the user to change them.

If a NULL pointer or a pointer to a zero length string is passed as the *title* argument the default title will be used. Acceptance of a pointer to a zero length string was included to support programming languages that have difficulty with NULL pointers (e.g. Visual Basic). To eliminate the title pass a pointer to a string with a single space (i.e. " ").

NOTE: Scaling changes will take effect following the next motor on command (**MCEnableAxis( )**) after **MCDLG_Scaling( )** completes.

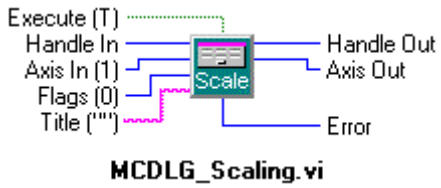
Compatibility
There are no compatibility issues with this function.

Requirements
Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes
Delphi: function MCDLG_Scaling(hWnd: HWnd; hCtlr: HCTRLR; axis: Word; flags: Longint; title: PChar): Longint; stdcall;

VB: Function MCDLG_Scaling(ByVal hWnd As Long, ByVal hCtlr As Integer, ByVal axis As Integer, ByVal flags As Long, ByVal title As String) As Long

LabVIEW:



MCDLG_SelectController

MCDLG_SelectController( ) displays a list of installed controllers and allows the user to select a controller from the list.

```
long int MCDLG_SelectController(
   HWND hWnd,                    // handle to parent window
   short int currentID,          // ID of currently selected controller
   long int flags,               // configuration flags
   LPCSTR title                  // optional title for the dialog box
);
```

Parameters
hWnd	Handle to parent window. May be NULL.
currentID	ID of the motion controller currently in use. In the selection list, this controller will be highlighted. Set to -1 to ignore.
flags	Currently no flags are defined for **MCDLG_ConfigureAxis( )**, and this field should be left blank.

title An optional title string for the dialog box. If this pointer is NULL or points to a zero length string the default title is used.

Returns

This function returns a controller ID if the user selected a controller and pressed the OK button to dismiss the dialog, or it returns a -1 if the user pressed the CANCEL button to dismiss the dialog. A value of -1 is also returned if there are no motion controllers currently configured.

Comments

This function displays a list of installed controllers and allows the user to select one from the list. If a valid ID is given for *currentID* that controller will be highlighted in the list as the default selection (set *currentID* to -1 prevent a default selection). If no motion controllers have been configured for use with the Motion Control Applet in the Motion Control Panel, a message is displayed indicating that no controllers are configured and -1 is returned to the calling program.

If a NULL pointer or a pointer to a zero length string is passed as the *title* argument the default title will be used. Acceptance of a pointer to a zero length string was included to support programming languages that have difficulty with NULL pointers (e.g. Visual Basic). To eliminate the title pass a pointer to a string with a single space (i.e. " ").

Compatibility

There are no compatibility issues with this function.

Requirements

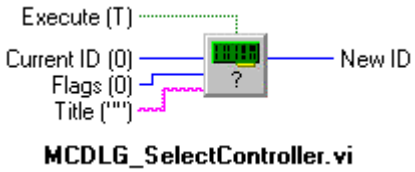
Header: include mcdlg.h, mccdlg.pas, or mcdlg32.bas
Library: use mcdlg32.lib and mcapi32.lib
Version: MCAPI 2.1 or higher

Prototypes

Delphi: function MCDLG_SelectController(hWnd: HWnd; currentID: SmallInt; flags: Longint; title: PChar): SmallInt; stdcall;

VB: Function MCDLG_SelectController(ByVal hWnd As Long, ByVal currentID As Integer, ByVal flags As Long, ByVal title As String) As Integer

LabVIEW:



Appendix

A

Appendix A - MCAPI Error Codes

The MCAPI defined error messages are listed numerically in the following table. Where possible corrective action is included in the description column. Please note that many MCAPI function descriptions also include information regarding errors that are specific to that function.

Error	Constant	Description
0	MCERR_NOERROR	No error has occurred.
1	MCERR_NO_CONTROLLER	No controller assigned at this ID. Use MCSETUP to configure a controller.
2	MCERR_OUT_OF_HANDLES	MCAPI driver out of handles. The driver is limited to 32 open handles. Applications that do not call **MCClose( )** when they exit may leave handles unavailable, forcing a reboot.
3	MCERR_OPEN_EXCLUSIVE	Cannot open - another application has the controller opened for exclusive use.
4	MCERR_MODE_UNAVAIL	Controller already open in different mode. Some controller types can only be open in one mode (ASCII or binary) at a time.
5	MCERR_UNSUPPORTED_MODE	Controller doesn't support this mode for **MCOpen( )** - i.e. ASCII or binary.
6	MCERR_INIT_DRIVER	Couldn't initialize the device driver.
7	MCERR_NOT_PRESENT	Controller hardware not present.
8	MCERR_ALLOC_MEM	Memory allocation error. This is an internal memory allocation problem with the DLL, contact Technical Support for assistance.
9	MCERR_WINDOWSERROR	A windows function returned an error - use **GetLastError( )** under WIN32 for details
10	-	reserved
11	MCERR_NOTSUPPORTED	Controller doesn't support this feature.
12	MCERR_OBSOLETE	Function is obsolete.
13	MCERR_CONTROLLER	Invalid controller handle.
14	MCERR_WINDOW	Invalid window handle.
15	MCERR_AXIS_NUMBER	Axis number out of range.
16	MCERR_AXIS_TYPE	Axis type doesn't support this feature.
17	MCERR_ALL_AXES	Cannot use MC_ALL_AXES for this function.
18	MCERR_RANGE	Parameter was out of range.
19	MCERR_CONSTANT	Constant value inappropriate.
20	MCERR_UNKNOWN_REPLY	Unexpected or unknown reply.
21	MCERR_NO_REPLY	Controller failed to reply.
22	MCERR_REPLY_SIZE	Reply size incorrect.
23	MCERR_REPLY_AXIS	Wrong axis for reply.
24	MCERR_REPLY_COMMAND	Reply is for different command.
25	MCERR_TIMEOUT	Controller failed to respond.
26	MCERR_BLOCK_MODE	Block mode error. Caused by calling **MCBlockEnd( )** without first calling **MCBlockBegin( )** to begin the block.
27	MCERR_COMM_PORT	Communications port (RS232) driver reported an error.

Error	Constant	Description
28	MCERR_CANCEL	User canceled action (such as when an MCDLG dialog box is dismissed with the CANCEL button.
29	MCERR_NOT_INITIALIZED	Feature was not correctly initialized before being enable or used.

Appendix

B

Appendix B - Constants

The symbolic constants described in this section provide a safe, descriptive way of accessing the MCAPI features. The actual numeric value of these constants may change in future versions of the API, however the constant names will remain fixed. Use of these symbolic values will help to insure that future changes to the API won't break existing code. The constant values also help to produce more readable code. To find the actual value of any given constant, please refer to the online Motion Control API Reference or the MCAPI.H header file.

Constant	Description
DC2PC100	Value for the **ControllerType** member of an **MCPARAMEX** structure, it indicates that a DC2 PC100 controller is installed.
DC2SERVO	Identifies an axis as one of the dedicated servo axes on a DC2PC100 controller.
DC2STEPPER	Identifies an axis as one of the optional stepper axes on a DC2PC100 controller.
DC2STN	Value for the **ControllerType** member of an **MCPARAMEX** structure, it indicates that a DC2 STN controller is installed.
DCXPC100	Value for the **ControllerType** member of an **MCPARAMEX** structure, it indicates that a DCX series PC100 controller is installed.
DCXAT100	Value for the **ControllerType** member of an **MCPARAMEX** structure, it indicates that a DCX series AT100 controller is installed.
DCXAT200	Value for the **ControllerType** member of an **MCPARAMEX** structure, it indicates that a DCX series AT200 controller is installed.
DCXAT300	Value for the **ControllerType** member of an **MCPARAMEX** structure, it indicates that a DCX series AT300 controller is installed.
DCXPCI100	Value for the **ControllerType** member of an **MCPARAMEX** structure, it indicates that a DCX series PC100 controller is installed.
DCXPCI300	Value for the **ControllerType** member of an **MCPARAMEX** structure, it indicates that a DCX series PCI300 controller is installed.
MC_ABSOLUTE	Specifies that a position is in absolute units.
MC_ALL_AXES	When used in place of an axis number this constant implies that the command be performed on all installed axes. This option is not generally permitted on get type commands, i.e. to get the current position for all installed axes you should issue an individual **MCGetPositionEx( )** call for each axis.
MC_BLOCK_CANCEL	Argument to **MCBlockBegin( )** function canceling any commands queued (but not yet executed) as a result of a previous call to **MCBlockBegin( )**.
MC_BLOCK_COMPOUND	Argument to **MCBlockBegin( )** function specifying this block as a compound command block. Commands will not be executed until the **MCBlockEnd( )** command is issued.
MC_BLOCK_CONTR_CCW	Argument to **MCBlockBegin( )** function specifying this block as a contour path counter-clockwise arc (valid only for controllers that support contouring).
MC_BLOCK_CONTR_CW	Argument to **MCBlockBegin( )** function specifying this block as a contour path clockwise arc (valid only for controllers that support contouring).

Constant	Description
MC_BLOCK_CONTR_LIN	Argument to **MCBlockBegin( )** function specifying this block as a contour path linear motion (valid only for controllers that support contouring).
MC_BLOCK_CONTR_USER	Argument to **MCBlockBegin( )** function specifying this block as a contour path user defined motion (valid only for controllers that support contouring).
MC_BLOCK_MACRO	Argument to **MCBlockBegin( )** function specifying this block as a macro command. All commands up to the **MCBlockEnd( )** will be included in the macro.
MC_BLOCK_RESETM	Argument to **MCBlockBegin( )** function that will cause macro storage to be cleared.
MC_BLOCK_TASK	Argument to **MCBlockBegin( )** function specifying this block as separate task (valid only for controllers that support multi-tasking).
MC_CAPTURE_ACTUAL	Used to select the actual position data from the data capture functions.
MC_CAPTURE_ADVANCED	capture flag for CaptureModes member of **MCAXISCONFIG**
MC_CAPTURE_ERROR	Used to select the following error data from the data capture functions.
MC_CAPTURE_OPTIMAL	Used to select the optimal position data from the data capture functions.
MC_CAPTURE_TORQUE	Used to select the torque data from the data capture functions.
MC_COMPARE_DISABLE	Disable position compare mode, also used to disable compare output on position match.
MC_COMPARE_ENABLE	Enable position compare mode.
MC_COMPARE_STATIC	Set compare output on position match.
MC_COMPARE_TOGGLE	Toggle compare output on position match.
MC_COMPARE_INVERT	Set compare output on position match.
MC_COMPARE_ONESHOT	Set compare output on position match.
MC_COUNT_CAPTURE	Return the current captured position count.
MC_COUNT_COMPARE	Return the current compare position count.
MC_COUNT_CONTOUR	Return the current contour position count.
MC_COUNT_FILTER	Return the current digital filter coefficient count.
MC_COUNT_FILTERMAX	Return the maximum digital filter size supported.
MC_CURRENT_FULL	Restores a stepper motor current to full power. Commonly used to restore full power, prior to driving, following a reduced current setting while a stepper motor was idle. This constant is used to set the value of the **Current** member of a **MCMOTIONEX** structure.
MC_CURRENT_HALF	Reduces stepper motor current to half power. Commonly used to reduce heating when a stepper motor is not driving. This constant is used to set the value of the **Current** member of a **MCMOTIONEX** structure.

Constant	Description
MC_DATA_ACTUAL	see MC_CAPTURE_ACTUAL.
MC_DATA_ERROR	see MC_CAPTURE_ERROR.
MC_DATA_OPTIMAL	see MC_CAPTURE_OPTIMAL.
MC_DIO_FIXED	Indicates that a digital I/O channel's I/O state (i.e. input or output) is fixed, and may not be changed with **MCConfigureDigitalIO( )**.
MC_DIO_HIGH	Configures a digital I/O channel for high true logic level when used as an argument to **MCConfigureDigitalIO( )**.
MC_DIO_INPUT	Configures a digital I/O channel for input when used as an argument to **MCConfigureDigitalIO( )**.
MC_DIO_LATCH	Configures a digital input channel for input latching when used as an argument to **MCConfigureDigitalIO( )**.
MC_DIO_LATCHABLE	Indicates that a digital I/O channel may be configured for latched input using **MCConfigureDigitalIO( )**.
MC_DIO_LOW	Configures a digital I/O channel for low true logic level when used as an argument to **MCConfigureDigitalIO( )**.
MC_DIO_OUTPUT	Configures a digital I/O channel for output when used as an argument to **MCConfigureDigitalIO( )**.
MC_DIO_STEPPER	Indicates that a digital I/O channel is configured for driving a stepper motor on a DC2-PC or DC2-STN controller
MC_DIR_NEGATIVE	When operating in velocity mode this constant may be used as argument to **MCDirection( )** to select the negative travel direction. The physical relationship of MC_DIR_NEGATIVE to the actual direction of travel (or rotation) will depend upon your mechanical setup.
MC_DIR_POSITIVE	When operating in velocity mode this constant may be used as argument to **MCDirection( )** to select the positive travel direction. The physical relationship of MC_DIR_POSITIVE to the actual direction of travel (or rotation) will depend upon your mechanical setup.
MC_ENC_FAULT_AUX	Enable encoder fault detection for the auxiliary encoder
MC_ENC_FAULT_PRI	Enable encoder fault detection for the primary encoder
MC_IM_CLOSEDLOOP	Selects the normal (open loop) input mode for MC360 Stepper Modules.
MC_IM_OPENLOOP	Selects the closed-loop input mode for MC360 Stepper Modules.
MC_INT_FREEZE	Selects the wait until move complete mode for the integral term option.
MC_INT_NORMAL	Selects the normal (always active) mode for the integral term option.
MC_INT_ZERO	Selects the zero and wait until move complete mode for the integral term option.
MC_LIMIT_ABRUPT	Selects abrupt stop mode when a limit is tripped.
MC_LIMIT_BOTH	Enables both the positive and negative limits.
MC_LIMIT_INVERT	Inverts limit logic mode for hard limits.

Constant	Description
MC_LIMIT_MINUS	Enables the negative limit for hard and soft limits.
MC_LIMIT_OFF	Selects axis off mode when a limit is tripped.
MC_LIMIT_PLUS	Enables the positive limit for hard and soft limits.
MC_LIMIT_SMOOTH	Selects smooth stop mode when a limit is tripped.
MC_LRN_POSITION	When used as an argument to the **MCLearnPoint( )** function, this mode will cause the actual position of the axis to be stored in point memory.
MC_LRN_TARGET	When used as an argument to the **MCLearnPoint( )** function, this mode will cause the current target position of the axis to be stored in point memory.
MC_MAX_ID	Specifies the maximum allowable value for the ID parameter to the **MCOpen( )** call, where 0 <= ID <= MC_MAX_ID.
MC_MODE_CONTOUR	Selects the contouring mode of operation for an axis when used as an argument to **MCSetOperatingMode( )**.
MC_MODE_GAIN	Selects the gain mode of operation for an axis when used as an argument to **MCSetOperatingMode( )**.
MC_MODE_POSITION	Selects the position mode of operation for an axis when used as an argument to **MCSetOperatingMode( )**.
MC_MODE_TORQUE	Selects the torque mode of operation for an axis when used as an argument to **MCSetOperatingMode( )**.
MC_MODE_UNKNOWN	Return value from **MCGetOperatingMode( )** when it is unable to determine the current operating mode.
MC_MODE_VELOCITY	Selects the velocity mode of operation for an axis when used as an argument to **MCSetOperatingMode( )**.
MC_OM_BIPOLAR	Selects the bipolar output mode for MC200 Advanced Servo Modules.
MC_OM_CW_CCW	Selects the clockwise - counterclockwise output mode for MC260 Advanced Stepper Modules.
MC_OM_PULSE_DIR	Selects the pulse and direction output mode for MC260 Advanced Stepper Modules.
MC_OM_UNIPOLAR	Selects the unipolar output mode for MC200 Advanced Servo Modules.
MC_OPEN_ASCII	When used as an argument to the **MCOpen( )** function it specifies that a controller is to be open for ASCII (character) based communication.
MC_OPEN_BINARY	When used as an argument to the **MCOpen( )** function it specifies that a controller is to be open for binary communication.
MC_OPEN_EXCLUSIVE	This constant may be combined with either MC_OPEN_ASCII or MC_OPEN_BINARY for calls to **MCOpen( )** to prevent other applications from gaining access to the controller while it is open with an exclusive handle.
MC_PHASE_REV	Selects reverse phasing for the servo module output when used as an argument to **MCSetServoOutputPhase( )**.

Constant	Description
MC_PHASE_STD	Selects standard phasing for the servo module output when used as an argument to **MCSetServoOutputPhase( )**.
MC_PROF_PARABOLIC	This constant may be used as the value of the mode argument to the **MCSetProfile( )** API function. It selects the parabolic profile for acceleration and deceleration.
MC_PROF_SCURVE	This constant may be used as the value of the mode argument to the **MCSetProfile( )** API function. It selects the S-Curve profile for acceleration and deceleration.
MC_PROF_TRAPEZOID	This constant may be used as the value of the mode argument to the **MCSetProfile( )** API function. It selects the trapezoidal profile for acceleration and deceleration.
MC_PROF_UNKNOWN	This constant is returned by the **MCGetProfile( )** API function if it is unable to determine the present profile setting. The most likely cause is older firmware, contact PMC for information on firmware updates.
MC_RATE_HIGH	This constant is used as an argument to the **UpdateRate** member of an **MCFILTEREX** structure. For servo motors and closed-loop steppers, setting **UpdateRate** to this value sets the maximum feedback loop update rate. When used for an open-loop stepper motor, it sets the maximum pulse rate range. Please refer to your User Manual for product specific information.
MC_RATE_LOW	This constant is used as an argument to the **UpdateRate** member of an **MCFILTEREX** structure. For servo motors and closed-loop steppers, setting **UpdateRate** to this value sets the low feedback loop update rate. When used for an open-loop stepper motor, it sets the low pulse rate range. Please refer to your User Manual for product specific information.
MC_RATE_MEDIUM	This constant is used as an argument to the **UpdateRate** member of an **MCFILTEREX** structure. For servo motors and closed-loop steppers, setting **UpdateRate** to this value sets the middle feedback loop update rate. When used for an open-loop stepper motor, it sets the middle pulse rate range. Please refer to your User Manual for product specific information.
MC_RATE_UNKNOWN	Returned if MCAPI cannot determine the current rate.
MC_RELATIVE	Specifies that a position supplied is relative to the current axis position.
MC_STAT_ACCEL	Selects the Accelerating status bit (DC2 PC100 only).
MC_STAT_AMP_ENABLE	Selects the Amp Fault Enabled status bit (DCX controllers only).
MC_STAT_AMP_FAULT	Selects the Amp Fault status bit (DCX controllers only).
MC_STAT_AT_TARGET	Selects the At Target status bit (DC2 PC100 controllers only).
MC_STAT_AUX_ENC_FAULT	Selects the Auxiliary Encoder Fault status bit (MFX-PCI1000 controllers only).
MC_STAT_AUX_IDX_FND	Selects the Auxiliary Encoder Looking for Index status bit (MFX-PCI1000 controllers only).

Constant	Description
MC_STAT_BREAKPOINT	Selects the Breakpoint status bit.
MC_STAT_BUSY	Selects the Busy status bit (DCX controllers only). When set indicates that dual port memory is being refreshed.
MC_STAT_CAPTURE	Selects the Position Capture status bit (DC2 PC100 controllers only).
MC_STAT_DIR	Selects the Direction status bit.
MC_STAT_EDGE_FOUND	Selects the Edge Found status bit (DCX PCI controllers only).
MC_STAT_ERROR	Selects the Motor Error status bit.
MC_STAT_FOLLOWING	Selects the Following Error status bit (DCX controllers only).
MC_STAT_FULL_STEP	Selects the Full Step status bit (DCX controllers only).
MC_STAT_HALF_STEP	Selects the Half Step status bit (DCX controllers only).
MC_STAT_HOMED	Selects the Motor Homed status bit.
MC_STAT_INDEX_FOUND	Selects the Index Found status bit (DCX PCI controllers only).
MC_STAT_INP_AMP	Selects the Amp Fault Input status bit (DCX controllers only).
MC_STAT_INP_AUX	Selects the Auxiliary Encoder Index Input status bit (DCX AT200, DCX AT300, DCX PCI controllers only).
MC_STAT_INP_HOME	Selects the Home Input status bit (DCX controllers only).
MC_STAT_INP_INDEX	Selects the Index Input status bit (DCX controllers only).
MC_STAT_INP_MJOG	Selects the Minus Jog Input status bit (DCX PC100 / DCX AT100 controllers only).
MC_STAT_INP_MLIM	Selects the Minus Limit Input status bit (DCX controllers only).
MC_STAT_INP_PJOG	Selects the Plus Jog Input status bit (DCX PC100 / DCX AT100 controllers only).
MC_STAT_INP_PLIM	Selects the Plus Limit Input status bit (DCX controllers only).
MC_STAT_INP_USER1	Selects the User #1 Input status bit (DCX AT200, DCX AT300 controllers only).
MC_STAT_INP_USER2	Selects the User #2 Input status bit (DCX AT200, DCX AT300 controllers only).
MC_STAT_JOG_ENAB	Selects the Jogging Enabled status bit (DCX AT200, DCX AT300 controllers only).
MC_STAT_JOGGING	Selects the Motor Jogging status bit (DCX PC100 / DCX AT100 controllers only).
MC_STAT_LMT_ABORT	Selects the Abort Limit Mode status bit (DC2 PC100 controllers only).
MC_STAT_LMT_STOP	Selects the Stop Limit Mode status bit (DC2 PC100 controllers only).
MC_STAT_LOOK_AUX_IDX	Selects the Looking for Auxiliary Encoder Index status bit (MFX-PCI1000 controllers only).
MC_STAT_LOOK_EDGE	Selects the Looking for Edge status bit.
MC_STAT_LOOK_INDEX	Selects the Looking for Index status bit.
MC_STAT_MJOG_ENAB	Selects the Minus Jog Enable status bit (DCX PC100 / DCX AT100 controllers only).

Constant	Description
MC_STAT_MJOG_ON	Selects the Minus Jog On status bit (DCX PC100 / DCX AT100 controllers only).
MC_STAT_MLIM_ENAB	Selects the Minus Hard Limit Enable status bit.
MC_STAT_MLIM_TRIP	Selects the Minus Hard Limit Tripped status bit.
MC_STAT_MODE_ARC	Selects the Arc Mode status bit (DC2 PC100 controllers only).
MC_STAT_MODE_CNTR	Selects the Contouring Mode status bit (DC2 PC100 controllers only).
MC_STAT_MODE_LIN	Selects the Linear Mode status bit (DC2 PC100 controllers only).
MC_STAT_MODE_POS	Selects the Position Mode status bit (DC2 PC100 controllers only).
MC_STAT_MODE_SLAVE	Selects the Slave Mode status bit (DC2 PC100 controllers only).
MC_STAT_MODE_TRQE	Selects the Torque Mode status bit (DC2 PC100 controllers only).
MC_STAT_MODE_VEL	Selects the Velocity Mode status bit.
MC_STAT_MSOFT_ENAB	Selects the Minus Soft Limit Enable status bit (DCX AT200, DCX AT300, DCX PCI controllers only).
MC_STAT_MSOFT_TRIP	Selects the Minus Soft Limit Tripped status bit (DCX AT200, DCX AT300, DCX PCI controllers only).
MC_STAT_MTR_ENABLE	Selects the Motor On status bit.
MC_STAT_NULL	Selects the NULL Stepper Position status bit (DCX PCI300 controllers only).
MC_STAT_PHASE	Selects the Phase Reversed status bit.
MC_STAT_PJOG_ENAB	Selects the Plus Jog Enable status bit (DCX PC100 / DCX AT100 controllers only).
MC_STAT_PJOG_ON	Selects the Plus Jog On status bit (DCX PC100 / DCX AT100 controllers only).
MC_STAT_PLIM_ENAB	Selects the Plus Hard Limit Enable status bit.
MC_STAT_PLIM_TRIP	Selects the Plus Hard Limit Tripped status bit.
MC_STAT_POS_CAPT	Selects the Position Captured status bit (DCX PCI300 controllers only).
MC_STAT_PRI_ENC_FAULT	Selects the Primary Encoder Fault status bit (MFX-PCI1000 controllers only).
MC_STAT_PROG_DIR	Selects the Programmed Direction status bit (DC2 PC100 controllers only).
MC_STAT_PSOFT_ENAB	Selects the Plus Soft Limit Enable status bit (DCX AT200, DCX AT300, DCX PCI controllers only).
MC_STAT_PSOFT_TRIP	Selects the Plus Soft Limit Tripped status bit (DCX AT200, DCX AT300, DCX PCI controllers only).
MC_STAT_RECORD	Selects the Position status bit (DC2 PC100 controllers only).
MC_STAT_STOPPING	Selects the Stopping status bit (DC2 PC100 controllers only).

Constant	Description
MC_STAT_SYNC	Selects the Synchronize status bit (DC2 PC100 controllers only).
MC_STAT_TRAJ	Selects the Trajectory Complete status bit.
MC_STEP_FULL	Selects stepper motor full step operation.
MC_STEP_HALF	Selects stepper motor half step operation.
MC_TYPE_DOUBLE	Used with register get/set functions to select a double precision floating point data type.
MC_TYPE_FLOAT	Used with **pmccmdex( )** and register get/set functions to select a single precision floating point data type.
MC_TYPE_LONG	Used with register get/set functions to select a long integer (32-bit) data type.
MC_TYPE_NONE	Used with **pmccmdex( )** to specify no argument.
MC_TYPE_REG	Used with **pmccmdex( )** to select a register based argument.
MC_TYPE_SERVO	Indicates the axis is a servo motor – used with the **MCAXISCONFIG** structure.
MC_TYPE_STEPPER	Indicates the axis is a stepper motor – used with the **MCAXISCONFIG** structure.
MC_TYPE_STRING	Used with **pmccmdex( )** and register get/set functions to select a string data type.
MC100	Identifies a DC Servo axis with analog signal output.
MC110	Identifies a DC Servo axis with motor output.
MC150	Identifies a stepper motor axis.
MC160	Identifies a stepper motor with encoder axis.
MC200	Identifies an Advanced Servo axis with analog signal output.
MC210	Identifies an Advanced Servo axis with PWM motor output.
MC260	Identifies an Advanced Stepper axis.
MC300	Identifies a DSP-Based Servo axis with analog signal output.
MC302	Identifies a DSP-Based Dual Servo axes with dual analog signal outputs.
MC320	Identifies a DSP-Based Brushless-AC Servo axis with analog signal output.
MC360	Identifies a DSP-Based Stepper axis.
MC362	Identifies a DSP-Based Dual Stepper axes.
MC400	Identifies this axis as providing additional digital I/O channels (16).
MC500	Identifies this axis as providing additional analog channels.
MCERR_ALL_AXES	Error code indicating you may not use the constant MC_ALL_AXES with this function.
MCERR_ALLOC_MEM	There was a memory allocation error during a call to **MCOpen( )**. Try closing other Windows programs to free memory.
MCERR_AXIS_NUMBER	Error code indicating that the specified axis number is out of range.

Constant	Description
MCERR_AXIS_TYPE	Error code indicating that the function does not apply to the axis specified.
MCERR_COMM_PORT	Error code indicating and invalid constant value was given as the argument to a function.
MCERR_CONSTANT	Error code indicating and invalid constant value was given as the argument to a function.
MCERR_CONTROLLER	Error code indicating the controller handle is invalid.
MCERR_INIT_DRIVER	**MCOpen( )** was unable to initialize the device driver for this controller.
MCERR_MODE_UNAVAIL	The requested open mode for **MCOpen( )** was unavailable. This can occur when a non-multitasking controller is already open in a mode that is different from the requested mode.
MCERR_NO_CONTROLLER	Returned by **MCOpen( )** when no controller has been configured for this ID number.
MCERR_NO_REPLY	Error code indicating a controller failed to reply.
MCERR_NOERROR	Error code return value indicating that no errors have occurred.
MCERR_NOT_FOUND	Restore operation could not find data.
MCERR_NOT_INITIALIZED	An attempt was made to use a controller feature before that feature had been initialized.
MCERR_NOT_PRESENT	The controller hardware was not found during a call to **MCOpen( )**. Check the MCAPI settings with the setup program.
MCERR_NOTSUPPORTED	Error code indicating function is not supported by this controller. The MCAPI will handle this condition by ignoring requests to set this parameter and by returning a fixed default value for the parameter. You may, therefore, safely ignore this error.
MCERR_OBSOLETE	Error code indicating function is obsolete. See manual for updated function.
MCERR_OPEN_EXCLUSIVE	Returned by **MCOpen( )** when it is unable to satisfy a request for an exclusive handle. You cannot obtain an exclusive handle to a controller if there are other open handles for the controller at the time of your request.
MCERR_OUT_OF_HANDLES	Returned by **MCOpen( )** when the device driver has no more free handles it can assign to this request.
MCERR_RANGE	Error code indicating a parameter was out of range.
MCERR_REPLY_AXIS	Error code indicating the wrong axis number replied to a function.
MCERR_REPLY_COMMAND	Error code indicating the controller reply does not match the command.
MCERR_REPLY_SIZE	Error code indicating the length of a reply was incorrect (too many or too few bytes).
MCERR_TIMEOUT	A timeout occurred while attempting to send a command or read a reply from the controller.

Constant	Description
MCERR_UNKNOWN_REPLY	Error code indicating an unknown or unexpected reply was received from a controller.
MCERR_UNSUPPORTED_MODE	Return value from **MCOpen( )** when the requested mode is not supported for this controller/interface combination.
MCERR_WINDOW	Error code indicating a window handle is invalid.
MCERRMASK_AXIS	Error mask value for **MCErrorNotify( )** to enable error messages for out of range axis numbers and invalid usage of MC_ALL_AXES.
MCERRMASK_HANDLE	Error mask value for **MCErrorNotify( )** to enable error messages for invalid controller or window handles.
MCERRMASK_IO	Error mask value for **MCErrorNotify( )** to enable error messages for controller communication errors.
MCERRMASK_PARAMETER	Error mask value for **MCErrorNotify( )** to enable error messages for invalid or out of range parameters to MCAPI functions.
MCERRMASK_STANDARD	Collection of most common error mask values for **MCErrorNotify( )** (includes all errors except MCERRMASK_UNSUPPORTED) .
MCERRMASK_UNSUPPORTED	Error mask value for **MCErrorNotify( )** that enables error notification when a function is called that is not supported by the controller.
MF300	Identifies this axis as an RS-232 communications module. This module is not normally used with a controller installed in a PC adapter slot.
MF310	Identifies this axis as an IEEE-488 (GPIB) communications module. This module is not normally used with a controller installed in a PC adapter slot.
NO_CONTROLLER	One setting for the **ControllerType** member of an **MCPARAMEX** structure, it indicates that no controller is installed at this ID.
NO_MODULE	Identifies this axis as having no module installed.
NONE	One setting for the **ControllerType** member of a **MCPARAMEX** structure, it indicates that no controller is installed at this ID. This is an old constant - it is recommended that you use NO_CONTROLLER instead of NONE.

Appendix

C

Appendix C - Status Word Constants Lookup Table

This table is provided for cross-platform comparisons of **MCDecodeStatusEx( )** constants. Suppose you are using the MC_STAT_TRAJ status bit on a DC2-PC100 controller and plan to migrate to the more powerful DCX-PCI300 controller. Locate the constant in the leftmost column, read across the row to the DCX-PCI300 column and you will see that the MC_STAT_TRAJ constant is also supported for the DCX-PCI300.

You will also notice that the bit positions for MC_STAT_TRAJ on the DC2-PC100 and the DCX-PCI300 are different. If you had hard-coded this bit in your application, you would be forced to change your program to accommodate a different controller. By using **MCDecodeStatusEx( )** and the appropriate constants, no changes are required!

The numbers in the table represent the status word bit position for the specific controller. A dash indicates the constant is not supported for a particular controller.

Bit	DC2-PC DC2-STN	DCX-PC100 DCX-AT100	DCX-AT200 DCX-AT300	DCX-PCI100 DCX-PCI300	MFX-PCI1000
0	MC_STAT_MTR_ENABLE	MC_STAT_BUSY	MC_STAT_BUSY	MC_STAT_BUSY	MC_STAT_ERROR
1	MC_STAT_ERROR	MC_STAT_MTR_ENABLE	MC_STAT_MTR_ENABLE	MC_STAT_MTR_ENABLE	MC_STAT_MTR_ENABLE
2	MC_STAT_CAPTURE	MC_STAT_MODE_VEL	MC_STAT_AT_TARGET	MC_STAT_AT_TARGET	MC_STAT_AT_TARGET
3	MC_STAT_BREAKPOINT	MC_STAT_TRAJ	MC_STAT_TRAJ	MC_STAT_TRAJ	MC_STAT_TRAJ
4	MC_STAT_TRAJ	MC_STAT_DIR	MC_STAT_DIR	MC_STAT_DIR	MC_STAT_DIR
5	MC_STAT_STOPPING	MC_STAT_PHASE	MC_STAT_JOG_ENAB	- NONE -	MC_STAT_POS_CAPT
6	- NONE-	MC_STAT_HOMED	MC_STAT_HOMED	MC_STAT_HOMED	MC_STAT_BREAKPOINT
7	MC_STAT_DIR	MC_STAT_ERROR	MC_STAT_ERROR	MC_STAT_ERROR	- NONE -
8	MC_STAT_AT_TARGET	MC_STAT_LOOK_INDEX	MC_STAT_LOOK_INDEX	MC_STAT_LOOK_INDEX	MC_STAT_FOLLOWING
9	MC_STAT_PHASE	MC_STAT_LOOK_EDGE	MC_STAT_LOOK_EDGE	MC_STAT_LOOK_EDGE	MC_STAT_AMP_FAULT
10	MC_STAT_LOOK_INDEX	MC_STAT_FULL_STEP	- NONE-	MC_STAT_INDEX_FOUND	MC_STAT_PLIM_TRIP
11	MC_STAT_LOOK_EDGE	MC_STAT_HALF_STEP	- NONE -	MC_STAT_POS_CAPT	MC_STAT_MLIM_TRIP
12	MC_STAT_HOMED	MC_STAT_BREAKPOINT	MC_STAT_BREAKPOINT	MC_STAT_BREAKPOINT	MC_STAT_PSOFT_TRIP
13	MC_STAT_INP_HOME	MC_STAT_JOGGING	MC_STAT_FOLLOWING	MC_STAT_FOLLOWING	MC_STAT_MSOFT_TRIP
14	MC_STAT_RECORD	MC_STAT_AMP_ENABLE	MC_STAT_AMP_ENABLE	MC_STAT_AMP_ENABLE	MC_STAT_PRI_ENC_FAULT
15	MC_STAT_SYNC	MC_STAT_AMP_FAULT	MC_STAT_AMP_FAULT	MC_STAT_AMP_FAULT	MC_STAT_AUX_ENC_FAULT
16	MC_STAT_ACCEL	MC_STAT_PLIM_ENAB	MC_STAT_PLIM_ENAB	MC_STAT_PLIM_ENAB	- NONE -
17	MC_STAT_MODE_POS	MC_STAT_PLIM_TRIP	MC_STAT_PLIM_TRIP	MC_STAT_PLIM_TRIP	MC_STAT_LOOK_INDEX
18	MC_STAT_MODE_VEL	MC_STAT_MLIM_ENAB	MC_STAT_MLIM_ENAB	MC_STAT_MLIM_ENAB	MC_STAT_INDEX_FOUND
19	MC_STAT_MODE_TRQE	MC_STAT_MLIM_TRIP	MC_STAT_MLIM_TRIP	MC_STAT_MLIM_TRIP	MC_STAT_LOOK_AUX_IDX
20	MC_STAT_MODE_ARC	MC_STAT_PJOG_ENAB	MC_STAT_PSOFT_ENAB	MC_STAT_PSOFT_ENAB	MC_STAT_AUX_IDX_FND
21	MC_STAT_MODE_CNTR	MC_STAT_PJOG_ON	MC_STAT_PSOFT_TRIP	MC_STAT_PSOFT_TRIP	MC_STAT_HOMED
22	MC_STAT_MODE_SLAVE	MC_STAT_MJOG_ENAB	MC_STAT_MSOFT_ENAB	MC_STAT_MSOFT_ENAB	- NONE -
23	MC_STAT_MODE_LIN	MC_STAT_MJOG_ON	MC_STAT_MSOFT_TRIP	MC_STAT_MSOFT_TRIP	- NONE -
24	MC_STAT_LMT_ABORT	MC_STAT_INP_INDEX	MC_STAT_INP_INDEX	MC_STAT_INP_INDEX	MC_STAT_INP_INDEX
25	MC_STAT_LMT_STOP	MC_STAT_INP_HOME	MC_STAT_INP_HOME	MC_STAT_INP_HOME	MC_STAT_INP_HOME
26	MC_STAT_MLIM_TRIP	MC_STAT_INP_AMP	MC_STAT_INP_AMP	MC_STAT_INP_AMP	MC_STAT_INP_AUX
27	MC_STAT_MLIM_ENAB	- NONE -	MC_STAT_INP_AUX	MC_STAT_INP_AUX	MC_STAT_INP_AMP
28	MC_STAT_INP_MLIM	MC_STAT_INP_PLIM	MC_STAT_INP_PLIM	MC_STAT_INP_PLIM	MC_STAT_INP_PLIM
29	MC_STAT_PLIM_TRIP	MC_STAT_INP_MLIM	MC_STAT_INP_MLIM	MC_STAT_INP_MLIM	MC_STAT_INP_MLIM
30	MC_STAT_PLIM_ENAB	MC_STAT_INP_PJOG	MC_STAT_INP_USER1	MC_STAT_INP_NULL	- NONE -
31	MC_STAT_INP_PLIM	MC_STAT_INP_MJOG	MC_STAT_INP_USER2	- NONE -	- NONE -

Appendix

D

Appendix D - Motion Dialog Window Classes

The motion dialog window classes supplement the motion dialog functions to provide the programmer simple and effective tools to build attractive graphical user interfaces.

MCDLG_LEDCLASS

#include "mcdlg.h"

Creates a window with a small graphical LED and text label to the right of it. The LED window class is based on the checkbox style windows BUTTON class. To change the color of the LED send it a BM_SETCHECK message with a WPARAM of BST_CHECKED for the on color (default green), BST_UNCHECKED for the off color (default dark gray), or BST_INDETERMINATE for the error color (default red).

LED CLASS Styles

The LED class responds to the standard window styles (WS_xxx) and button styles (BS_xxx) applicable to checkbox windows. Use BS_LEFTTEXT to locate the text to the left of the LED graphic.

LED CLASS Messages

LEDM_GETCHECKCOLOR

Returns the current color of the "Checked" (on) state for the LED as a COLORREF.

```
   wParam = (WPARAM) 0;          // unused, must be 0
   lParam = (LPARAM) 0;          // unused, must be 0
```

LEDM_GETUNCHECKCOLOR

Returns the current color of the "Unchecked" (off) state for the LED as a COLORREF.

```
   wParam = (WPARAM) 0;          // unused, must be 0
   lParam = (LPARAM) 0;          // unused, must be 0
```

LEDM_GETINDETRMCOLOR

Returns the current color of the "Indeterminate" state for the LED as a COLORREF.

```
   wParam = (WPARAM) 0;          // unused, must be 0
   lParam = (LPARAM) 0;          // unused, must be 0
```

LEDM_SETCHECKCOLOR

Sets the color of the "Checked" (on) state for the LED. By default this color is bright green - RGB(0, 255, 0).

```
   wParam = (WPARAM) 0;          // TRUE to force an immediate redraw
   lParam = (LPARAM) rgbColor;   // COLORREF color value
```

LEDM_SETUNCHECKCOLOR

Sets the color of the "Unchecked" (off) state for the LED.

```
   wParam = (WPARAM) 0;          // TRUE to force an immediate redraw
   lParam = (LPARAM) rgbColor;   // COLORREF color value
```

LEDM_SETINDETRMCOLOR

Sets the color of the "Indeterminate" state for the LED. By default this color is bright red - RGB(255, 0, 0).

```
   wParam = (WPARAM) 0;          // TRUE to force an immediate redraw
   lParam = (LPARAM) rgbColor;   // COLORREF color value
```

MCDLG_READOUTCLASS

#include "mcdlg.h"

Creates a single line "readout" window, similar to a text box. By default the text is green on a black background, and the window font is scaled to the window size to make it easy to create large readouts. The READOUT window class is based on the Windows STATIC class. To change the displayed text of the READOUT the standard WM_SETTEXT message may be sent to the window.

READOUT CLASS Styles

The READOUT class responds to the standard window styles (WS_xxx) and static styles (SS_xxx) applicable to static windows. Use RDTS_LEFT, RDTS_CENTER, or RDTS_RIGHT to set the justification of the text within the window.

When you declare a READOUT in a dialog box template using the CONTROL statement the dialog box manager will set the READOUT font to the default dialog box font. This can lead to undesirable behavior (i.e. the wrong size font). The READOUT class normally responds to the WM_SETFONT message (which is what the dialog box manager sends to mess things up), however if you specify the RDTS_DIALOGBOX style when creating the READOUT window it will ignore WM_SETFONT messages. See the CWDEMO sample program for an example.

READOUT CLASS Messages

RDTM_GETTEXTCOLOR

Returns the current color of the readout text (default green) as a COLORREF.

```
   wParam = (WPARAM) 0;          // unused, must be 0
   lParam = (LPARAM) 0;          // unused, must be 0
```

RDTM_GETBKCOLOR

Returns the current color of the readout background (default black) as a COLORREF.

```
   wParam = (WPARAM) 0;          // unused, must be 0
   lParam = (LPARAM) 0;          // unused, must be 0
```

RDTM_SETTEXTCOLOR

Sets the color of the readout text.

```
   wParam = (WPARAM) 0;          // TRUE to force an immediate redraw
   lParam = (LPARAM) rgbColor;   // COLORREF color value
```

RDTM_SETBKCOLOR

Sets the color of the readout background.

```
   wParam = (WPARAM) 0;          // TRUE to force an immediate redraw
   lParam = (LPARAM) rgbColor;   // COLORREF color value
```

Appendix

E

Appendix E - Printing a PDF Document

Introduction to PDF
PDF stands for Portable Document Format. It is the de facto standard for transporting electronic documents. PDF files are based on the PostScript language imaging model. This enables sharp, color-precise printing on almost all printers.

Printing a complete PDF document
It is **not recommended** that large PDF documents be printed on personal computer printers. The 'wear and tear' incurred by these units, coupled with the difficulties of two sided printing, typically resulting in degraded performance of the printer and a whole lot of wasted paper. PMC recommends that PDF document be printer by a full service print shop that uses digital (computer controlled) copy systems with paper collating/sorting capability.

Printing selected pages of a PDF document
While viewing a PDF document with Adobe Reader (or Adobe Acrobat), any page or range of pages can be printed by a personal computer printer by:

Selecting the printer icon on the tool bar
Selecting **Print** from the Adobe **File** menu

Paper
The selection of the paper type to be used for printing a PDF document should be based on the target market for the document. For a user's manual with extensive graphics that is printed on both sides of a page the minimum recommended paper type is 24 pound. A heavier paper stock (26 – 30 pound) will reduce the 'bleed through' inherent with printed graphics. Typically the front and back cover pages are printed on heavy paper stock (50 to 60 pound).

Binding
Unlike the binding of a book or catalog, a user's manual distributed in as a PDF file will typically use 'comb' or 'coil' binding. This service is provided by most full service print shops. Coil binding is

suitable for documents with no more than 100 pieces of paper (24 pound). Comb binding is acceptable for documents with as many as 300 pieces of paper (24 pound). Most print shops stock a wide variety of 'combs'. The print shop can recommend the appropriate 'comb' based on the number of pages.

Pricing
The final cost for printing and binding a PDF document is based on:

- Quantity per print run
- Number of pages
- Paper type

The price range for printing and binding a PDF document similar to this user manual will be $15 to $30 (printed in Black & White) in quantities of 1 to 10 pieces.

Obtaining a Word 2000 version of this user manual
This user document was written using Microsoft's Word 2000. Qualified OEM's, Distributors, and Value Added Reps (VAR's) can obtain a copy of this document for

- Editing
- Customization
- Language translation.

Please contact Precision MicroControl to obtain a Word 2000 version of this document.

Index

Index

D

E

F

G

H

I

J

L

M

P

Q

R

S

T

U

V

W

Y

Z

Precision MicroControl Corporation
2075-N Corte del Nogal
Carlsbad, CA 92009-1415 USA

Tel: (760) 930-0101
Fax: (760) 930-0222

www.pmccorp.com

Information: info@pmccorp.com
Technical Support: support@pmccorp.com