```
         code for executing a profile and tracing
       captured in this example could be used for tuning th
      trace buffer wrap mode to a one time trace
    raceMode (hAxis1, PMDTraceOneTime);
   t the processor variables that we want to capture
  tTraceVariable (hAxis1, PMDTraceVariable1, PMDAx
 etTraceVariable (hAxis1, PMDTraceVariable2, PMDAxi
SetTraceVariable (hAxis1, PMDTraceVariable3, PMDAxis
// set the trace to begin when we issue the next update command
SetTraceStart (hAxis1, PMDTraceConditionNextUpdate);
// set the trace to stop when the MotionComplete event occurs
SetTraceStop (hAxis1, PMDTraceConditionEventStatus,
  PMDEventMotionCompleteBit, PMDTraceStateHigh);
 etProfileMode (hAxis1, PMDTrapezoidalProfile);
  et the profile parameters
   Position(hAxis1, 200000);
    elocity(hAxis1, 0x200000);
      eleration(hAxis1, 0x1000);
        eration(hAxis1, 0x1000);
```

C-Motion Magellan

Programming Reference

Revision 4.3/ July 2023

Performance Motion Devices, Inc.

80 Central Street, Boxborough, MA 01719

www.pmdcorp.com

NOTICE

Warranty

Performance Motion Devices, Inc. warrants that its products shall substantially comply with the specifications applicable at the time of sale, provided that this warranty does not extend to any use of any Performance Motion Devices, Inc. product in an Unauthorized Application (as defined below). Except as specifically provided in this paragraph, each Performance Motion Devices, Inc. product is provided "as is" and without warranty of any type, including without limitation implied warranties of merchantability and fitness for any particular purpose.

Performance Motion Devices, Inc. reserves the right to modify its products, and to discontinue any product or service, without notice and advises customers to obtain the latest version of relevant information (including without limitation product specifications) before placing orders to verify the performance capabilities of the products being purchased. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement and limitation of liability.

Unauthorized Applications

Performance Motion Devices, Inc. products are not designed, approved or warranted for use in any application where failure of the Performance Motion Devices, Inc. product could result in death, personal injury or significant property or environmental damage (each, an "Unauthorized Application"). By way of example and not limitation, a life support system, an aircraft control system and a motor vehicle control system would all be considered "Unauthorized Applications" and use of a Performance Motion Devices, Inc. product in such a system would not be warranted or approved by Performance Motion Devices, Inc.

By using any Performance Motion Devices, Inc. product in connection with an Unauthorized Application, the customer agrees to defend, indemnify and hold harmless Performance Motion Devices, Inc., its officers, directors, employees and agents, from and against any and all claims, losses, liabilities, damages, costs and expenses, including without limitation reasonable attorneys' fees, (collectively, "Damages") arising out of or relating to such use, including without limitation any Damages arising out of the failure of the Performance Motion Devices, Inc. product to conform to specifications.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent procedural hazards.

Disclaimer

Performance Motion Devices, Inc. assumes no liability for applications assistance or customer product design. Performance Motion Devices, Inc. does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of Performance Motion Devices, Inc. covering or relating to any combination, machine, or process in which such products or services might be or are used. Performance Motion Devices, Inc.'s publication of information regarding any third party's products or services does not constitute Performance Motion Devices, Inc.'s approval, warranty or endorsement thereof.

Patents

Performance Motion Devices, Inc. may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Patents and/or pending patent applications of Performance Motion Devices, Inc. are listed at https://www.pmdcorp.com/company/patents.

Related Documents

Magellan Motion Control IC User Guide

Complete description of the Magellan Motion Control IC features and functions with detailed theory of its operation.

MC58000 Electrical Specification

For DC brush, brushless DC, Microstepping, and Pulse & Direction motion control ICs

MC55000 Electrical Specification

For Pulse & Direction motion control ICs

MC58113 Electrical Specification

For single chip DC Brush, brushless DC, microstepping, and Pulse & Direction motion control ICs with closed loop current control.

Other Documents

ION/CME N-Series Digital Drive User Manual

How to install and configure ION/CME N-Series Digital Drives.

ION Digital Drive User Manual

How to install and configure ION 500 and ION 3000 Digital Drives.

Prodigy-PC/104 Motion Card User Guide

How to install and configure the Prodigy-PC/104 motion board.

Prodigy/CME Standalone User Guide

How to install and configure the Prodigy/CME standalone motion board.

Prodigy/CME Machine-Controller User Guide

How to install and configure the Prodigy/CME machine controller motion board.

Table of Contents

This page intentionally left blank.

1. Introduction

1.1 Introduction

This manual documents C-Motion Magellan, which is a software library used to control and monitor Magellan and Juno-based PMD motion control products.

There are two other C-Motion versions; C-Motion PRP and C-Motion PRP II. All of these software systems are available in separate SDKs as detailed below:

- **C-Motion Magellan SDK** – an SDK (Software Developer Kit) for creating motion applications using the C/C++ programming language for PMD products that utilize a direct Magellan or Juno formatted protocol.
- **C-Motion PRP SDK** – an SDK for creating PC and downloadable user code for systems utilizing either a PRP (PMD Resource Access Protocol) protocol device or a Magellan/Juno protocol device. C-Motion PRP is also used in motion applications that will use the .NET (C#, VB) programming languages.
- **C-Motion PRP II SDK** – This SDK is similar to C-Motion PRP but is used with ION/CME N-Series ION Digital Drives. Compared to standard C-Motion PRP, C-Motion PRP II supports additional features such as multi-tasking, mailboxes, mutexes, and enhanced event management.

For detailed information on Magellan/Juno protocol C-Motion refer to the *C-Motion Magellan Programming Reference*. For detailed information on C-Motion PRP refer to the *C-Motion PRP Programming Reference*.

1.2 PMD Products and C-Motion Version

The following table shows the C-Motion versions that can be used with each PMD product family:

Product Family	Compatible C-Motion Versions
Magellan ICs	C-Motion Magellan, C-Motion PRP*
Juno ICs	C-Motion Magellan, C-Motion PRP*
ION/CME N-Series	C-Motion PRP II
ION 500	C-Motion Magellan, C-Motion PRP*
ION/CME 500	C-Motion PRP
ION 3000	C-Motion Magellan, C-Motion PRP*
Prodigy PC/104	C-Motion Magellan, C-Motion PRP*
Prodigy/CME PC/104	C-Motion PRP
Prodigy/CME Stand-Alone	C-Motion PRP
Prodigy/CME Machine-Controller	C-Motion PRP

*C-Motion PRP typically only used for .NET support, or if a mix of Magellan/Juno protocol and PRP protocol devices are attached.

1.3 Overview of C-Motion Magellan

1.3.1 Introduction

C-Motion Magellan is a “C” source code library that contains all the code required for communicating with the Magellan Motion Control IC.

C-Motion includes the following features:

- Axis virtualization.
- The ability to communicate to multiple Magellan Motion Control ICs.
- Can be easily linked to any “C/C++” application.

C-Motion callable functions are broken into two groups, those callable functions that encapsulate motion control IC specific commands, and those callable functions that encapsulate product-specific capabilities.

The motion control IC specific commands are detailed in Chapter 2, *Instruction Reference*. They are the primary commands that you will use to control the major motion features including profile generation, servo loop closure, motor output signal generation (PWM and analog), breakpoint processing, trace operations, and many other functions.

Each Magellan Motion Control IC command has a C-Motion command of the identical name, but prefaced by the letters “PMD.” For example, the Magellan command **SetPosition** is called **PMDSetPosition**.

1.3.2 Files

The following table lists the files that make up the C-Motion distribution.

C-Motion.h/C-Motion.c	Definition/declaration of the PMD Magellan command set
PMDpar.h/PMDpar.c	Parallel interface functions
PMDW32ser.h/PMDW32ser.c	Windows serial communication interface functions
PMDutil.h/PMDutil.c	General utility functions
PMDtrans.h/PMDtrans.c	Generic transport (interface) functions
PMDecode.h	Defines the PMD Magellan and C-Motion error codes
PMDocode.h	Defines the control codes for Magellan commands
PMDtypes.h	Defines the basic types required by C-Motion
PMDCAN.h/PMDCAN.c	CAN interface command/data transfer functions.
PMDIXXATCAN.c	CAN interface for IXXAT VCI (Virtual Can Interface) API
PMDNISPI.c	SPI interface for National Instruments USB-8452
PMDcommon.c	Miscellaneous procedures
PMDdevice.h	
PMDdiag.h/PMDdiag.c	Diagnostic functions
IXXAT*.*	IXXAT VCI include and library files
PLX*.*	PLX Technology (PCI) and library include files
NI*.*	National Instruments include and library files

1.3.3 Using C-Motion

C-Motion can be linked to your application code by including the above “C” source files in your application. Then, for any application source file that requires access to the motion control IC, include C-Motion.h. In addition, the required interfaces need to be defined as shown below. Only the required interfaces need to be included.

```
#define PMD_W32SERIAL_INTERFACE
// use this for a standard serial interface under Windows
```

```
#define PMD_PCI_INTERFACE
// use this for a standard PCI parallel interface under Windows

#define PMD_CAN_INTERFACE
// use this for a CAN interface under Windows
```

By customizing the base interface functions, C-Motion can be ported to virtually any hardware platform. An example would be a memory-mapped IO scheme that uses the parallel interface. This would be built using the PMDPar.c/.h source files as a basis.

The Magellan Motion Control IC Developer Kit board and the Prodigy-PCI Motion Card use the PCI interface chip provided by PLX Technology. To fully understand the interface mechanism, or to write your own interface software, you can download the PLX SDK. More information on the functionality and features can be found on the PLX website – http://www.plxtech.com – in the software development kits area.

C-Motion is a set of functions that encapsulate the motion control IC command set. Every command has as its first parameter an "axis handle." The axis handle is a structure containing information about the interface to the motion control IC and the axis number that the handle represents. Before communicating to the motion control IC, the axis handle must be initialized using the following sequence of commands:

```
// the axis handles
PMDAxisHandle hAxis1, hAxis2;

// open interface to PMD processor and initialize handle to axis one
PMDSetupAxisInterface_PCI( &hAxis1, PMDAxis1, 0 );

// initialize handle to the second axis
PMDCopyAxisInterface( &hAxis2, &hAxis1, PMDAxis2 );
```

The above is an example of initializing communication using the parallel communication interface. Each interface .c source file contains an example of initializing the interface. Once the axis handle has been initialized, any of the motion control IC commands can be executed.

The header file C-Motion.h includes the function prototypes for all motion control IC commands as implemented in C-Motion. See this file for the required parameters for each command. For information about the operation and purpose of each command, see Chapter 2, *Instruction Reference*.

Many functions require additional parameters. Some standard values are defined by C-Motion and can be used with the appropriate functions. See PMDtypes.h for a complete list of defined types. An example of calling one of the C-Motion functions with the pre-defined types is shown below:

```
PMDSetBreakpoint(&Axis2, PMDBreakpoint1, PMDAxis2, PMDBreakpointActionAbruptStop,
  PMDBreakpointActualPositionCrossed);
```

In a few cases commands must be directed explicitly to the Atlas amplifier associated with a Magellan control axis, examples are the GetVersion and Reset commands. In order to do so an axis handle must be opened for the Atlas amplifier itself, to do so for axis 2 the following call may be used:

```
PMDAxisHandle hAxis2, hAtlas2;
  PMDGetAtlasAxisHandle(&hAxis2, &hAtlas2);
```

1.3.4 C-Motion Functions

The table below describes the functions that are provided by C-Motion in addition to the standard chip command set.

C-Motion functions	Arguments	Function description
PMDSetupAxisInterface_PCI	*axis_handle* *axis_number* *board_number*	Used to setup an axis interface connection for communicating over a PCI bus.
PMDSetupAxisInterface_Serial	*axis_handle* *axis_number* *port_number*	Used to setup an axis interface connection for communicating over a RS232 or RS485 serial bus.
PMDSerial_SetConfig	*transport_data* *baud_rate* *parity*	Used for setting baud rate and parity for a serial axis. transport_data is a member of the axis handle struct, which must be cast to (PMDSerialIOData *). baud_rate is an integer. parity takes the same enumerated values as the Parity member of the Windows DCB struct.
PMDSetupAxisInterface_CAN	*axis_handle* *axis_number* *board_number*	Used to setup an axis interface connection for communicating over a CAN bus.
PMDSetupAxisInterface_Parallel	*axis_handle* *axis_number* *board_address*	Low level function used to setup an axis interface for parallel communications in an embedded system.
PMDSetupAxisInterface_SPI	*axis_handle* *axis_number* *device*	Used to setup an axis interface connection for communicating over an SPI bus.
PMDCloseAxisInterface	*axis_handle*	Should be called to terminate an interface connection.
PMDCopyAxisInterface	*dest_axis_handle* *src_axis_handle* *axis_number*	Used for opening an axis interface connection to the same device as used by src_axis_handle, but a different axis.
PMDGetErrorMessage	*ErrorCode*	Returns a character string representation of the corresponding PMD chip or C-Motion error code.
GetCMotionVersion	*MajorVersion* *MinorVersion*	Returns the major and minor version number of C-Motion.
PMDHardReset	*axis_handle*	This function causes a “hard” reset of the motion control IC. Unlike all other card-specific commands, this command is processed directly through the bus interface.
PMDReadDPRAM	*axis_handle* *data* *offset_in_dwords* *words_to_read*	This function reads directly from the onboard dual-port RAM via the bus interface (if applicable).
PMDWriteDPRAM	*axis_handle* *data* *offset_in_dwords* *words_to_write*	This function writes directly to the onboard dual-port RAM via the bus interface (if applicable).

1.3.5 Prodigy Motion Card Specific Functions

Several auxiliary functions are included in addition to the standard Magellan API commands for use with the Magellan-based Prodigy Motion Cards only. The functions are for configuring functions on the motion control board. The following table describes the functions. For more information, see the user guide for your motion control card.

C-Motion function	Arguments	Function description
PMDMBWriteDigitalOutput	*axis_handle, write_value*	This function writes to the eight general-purpose digital I/O signals (digitalOut0-7). Write_value holds the eight signals in its low order 8 bits.
PMDMBReadDigitalInput	*axis_handle, read_value*	This function reads the value of the signals DigitalIn0-7, and returns them in the low order 8 bits of read_value.
PMDMBReadDigitalOutput	*axis_handle, read_value*	This function reads the value of the signals DigitalOut0-7, and returns them in the low order 8 bits of read_value.
PMDMBSetAmplifierEnable	*axis handle, mask, write_value*	This function writes to the 4 amplifier enable signals (AmpEnable1-4) using mask and write_value. When a 1 appears in mask, the corresponding bit position in write_value is written to the corresponding signal. The values for mask and write_value are all 0- shifted; that is, they are stored in the lowest order 4 bits.
PMDMBGetAmplifierEnable	*axis_handle, read_value*	This function reads the values of AmpEnable 1-4, and returns them in the low order 4 bits of read_value.
PMDMBSetDACOutputEnable	*axis handle, write_value*	This function sets the DACOutputEnable status. A written value of 1 enables DAC output, while a written value of 0 disables DAC output.
PMDMBGetDACOutputEnable	*axis_handle, read_value*	This function reads the value of the DACOutputEnable function. A value of 1 indicates DAC output enabled; a value of 0 indicates DAC output disabled.
PMDMBSetWatchDog	*axis handle*	This function writes to the correct value to the watchdog register, so that for the next 104 milliseconds the card will not be reset by the watchdog circuitry.
PMDMBGetResetCause	*axis_handle, reset_cause*	This function returns the reset cause in the variable reset_-cause, reset_cause and also clears the reset condition.
PMDMBReadCardID	*axis_handle, card_ID*	This function returns the card ID, encoded as defined in the preceeding table.

1.4 Microsoft .NET Programming

1.4.1 Visual Basic Classes

The file PMDLibrary.vb defines a Visual Basic class for each of the opaque data types used in the PMD library:

PMDPeripheral, **PMDDevice**, **PMDAxis**, and **PMDMemory**. **PMDPeripheral** is inherited by a set of derived classes for each peripheral type: **PMDPeripheralCOM**, **PMDPeripheralCAN**, **PMDPeripheralPCI**, and

PMDPeripheralTCP. Each class takes care of allocating and freeing the memory used for the “handle” structures used in the C language interface.

The following example illustrates how to obtain a Magellan axis object connected to a serial port.

```
Public Class Examples
    Public Sub Example2()
        Dim periph As PMDPeripheral
        Dim Magellan As PMDDevice
        Dim axis2 As PMDAxis
```

```
            ' Open the connection on COM1, using appropriate serial port parameters
            periph = New PMDPeripheralCOM(1, PMDSerialBaud.Baud57600, _
            PMDSerialParity.None, PMDSerialStopBits.Bits1)

            ' Obtain a Magellan device object using the peripheral.
            Magellan = New PMDDevice(periph, PMDDeviceType.MotionProcessor)

            ' Finally instantiate an axis object for axis number 2.
            axis2 = New PMDAxis(Magellan, PMDAxisNumber.Axis2)

            ' Example VB-Motion operation: Get the event status
            Dim status As UInt16
            status = axis2.EventStatus
        End Sub
    End Class
```

1.4.2 Visual Basic Programming

The Visual Basic PMD Library is the interface from Microsoft Visual Basic .NET to the PMD C-Motion library for control of Magellan Motion Control ICs, which is documented in the *Magellan Motion Control IC Programming Reference*. The Visual Basic interface documented in that manual is similar to but not identical to that used for PRP devices. Basic language programming is supported only for Microsoft Windows hosts, C-Motion Engine programming must be done in the C language.

There are two parts to the Visual Basic interface code:

1 C-Motion.dll is a dynamically loadable library of all documented procedures in the PMD host libraries, including all C-Motion procedures.

2 PMDLibrary.vb is Visual Basic source code containing definitions and declarations for DLL procedures, enumerated types, and data structures supporting the use of C-Motion.dll from Visual Basic. PMDLibrary.vb should be included in any Visual Basic project for PRP or Magellan device control.

Both debug and release versions of C-Motion.dll are provided in directories CMESDK\HostCode\Debug and CMESDK\HostCode\Release, respectively. The library input file C-Motion.lib is also provided so that C-Motion.dll may be used with C/C++ language programs. When compiling C/C++ programs to be linked against the DLL the preprocessor symbol PMD_IMPORTS must be defined.

C-Motion.dll must be in the executable path when using it, either from a C or a Visual Basic program. Frequently the easiest and safest way of doing this is to put it in the same directory as the executable file.

PMDLibrary.vb is located in the directory CMESDK\HostCode\DotNet.

1.4.3 Visual Basic Classes

The file PMDLibrary.vb defines a Visual Basic class for each of the opaque data types used in the PMD library: **PMDPeripheral**, **PMDDevice**, **PMDAxis**, and **PMDMemory**. **PMDPeripheral** is inherited by a set of derived classes for each peripheral type: **PMDPeripheralSerial**, **PMDPeripheralMultiDrop**, **PMDPeripheralPRP, PMDPeripheralCAN**, **PMDPeripheralSPI**, and **PMDPeripheralTCP**.

Each class takes care of allocating and freeing the memory used for the “handle” structures used in the C language interface. The first pointer argument to, for example, a **PMDPeriphHandle** in a C language procedure call is not needed because a method call for a particular **PMDPeripheral** object is used instead, and each object manages its own **PMDPeriphHandle**.

The “Open” procedures used in the C language interface are replaced in Visual Basic with constructor methods that take the same arguments in the same order, with the exception that the first pointer argument is not needed. “Close” methods are provided that call the C language “Close” procedures, however these procedures may also be called automatically as part of the finalization process when objects are garbage collected.

The following example demonstrates how to open a peripheral connection to a PRP device accessible by TCP/IP, and to access the resources of that device.

```
Public Class Examples
    Public Sub Example1()

' Allocate and open a peripheral connection to a PRP device using TCP/IP.
' Note that the arguments for the PMDPeripheralTCP object are the same as for the
' C language call PMDDeviceOpenPeriphTCP, except that the first argument for the peripheral
' struct pointer and the second argument for the device are not used.
' The standard .NET class for IP addresses is used instead of a numeric IP address.
' DEFAULT_ETHERNET_PORT is a constant defined in PMDLibrary.vb for the default
' TCP port used for commands by the PRP device.
' 1000 is a timeout value in milliseconds.
Dim periph As New PMDPeripheralTCP(System.Net.IPAddress.Parse("192.168.0.27"), _
                                           DEFAULT_ETHERNET_PORT, _
                                           1000)

' Now allocate and connect a device object using the newly opened peripheral.
' Instead of using two different names the second argument specifies whether a
' PRP device or attached Magellan device is expected.
Dim DevCME As New PMDDevice(periph, PMDDeviceType.ResourceProtocol)

' Once the PRP device is open we can obtain an axis object, which may be used
' for any C-Motion commands. Notice that the enumerated value used to specify the axis is
' called "Axis1" instead of "PMDAxis1" because the enumeration name already includes
‘ the “PMD” prefix.
Dim axis1 As New PMDAxis(DevCME, PMDAxisNumber.Axis1)

' C-Motion procedures returning a single value become class properties, and may be
' retrieved or set by using an assignment. The "Get" or "Set" part of the name is dropped.
Dim pos As Int32
pos = axis1.ActualPosition

' The following line sets the actual position of the axis to zero.
axis1.ActualPosition = 0

' Properties may accept parameters, for example the CurrentLoop parameter is used to set
' control gains for the current loops, and takes two parameters. This example sets
' the proportional gain for phaseA to 1000
axis1.CurrentLoop(PMDCurrentLoopNumber.PhaseA,  _
PMDCurrentLoopParameter.ProportionalGain) = 1000

' C-Motion procedures returning multiple values become Sub methods, and return their
' values using ByRef parameters. The "Get" and "Set" parts of the names are the same as
' in the C language binding.
Dim MPmajor, MPminor, NumberAxes, special, custom, family As UInt16
Dim MotorType As PMDMotorTypeVersion
axis1.GetVersion(family, MotorType, NumberAxes, special, custom, MPmajor, MPminor)

' If the objects opened here are not explicitly closed they will be closed by the
' garbage collector.
  End Sub
End Class
```

Several general points about the translation from C to Visual Basic are shown in the example:

- Argument type and order are the same, except that the initial “handle” pointer argument is not needed. The null device pointer used to indicate that a peripheral is opened on the local device is also not needed.
- “Get/Set” procedures returning a single argument become object properties, with parameters if needed. The property name does not contain “Get” or “Set”, or the “PMD” prefix.
- Procedures returning or setting multiple values are implemented as Sub methods, returning values via ByRef parameters. “Get” or “Set” is retained in the names, but the “PMD” prefix is not.

- Enumerated value names do not use the "PMD" prefix, but the enumeration names do.
- Procedures reading or writing array data through C pointers instead take Visual Basic arrays of the appropriate type.

1.4.4 C# Programming

The C# language is very similar to the VB language. A C# PMD program uses the PMDLibrary.dll created by the ClassLibrary project located in CMESDK\HostCode\DotNet\ClassLibrary. An example C# PMD program can be found in CMESDK\HostCode\DotNet\CSTestApp.

1.4.5 Error Handling

Almost all of the PMD C language library procedures return an error code to indicate success or failure. The Visual Basic versions of these procedures instead throw an exception if the wrapped DLL procedures return an error code. The exception message will contain the error number and a short description of the error. The Data member of the exception will contain the error number as an enumeration of type **PMDresult**, associated with the key "PMDresult", so that structured exception handling may be used to appropriately handle errors.

The following example commands a PRP device to reset, and then ignores the expected error return on the next command:

```
dev.Reset()
Try
     Dim major, minor As UInt32
     dev.Version(major, minor)
Catch ex As Exception When ex.Data("PMDresult").Equals(PMDresult.ERR_RP_Reset)
' Ignore the expected error
 End Try
```

Any errors that are not caught will cause the application to display a popup window displaying an error message, including the error number and description, and a stack trace with file names and line numbers. The popup window allows a user to continue, ignoring the error, or to abort the application.

While popup windows are useful for debugging, any application controlling motors should be designed to recover gracefully and safely from any foreseeable error condition, and it is recommended to use Try blocks liberally to make applications more robust.

2. Instruction Reference

2.1 How to Use This Reference

The instructions are arranged alphabetically, except that all "Set/Get" pairs (for example, **SetVelocity** and **GetVelocity**) are described together. Each description begins on a new page and most occupy no more than a single page. Each page is organized as follows:

Name	The instruction mnemonic is shown at the left, its hexadecimal code at the right.
Syntax	The instruction mnemonic (in bold) and its required arguments (in italic) are shown with all arguments separated by spaces. Note that depending on the product being used the axis argument may or may not be needed.
Buffered	Certain parameters and other data written to the motion control IC are buffered. That is, they are not acted upon until the next **Update** or **MultiUpdate** command is executed. These parameters are identified by the word "buffered" in the instruction heading.
Motor Types	The motor types to which this command applies. Supported motor types are printed in black; unsupported motor types for the command are greyed out.
Arguments	There are two types of arguments: encoded-field and numeric. Encoded-field arguments are packed into a single 16-bit data word, except for axis, which occupies bits 8–9 of the instruction word. The name of the argument (in italic) is that shown in the generic syntax. Instance (in italic) is the mnemonic used to represent the data value. Encoding is the value assigned to the field for that instance. For numeric arguments, the parameter value, the type (signed or unsigned integer), and the range of acceptable values are given. Numeric arguments may require one or two data words. For 32-bit arguments, the high-order part is transmitted first.
Packet Structure	This is a graphic representation of the 16-bit words transmitted in the packet: the instruction, which is identified by its name, followed by 1, 2, or 3 data words. Bit numbers are shown directly below each word. For each field in a word, only the high and low bits are shown. For 32-bit numeric data, the high-order bits are numbered from 16 to 31, the low-order bits from 0 to 15. The hex code of the instruction is shown in boldface. Argument names are shown in their respective words or fields. For data words, the direction of transfer—read or write—is shown at the left of the word's diagram. Unused bits are shaded. All unused bits must be 0 in data words and instructions sent (written) to the motion control IC. In the case of a Magellan controlling an Atlas amplifier, an axis field with bit 5 set is used to indicate that a command should be passed directly to the Atlas connected to the axis indicated by the lower 4 axis bits, and the result returned.
Description	Describes what the instruction does and any special information relating to the instruction.
Atlas	Describes any communication to an associated Atlas amplifier as a result of the instruction. Atlas operation is quite transparent, but extra SPI communication can significantly slow down Magellan command processing because a result must be received from Atlas before it is passed on to the Magellan host. Any comments in this section do not apply to any Magellan axis not connected to an Atlas amplifier. This section will not be present in the case of commands without any Atlas implications. For more information on the behavior of Atlas commands, see the *Atlas Digital Amplifier Complete Technical Reference*.
Restrictions	Describes the circumstances in which the instruction is not valid, that is, when it should not be issued. For example, velocity, acceleration, deceleration, and jerk parameters may not be issued while an S-curve profile is being executed.
C-Motion API	The syntax of the C function call in the PMD C-Motion library that implements this motion control IC command.
VB-Motion API	The Visual Basic syntax for the function in the PMD VB-Motion library that implements this motion control IC command. Properties and methods are shown with their associated root object name separated by a period.
see	Refers to related instructions.

AdjustActualPosition F5h

Syntax

AdjustActualPosition *axis position*

Motor Types

DC Brush	**Brushless DC**	**Microstepping**	**Pulse & Direction**

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	**Range**	**Scaling**	**Units**
position	signed 32 bits	-2^{31} to $2^{31}-1$	unity	counts microsteps

Packet Structure

AdjustActualPosition

0	*axis*	**F5**h
15 12	11 8	7 0

First data word

write	*position* (high-order part)
	31 16

Second data word

write	*position* (low-order part)
	15 0

Description

The *position* specified as the parameter to **AdjustActualPosition** is summed with the actual position register (encoder position) for the specified *axis*. This has the effect of adding or subtracting an offset to the current actual position. At the same time, the commanded position is replaced by the new actual position value minus the position error. This prevents a servo "bump" when the new axis position is established. The destination position (see **SetPosition** (p. 172)) is also modified by this amount so that no trajectory motion will occur when a trajectory update is performed. In effect, this command establishes a new reference position from which subsequent positions can be calculated. It is commonly used to set a known reference position after a homing procedure.

On axes configured for stepping and microstepping motors, the position error is zeroed by this command.

AdjustActualPosition takes effect immediately; it is not buffered.

Restrictions

C-Motion API

```
PMDresult PMDAdjustActualPosition(PMDAxisInterface axis_intf,
                                   PMDint32 position)
```

VB-Motion API

```
MagellanAxis.AdjustActualPosition([in] position)
```

see

GetPositionError (p. 51), **GetActualVelocity** (p. 27), **Set/GetActualPositionUnits** (p. 87), **Set/GetActualPosition** (p. 85)

CalibrateAnalog 6Fh

Syntax

CalibrateAnalog *axis position*

Motor Types

DC Brush	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
option	*leg currents*	0
	(Reserved)	1-3
	sin/cos	4

Returned data

None

Packet Structure

		axis	**6F**h
	15 … 12	11 … 8	7 … 0

First data word

write	*option*
	15 … 0

Description

The **CalibrateAnalog** command is used to adjust the adjustable offsets for some analog input channels. The option argument controls the set of analog channels calibrated.

The **CalibrateAnalog** command clears the calibrated bit (bit 0) in the drive status register. The bit is set when the calibration process is complete.

For leg currents the calibration process assumes that the actual input to the analog channels will be zero. For the leg current sensors it is generally sufficient to set the motor command to zero and ensure that the motor is not moving. Whether motor output should be enabled or not depends on external circuitry. Calibration is accomplished by averaging a number of readings. 100 ms after sending the command the process may be assumed to be complete.

For sin/cos encoders the encoder should be moving during the calibration process; multiple complete electrical rotations are required to complete the calibration. The time taken depends on encoder motion. The **GetDriveStatus** command should be called repeatedly to determine when calibration is complete.

Restrictions

This command is supported only by products with leg current sensing or sin/cos encoder input. Consult the appropriate product user guide.

C-Motion API

```
PMDresult PMDCalibrateAnalog(PMDAxisInterface axis_intf,
                             PMDuint16 option);
```

VB-Motion AP

```
MagellanAxis.CalibrateAnalog( [in] option )
```

see

GetDriveStatus (p. 38), **Set/GetAnalogCalibration** (p. 88), **ReadAnalog** (p. 71)

ClearDriveFaultStatus 6Ch

Syntax

ClearDriveFaultStatus *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet Structure

ClearDriveFaultStatus

15–12	11–8	7–0
0	*axis*	**6C**h

Description

ClearDriveFaultStatus clears all bits in the Drive Fault Status register. For ION, it should be executed after power-up, after using **GetDriveFaultStatus** to examine if any hard faults caused the power cycle. For other products, **ClearDriveFaultStatus** should be used after determining the cause of a Drive Exception event, before re-enabling output.

Atlas

This command is relayed to any attached Atlas amplifier before being applied to internal Magellan state.

Note that the Atlas Motor Type Mismatch bit, which is maintained by Magellan, may not be cleared by this command. That bit may be cleared by **SetMotorType**.

Restrictions

This command is not available in products that do not include drive amplifier support.

For ION, this command can only be executed when motor output is disabled (e.g., immediately after power-up or reset).

C-Motion API

```
PMDresult PMDClearDriveFaultStatus (PMDAxisInterface axis_intf)
```

VB-Motion API

```
MagellanAxis.ClearDriveFaultStatus()
```

see

GetDriveFaultStatus (p. 36)
SetMotorType (p. 154)

ClearInterrupt ACh

Syntax

ClearInterrupt *axis*

Motor Types

DC Brush	**Brushless DC**	**Microstepping**	**Pulse & Direction**

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet Structure

ClearInterrupt

0	*axis*	**4C**h
15 12	11 8	7 0

Description

ClearInterrupt resets the /HostInterrupt signal to its inactive state. If interrupts are still pending, the /HostInterrupt line will return to its active state within one chip cycle. See **Set/GetSampleTime** (p. 180) for information on chip cycle timing. This command is used after an interrupt has been recognized and processed by the host; it does not affect the Event Status register. The **ResetEventStatus** command should be issued prior to the **ClearInterrupt** command to clear the condition that generated the interrupt. The **ClearInterrupt** command has no effect if it is executed when no interrupts are pending.

When communicating using CAN, this command resets the interrupt message sent flag. When an interrupt is triggered on an ***axis***, a single interrupt message is sent and no further messages will be sent by that ***axis*** until this command is issued.

When serial or parallel communication is used, the axis number is not used.

Restrictions

For products without a /HostInterrupt line, this command is still applicable to the CAN communications. For products without a /HostInterrupt line or CAN communications, this command is not used.

C-Motion API

```
PMDresult PMDClearInterrupt (PMDAxisInterface axis_intf)
```

VB-Motion API

```
MagellanAxis.ClearInterrupt()
```

see

GetInterruptAxis (p. 49), **Set/GetInterruptMask** (p. 146), **ResetEventStatus** (p. 80).

ClearPositionError buffered 47h

Syntax

ClearPositionError *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet Structure

ClearPositionError

0	*axis*	**47**h
15 12	11 8	7 0

Description

ClearPositionError sets the profile's commanded position equal to the actual position (encoder input), thereby clearing the position error for the specified ***axis***. This command can be used when the axis is at rest, or when it is moving.

Restrictions

ClearPositionError is a buffered command. The new value set will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory bit set in the update mask commands.

This command should not be sent while the chip is executing a move using the S-curve profile mode.

C-Motion API

```
PMDresult PMDClearPositionError (PMDAxisInterface axis_intf)
```

VB-Motion API

```
MagellanAxis.ClearPositionError()
```

see

GetPositionError (p. 51), **MultiUpdate** (p. 65), **Set/GetPositionErrorLimit** (p. 173), **Update** (p. 215)

ExecutionControl 35h

Syntax

ExecutionControl *axis option value*

Motor Types

Brush DC	Brushless DC	Microstepping

Arguments

Name	Instance	Encoding	
axis	Axis1	0	
condition	delay	0	
	— (Reserved)	1-7	
	event status	8	
	activity status	9	
	signal status	10	
	drive status	11	
	— (Reserved)	12-255	
timeScale	multiply by 2	0	
	multiply by 256 (2^8)	1	
	multiply by 32768 (2^{15})	2	
	multiply by 4194034 (2^{22})	3	
timeValue	unsigned 6 bit	0-63	51.2 µs
value	unsigned 32bit	see below	

Packet Structure

ExecutionControl

	0	*axis*	1h
	15 12	11 8	7 0

write	*timeScale*	*timeValue*	*condition*
	15 14	13 8	7 0

write	*value* (high-order part)
	15 0

write	*value* (low-order part)
	15 0

Description

ExecutionControl is used to delay execution during NVRAM initialization, usually so that some hardware external to the Magellan IC may become ready. In all cases the timeout value is measured in units of the 51.2 µs commutation time.

If the condition is ***delay***, then a pure delay for a fixed time. In this case the ***value*** argument is an unsigned count of commutation cycles to wait. The exit status in this case is always zero, or no error. In this case the ***timeScale*** and ***timeValue*** arguments must both be zero.

If the condition is ***event status, activity status, signal status***, or ***drive status***, then execution will be delayed until either a specified condition becomes true for the specified register, or a timeout expires. The condition is defined by the supplied ***value*** – the high order part is a selection mask for the register value, and the low order part is a sense mask. The wait will end successfully when the register value, logically ANDed with the selection mask is equal to the sense mask.

For example, to wait for phase initialization to complete, the condition should be *activity status*, because bit 0 of the activity status register is defined as *Phasing Initialized*. The selection mask in this case would be 0001h, and the sense mask also 00001h.

Description (cont.)

As another example, to wait until the ~*Enable* signal is low (active), one should wait until bit 13 of the Signal Status register is clear. The condition should be *signal status*, the selection mask 2000h, and the sense mask 0000h.

When waiting conditionally on a register value, the ***timeScale*** and ***timeValue*** arguments specify a timeout period in commutation cycles. If the timeout period elapses before the condition becomes true then the command will exit with an error status of *Wait Timed Out*, NVRAM command processing will stop, and motor output will be disabled. The *Instruction Error* bit of the event status register will be set, and the **GetInstructionError** command may be used to read the error status.

A ***timeValue*** of zero means "wait forever"; a timeout will never occur.

timeValue is multiplied by ***timeScale***, to give a wider range. The minimum timeout is 2 commutation cycles, the maximum value is 63 x 2^{22} = 264,241,152, or approximately 3.7 hours.

Magellan does not normally accept host input on the serial, CAN, or SPI channels until NVRAM initialization has completed, however if an **ExecutionControl** wait is started then the host interfaces will be initialized and host commands accepted. In this situation it is possible for NVRAM commands to be executed after outside host commands, changing Magellan state. In all cases only one command, from any source, is executed at a time.

The script interface combines the condition, ***timeValue*** and ***timeScale*** arguments into a single option argument as shown below. For example, if the condition is event status (8), and the desired timeout value is 768 commutation cycles, then the ***timeScale*** x256 (1) and the ***timeValue*** is 3. The option argument should be 8 + 256*3 + 16384*1 = 17160

Restrictions

Valid only when executed from NVRAM.

Errors

Invalid Parameter: Condition is not a supported value, tvalue or tscale nonzero for pure delay.

Initialization Only: Command was sent using serial, CAN, or SPI host channel.

Wait Timed Out: Timeout elapsed before condition became true.

C-Motion API

```
PMDresult PMDExecutionControl(PMDAxisInterface axis_intf, PMDuint8
                              condition, PMDuint8 timeScale, PMDuint16
                              timeValue, PMDint32 value);
```

Script API

```
ExecutionControl option value
where option = condition + 256*timeValue + 16384*timeScale
```

C# API

```
PMDAxis.ExecutionControl(Int16 condition, Int16 timeValue,
                         Int16 timeScale, Int32 value);
```

Visual Basic API

```
PMDAxis.ExecutionControl(ByVal condition As Int16, ByVal timeValue
                         As Int16,ByVal timeScale As Int16,ByVal
                         value as Int32)
```

see

NVRAM (p. 68), **GetEventStatus** (p. 42), **GetActivityStatus** (p. 25), **GetDriveStatus** (p. 38), **GetSignalStatus** (p. 55), **GetInstructionError** (p. 46)

GetActiveMotorCommand 3Ah

2

Syntax

GetActiveMotorCommand *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	▨

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	Range	Scaling	Units
command	signed 16 bits	-2^{15} *to* $2^{15}-1$	$100/2^{15}$	% output

Packet Structure

GetActiveMotorCommand

0	*axis*	**3A**h
15 — 12	11 — 8	7 — 0

Data

read	*command*	
	15	0

Description

GetActiveMotorCommand returns the value of the motor output command for the specified ***axis***. This is the input to the commutation or FOC current control. Its source depends on the motor type, as well as the operating mode of the ***axis***.

For brushless DC and DC brush motors: If position loop is enabled, it is the output of the position servo filter. If trajectory generator is enabled without the position loop, it is the output of the trajectory generator. If both trajectory generator and position loop are disabled, it is the contents of the motor output command register.

For microstepping motors: It is the contents of the motor output command register, subject to holding current reduction.

Atlas

Restrictions

C-Motion API

```
PMDresult PMDGetActiveMotorCommand (PMDAxisInterface axis_intf,
                                    PMDint16* command)
```

VB-Motion API

```
Dim command as Short
command = MagellanAxis.ActiveMotorCommand
```

see

Set/GetMotorCommand (p. 151), **Set/GetOperatingMode** (p. 156), **GetActiveOperatingMode** (p. 24)

GetActiveOperatingMode 57h

Syntax GetActiveOperatingMode *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned Data

	Type	
mode	unsigned 16 bits	bit field

Packet Structure

GetActiveOperatingMode

0	*axis*	**57**h
15 – 12	11 – 8	7 – 0

First data word

read	*mode*
	15 – 0

Description **GetActiveOperatingMode** gets the actual operating mode that the *axis* is currently in. This may or may not be the same as the static operating mode, as safety responses or programmable conditions may change the **Active Operating Mode**. When this occurs, the **Active Operating Mode** can be changed to the programmed static operating mode using the **RestoreOperatingMode** command. The bit definitions of the operating mode are given below.

Name	Bit	Description
Axis Enabled	0	0: No *axis* processing, axis outputs in Reset state. 1: *axis* active.
Motor Output Enabled	1	0: *axis* motor outputs disabled. 1: *axis* motor outputs enabled.
Current Control Enabled	2	0: *axis* current control bypassed. 1: *axis* current control active.
—	3	Reserved
Position Loop Enabled	4	0: *axis* position loop bypassed. 1: *axis* position loop active.
Trajectory Enabled	5	0: trajectory generator disabled. 1: trajectory generator enabled.
—	6–15	Reserved

When the axis is disabled, no processing will be done on the axis, and the axis outputs will be at their reset states. When the axis motor output is disabled, the axis will function normally, but its motor outputs will be in their disabled state. When a loop is disabled (position or current loop), it operates by passing its input directly to its output, and clearing all internal state variables (such as integrator sums, etc.). When the trajectory generator is disabled, it operates by commanding zero (0) velocity.

Atlas Note that the current control bit is meaningful whenever an axis is connected to an Atlas amplifier.

Restrictions The possible modes of an axis are product specific, and in some cases axis specific. See the product user guide for a description of what modes are supported on each axis.

C-Motion API

```
PMDresult PMDGetActiveOperatingMode(PMDAxisInterface axis_intf,
                                    PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
mode = MagellanAxis.ActiveOperatingMode
```

see **GetOperatingMode** (p. 156), **RestoreOperatingMode** (p. 82), **Set/GetEventAction** (p. 135), **Set/GetBreakpoint** (p. 94)

GetActivityStatus A6h

Syntax

GetActivityStatus *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned Data

	Type	
status	unsigned 16 bits	see below

Packet Structure

GetActivityStatus

15–12	11–8	7–0
0	*axis*	**A6**h

Data

	15–13	12	11	10	9	8	7	6	5–3	2	1	0
read								0				

Description

GetActivityStatus reads the 16-bit Activity Status register for the specified *axis*. Each of the bits in this register continuously indicate the state of the motion control IC without any action on the part of the host. There is no direct way to set or clear the state of these bits, since they are controlled by the motion control IC.

The following table shows the encoding of the data returned by this command.

Name	Bit(s)	Description
Phasing Initialized	0	Set to 1 if phasing is initialized (brushless DC axes only).
At Maximum Velocity	1	Set to 1 when the trajectory is at maximum velocity. This bit is determined by the trajectory generator, not the actual encoder velocity.
Tracking	2	Set to 1 when the axis is within the tracking window.
Current Profile Mode	3–5	Contains trajectory mode encoded as follows: bit 5 bit 4 bit 3 Profile Mode 0 0 0 Trapezoidal 0 0 1 Velocity Contouring 0 1 0 S-curve 0 1 1 Electronic Gear
—	6	Reserved; not used; may be 0 or 1.
Axis Settled	7	Set to 1 when the axis is settled.
Position Loop Enabled	8	Set to 1 when position loop or trajectory is enabled.
Position Capture	9	Set to 1 when a value has been captured by the high speed position capture hardware but has not yet been read.

Description (cont.)

Name	Bit(s)	Description
In-motion	10	Set to 1 when the trajectory generator is executing a profile.
In Positive Limit	11	Set to 1 when the positive limit switch is active.
In Negative Limit	12	Set to 1 when the negative limit switch is active.
Profile Segment	13–15	When the profile mode is S-curve, it contains the profile segment number 1–7 while profile is in motion, and contains a value of 0 when the profile is at rest. This field is undefined when using the Trapezoidal and Velocity Contouring profile modes.

Restrictions

C-Motion API

```
PMDresult PMDGetActivityStatus(PMDAxisInterface axis_intf,
                               PMDuint16* status)
```

VB-Motion API

```
Dim status as Short
status = MagellanAxis.ActivityStatus
```

see

GetEventStatus (p. 42), **GetSignalStatus** (p. 55), **GetDriveStatus** (p. 38)

GetActualVelocity ADh

Syntax

GetActualVelocity *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned Data

	Type	Range	Scaling	Units
actual velocity	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	counts/cycle

Packet Structure

GetActualVelocity

0	*axis*	**AD**h
15 — 12	11 — 8	7 — 0

First data word

read	*actual velocity* (high-order part)
	31 — 16

Second data word

read	*actual velocity* (low-order part)
	15 — 0

Description

GetActualVelocity reads the value of the *actual velocity* for the specified *axis*. The *actual velocity* is derived by subtracting the actual position during the previous chip cycle from the actual position for this chip cycle. The result of this subtraction will always be integer because position is always integer. As a result the value returned by **GetActualVelocity** will always be a multiple of 65,536 since this represents a value of one in the 16.16 number format. The low word is always zero (0). This value is the result of the last encoder input, so it will be accurate to within one cycle.

Scaling example: If a value of 1,703,936 is retrieved by the **GetActualVelocity** command (high word: 01Ah, low word: 0h), this corresponds to a velocity of 1,703,936/65,536 or 26 counts/cycle.

Restrictions

C-Motion API

```
PMDresult PMDGetActualVelocity(PMDAxisInterface axis_intf,
                               PMDint32* velocity)
```

VBI-Motion API

```
Dim velocity as Long
velocity = MagellanAxis.ActualVelocity
```

see

GetCommandedVelocity (p. 33), **GetActualPosition** (p. 85)

GetBusVoltage 40h

Syntax **GetBusVoltage** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned Data

	Type	Range	Scaling
voltage	unsigned 16 bits	0 *to* 2^{16}–1	product specific

Packet Structure

GetBusVoltage

0	*axis*	**40**h
15 12	11 8	7 0

First data word

read	*voltage*
	15 0

Description **GetBusVoltage** gets the most recent bus voltage reading from the ***axis***. Consult specific product documentation for scaling information.

Atlas This command is relayed to any connected Atlas amplifier.

Restrictions **GetBusVoltage** is only available in products equipped with bus voltage sensors.

C-Motion API

```
PMDresult PMDGetBusVoltage(PMDAxisInterface axis_intf,
                           PMDuint16* voltage)
```

VB-Motion API

```
Dim voltage as Short
voltage = MagellanAxis.BusVoltage
```

see **Get/SetDriveFaultParameter** (p. 126)

GetCaptureValue 36h

Syntax

GetCaptureValue *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	Range	Scaling	Units
position	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts microsteps

Packet Structure

GetCaptureValue

0	*axis*	**36**h
15 … 12	11 … 8	7 … 0

First data word

read	*position* (high-order part)
	31 … 16

Second data word

read	*position* (low-order part)
	15 … 0

Description

GetCaptureValue returns the contents of the position capture register for the specified ***axis***. This command also resets bit 9 of the Activity Status register, thus allowing another capture to occur.

If actual position units is set to steps, the returned position will be in units of steps.

Restrictions

C-Motion API

```
PMDresult PMDGetCaptureValue(PMDAxisInterface axis_intf,
                             PMDint32* position)
```

VBI-Motion API

```
Dim position as Long
position = MagellanAxis.CaptureValue
```

see

Set/GetCaptureSource (p. 108), **Set/GetActualPositionUnits** (p. 87), **GetActivityStatus** (p. 25)

GetChecksum F8h

Syntax **GetChecksum**

Motor Types

DC Brush	**Brushless DC**	**Microstepping**	**Pulse & Direction**

Arguments None

Returned data

Name	Type
checksum	unsigned 32 bits

Packet Structure

GetChecksum

	0	**F8**h
	15 … 8	7 … 0

First data word

read	*checksum* (high-order part)
	31 … 16

Second data word

read	*checksum* (low-order part)
	15 … 0

Description **GetChecksum** reads the chips internal 32-bit *checksum* value. The return value is dependent on the silicon revision number of the motion control IC.

Restrictions

C-Motion API

```
PMDresult PMDGetChecksum(PMDAxisInterface axis_intf,
                         PMDuint32* checksum)
```

VB-Motion API

```
Dim checksum as Long
checksum = MagellanObject.Checksum
```

see

GetCommandedAcceleration A7h

Syntax

GetCommandedAcceleration *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	Range	Scaling	Units
acceleration	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	counts/cycle^2 microsteps/cycle^2

Packet Structure

GetCommandedAcceleration

15–12	11–8	7–0
0	*axis*	**A7**h

First data word

	31–16
read	*acceleration* (high-order part)

Second data word

	15–0
read	*acceleration* (low-order part)

Description

GetCommandedAcceleration returns the commanded *acceleration* value for the specified *axis*. Commanded acceleration is the instantaneous acceleration value output by the trajectory generator.

Scaling example: If a value of 114,688 is retrieved using this command then this corresponds to 114,688/65,536 = 1.750 counts/cycle^2 acceleration value.

Restrictions

C-Motion API

```
PMDresult PMDGetCommandedAcceleration(PMDAxisInterface axis_intf,
                                      PMDint32* acceleration)
```

VB-Motion API

```
Dim acceleration as Long
acceleration = MagellanAxis.CommandedAcceleration
```

see

GetCommandedPosition (p. 32), **GetCommandedVelocity** (p. 33)

GetCommandedPosition 1Dh

Syntax

GetCommandedPosition *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	Range	Scaling	Units
position	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts microsteps

Packet Structure

GetCommandedPosition

15–12	11–8	7–0
0	*axis*	**1D**h

First data word

	31–16
read	*position* (high-order part)

Second data word

	15–0
read	*position* (low-order part)

Description

GetCommandedPosition returns the commanded *position* for the specified *axis*. Commanded position is the instantaneous position value output by the trajectory generator.

This command functions in all profile modes.

Restrictions

C-Motion API

```
PMDresult PMDGetCommandedPosition(PMDAxisInterface axis_intf,
                                  PMDint32* position)
```

VB-Motion API

```
Dim position as Long
position = MagellanAxis.CommandedPosition
```

see

GetCommandedAcceleration (p. 31), **GetCommandedVelocity** (p. 33)

GetCommandedVelocity 1Eh

Syntax

GetCommandedVelocity *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	Range	Scaling	Units
velocity	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	counts/cycle microsteps/cycle

Packet Structure

GetCommandedVelocity

15–12	11–8	7–0
0	*axis*	**1E**h

First data word

	31–16
read	*velocity* (high-order part)

Second data word

	15–0
read	*velocity* (low-order part)

Description

GetCommandedVelocity returns the commanded *velocity* value for the specified *axis*. Commanded velocity is the instantaneous velocity value output by the trajectory generator.

Scaling example: If a value of –1,234,567 is retrieved using this command (FFEDh in high word, 2979h in low word) then this corresponds to –1,234,567/65,536 = –18.8380 counts/cycle velocity value.

Restrictions

C-Motion API

```
PMDresult PMDGetCommandedVelocity(PMDAxisInterface axis_intf,
                                  PMDint32* velocity)
```

VB-Motion API

```
Dim velocity as Long
velocity = MagellanAxis.CommandedVelocity
```

see

GetCommandedAcceleration (p. 31), **GetCommandedPosition** (p. 32)

GetCurrentLoopValue 71h

Syntax

GetCurrentLoopValue *axis phase_node*

Motor Types

DC Brush	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
phase	*Phase A*	0
	Phase B	1
node	*Reference*	0
	Actual Current	1
	Error	2
	Integrator Sum	3
	— (Reserved)	4
	Integrator Contribution	5
	Output	6
	I^2t Energy	10

Returned data

	Type	Range/Scaling
value	signed 32 bits	see below

Packet Structure

GetCurrentLoopValue

0	*axis*	**71**h
15 12	11 8	7 0

First data word

write	0	*phase*	*node*
	15 12	11 8	7 0

Second data word

read	*value* (high-order part)
	31 16

Third data word

read	*value* (low-order part)
	15 0

Description

GetCurrentLoopValue is used to read the value of a *node* in one of the digital current loops. See the product user guide for more information on the location of each *node* in the current loop processing. Though the data returned is signed 32 bits regardless of the *node*, the range and format vary depending on the *node*, as follows:

Node	Range	Scaling	Units
Reference	-2^{15} to $2^{15}-1$	$100/2^{14}$	% max current
Actual Current	-2^{15} to $2^{15}-1$	$100/2^{14}$	% max current
Error	-2^{15} to $2^{15}-1$	$100/2^{14}$	% max current
Integrator Sum	-2^{31} to $2^{31}-1$	$100/2^{14}$	(% max current)* current loop cycles
Integrator Contribution	-2^{31} to $2^{31}-1$	$100/2^{14}$	% max current
Output	-2^{15} to $2^{15}-1$	$100/2^{14}$	% max current
I^2t Energy	-2^{31} to $2^{31}-1$	$100/2^{30}$	% max energy

Description (cont.)

All of the ***nodes*** have units of % maximum current, and most have scaling of $100/2^{14}$. That is, a value of 2^{14} corresponds to 100% maximum current. The range is extended to allow for overshoot in excess of maximum peak current, and thus values can be more than 100% of the maximum output current.

The Integrator Sum is a signed 32-bit number, with scaling of $100/2^{14}$. That is, a current error of 100% maximum, present for 16 current loop cycles, will result in an integrator sum of $16*(100\%)*2^{14}/100 = 2^{18}$. Current loop cyles are not the same as position loop servo cycles. The current loop runs at 20 kHz, regardless of the servo cycle time.

Atlas

This command is relayed to any connected Atlas amplifier.

Restrictions

This command is only supported in products that include digital current control, and when the current control mode is Phase A /B.

C-Motion API

```
PMDresult PMDGetCurrentLoopValue(PMDAxisInterface axis_intf,
                                 PMDuint8 phase,
                                 PMDuint8 node,
                                 PMDint32* value)
```

VB-Motion API

```
MagellanAxis.CurrentLoopValue ([in] phase,
                               [in] node,
                               [out] value
```

see

Set/GetCurrentLoop (p. 120), **Set/GetCurrentControlMode** (p. 115)
Set/Get Current Foldback ((p. 117)

GetDriveFaultStatus 6Dh

Syntax **GetDriveFaultStatus** *axis*

Motor Types

DC Brush	**Brushless DC**	**Microstepping**	**Pulse & Direction**

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned Data

	Type	
status	unsigned 16 bits	see below

Packet Structure

GetDriveFaultStatus

15–12	11–8	7–0
0	*axis*	**6D**h

First data word

	15–7	6–0
read	0	*Status*

Description

GetDriveFaultStatus reads the Drive Fault Status register, which contains a bitmap showing all hard faults that have occurred since the Drive Fault Status register was last cleared. In the ION products, this register is kept in non-volatile memory, so that a record of hard faults is retained even through power cycles, which must be done upon any hard fault event.

The table below shows the bit definitions of the Drive Fault Status register.

Name	**Bit**
Overcurrent Fault	0
Ground Fault	1
External Logic Fault	2
Atlas Operating Mode Mismatch	3
Internal Logic Fault	4
Overvoltage Fault	5
Undervoltage Fault	6
Atlas Disabled by /Enable Signal	7
Current Foldback	8
Overtemperature Fault (non-Atlas)	9
Atlas Detected SPI Checksum Error	10
Atlas Watchdog Timeout	11
— (Reserved)	12
Disabled by ~PWMOutputDisable signal	13
Magellan Detected SPI Checksum Error	14
Atlas Motor Type Mismatch	15

ION products enforce a "hard fault" for events 0, 1, 2, and 4, meaning that if one of these occur the unit will shut down, and power must be cycled before it will accept any communication. Upon power-up, **GetDriveFaultStatus** should be used to check which, if any, hard fault may have caused the previous power cycle. After querying the Drive Fault Status register, it should be cleared using **ClearDriveFaultStatus**. If this is not done, the bits will be retined in non-volatile memory, which will make it difficult to detect the cause of any subsequent hard faults.

For all other events, and for non-ION products, there is no non-volatile storage of the Drive Fault Status register, and a power cycle is not required to recover from a fault.

Events 5 and 6 will not cause the system to shut down. Instead, they will cause the system to change to the disabled state, and will cause the Drive Fault bit in **GetEventStatus** to be set. Normally, the Drive Fault Status register does not need to be monitored. In the case of a Drive Fault event, however, the Drive Fault Status register can be used to determine the particular fault that occurred. The Overvoltage Fault and Undervoltage Fault bits are cleared upon power-up.

Event 13 indicates that motor output was disabled by the ~*PWMOutputDisable* signal, in which case the DriveException bit will be set in the Event Status register. Not all products support this signal, check your product user guide for more information.

Event 8 indicates that the current foldback limit was exceeded. If current control is not enabled this will result in output being disabled. If current control is enabled then the action taken may be specified using **SetEventAction**. ION does not use this event bit.

Atlas

This command is relayed to any connected Atlas amplifier, and the result combined with bits 14 and 15 from internal Magellan state to form the result.

The Atlas amplifier does not implement hard faults; events 3, 7, 9 and 11 will unconditionally cause Atlas to disable output, and raise a Drive Exception event. The Drive Exception event is transmitted to the Magellan using the Atlas SPI status word, which is received with every torque command sent, and will cause the Magellan axis to disable output as well. Event 8 may similarly disable output depending on the current foldback event action.

Events 10 and 14 are not handled by Magellan, but indicate a problem with SPI communication, which may seriously affect Atlas amplifier operation. Event status bit 7 (Instruction Error), will also be set whenever one of the SPI checksum fault bits is set.

Event 15 indicates that the Magellan motor type and an attached Atlas amplifier motor type are not compatible. This bit may be cleared only by using **SetMotorType**.

Restrictions

This command is not available in products without drive amplifier support.

C-Motion API

```
PMDresult PMDGetDriveFaultStatus(PMDAxisInterface axis_intf,
                                 PMDuint16* status)
```

VB-Motion API

```
Dim status as Short
status = MagellanAxis.DriveFaultStatus
```

see

ClearDriveFaultStatus (p. 18)
GetEventStatus (p. 42)
SetMotorType (p. 154)
SetEventAction (p. 135)

GetDriveStatus 0Eh

Syntax

GetDriveStatus *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	
status	unsigned 16 bits	see below

Packet Structure

GetDriveStatus

0	*axis*	**0E**h
15 12	11 8	7 0

First Data Word

	Status
read	15 0

Description

GetDriveStatus reads the Drive Status register for the specified *axis*. All of the bits in this status word are set and cleared by the motion control IC. They are not settable or clearable by the host. The bits represent states or conditions in the motion control IC that are of a transient nature.

Name	Bit(s)	Description
Calibrated	0	Set to 0 when calibration is started, set to 1 when calibration is complete.
In Foldback	1	Set to 1 when the unit is in the current foldback state–the output current is limited by the foldback limit.
Overtemperature	2	Set to 1 when the overtemperature condition is present.
Shunt active	3	The bus voltage limiting shunt PWM is active.
In Holding	4	Set to 1 when the unit is in the holding current state–the output current is limited by the holding current limit.
Overvoltage	5	Set to 1 when the overvoltage condition is present.
Undervoltage	6	Set to 1 when the undervoltage condition is present.
Atlas Disabled	7	The attached Atlas amplifier is disabled by an inactive */Enable* signal.
—	8–11	Reserved; not used; may be 0 or 1.
Output Clipped	12	Drive output is limited because it has reached 100%, or the Drive PWM limit, or the current loop integrator limit.
—	13, 14	Reserved; not used; may be 0 or 1.
Atlas not connected	15	The output mode is Atlas, but SPI communication has not been established.

Atlas

This command does not require any additional Atlas communication, all of the required data is transmitted in the Atlas SPI Status Word received when sending torque commands.

Restrictions

The bits available in this register depend upon the products. See the product user guide.

GetDriveStatus (cont.) 0Eh

C-Motion API

```
PMDresult PMDGetDriveStatus(PMDAxisInterface axis_intf,
                            PMDuint16* status)
```

VB-Motion API

```
Dim status as Short
status = MagellanAxis.DriveStatus
```

see

GetDriveValue 70h

Syntax

GetDriveValue *axis node*

Motor Types

DC Brush	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
node	*Bus Voltage*	0
	Temperature	1
	Bus Current Supply	2
	Bus Current Return	3

Returned data

	Type	Range/Scaling
value	signed or unsigned 16 bits	see below

Packet Structure

GetDriveValue

0	*axis*	**70**h
15 12	11 8	7 0

First Data Word

	node
write	node
	15 0

Second Data Word

	value
read	value
	15 0

Description

GetDriveValue is used to read values associated with drive output or state, and enumerated by node. Some of the functionality provided by **GetDriveValue** is duplicated by **GetTemperature** and **GetBusVoltage**, however **GetDriveValue** is preferred for future use.

The following nodes are supported:

Bus Voltage is the most recent bus voltage reading from the axis, returned as an unsigned 16 bit value. The scaling depends on the product and on the external bus voltage sensing circuit.

Temperature is the most recent temperature reading from temperature sensor monitoring axis, returned as a signed 16 bit value. The scaling depends on the product and on the external temperature sensing circuit. For MC58113, if the temperature limit set by **SetDriveFaultParameter** is negative then the sense of the temperature is inverted by subtracting the measured value from 32768.

Bus Current Supply is the most recent reading from the bus current supply sensor, returned as an unsigned 16 bit value. Scaling depends on the product and the external current sensing circuit.

Bus Current Return is the most recent current return reading computed from all leg current readings and PWM duty cycles, returned as a signed 16 bit number. The scaling depends on the product and on the external leg current sensing circuit; it is the same as the leg current scaling.

Restrictions

GetDriveValue is currently supported only by MC58113 series motion control ICs.

C-Motion API

```
PMDresult PMDGetDriveValue(PMDAxisInterface axis_intf,
                 PMDuint8 node,
                 PMDuint16 * value);
```

VB-Motion API

```
MagellanAxis.DriveValue([in] node,
                 [out] value)
```

see **GetTemperature** (p. 57), **GetBusVoltage** (p. 28), **SetDriveFaultParameter** (p. 126)

GetEventStatus 31h

Syntax **GetEventStatus** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	
status	unsigned 16 bits	see below

Packet Structure

GetEventStatus

15 – 12	11 – 8	7 – 0
0	*axis*	**31**h

Data

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
read	0		0													

Description **GetEventStatus** reads the Event Status register for the specified *axis*. All of the bits in this status word are set by the motion control IC and cleared by the host. To clear these bits, use the **ResetEventStatus** command. The following table shows the encoding of the data returned by this command.

Name	Bit(s)	Description
Motion Complete	0	Set to 1 when motion has completed. **SetMotionCompleteMode** determines if this bit is based on the trajectory generator position or the encoder position.
Wrap-around	1	Set to 1 when the actual (encoder) position has wrapped from maximum allowed position to minimum, or vice versa.
Breakpoint 1	2	Set to 1 when breakpoint 1 has been triggered.
Capture Received	3	Set to 1 when a position capture has occurred.
Motion Error	4	Set to 1 when a motion error has occurred.
Positive Limit	5	Set to 1 when the axis has entered a positive limit switch.
Negative Limit	6	Set to 1 when the axis has entered a negative limit switch.
Instruction Error	7	Set to 1 when an instruction error has occurred. This bit is also set when an Atlas checksum error is detected. In that case either the Magellan Detected SPI Checksum or the Atlas Detected SPI Checksum error bits will be set in the Drive Fault status register.
Disable	8	Set to 1 when "disable" due to user /Enable line has occurred.
Overtemperature Fault	9	Set to 1 when overtemperature condition has occurred.
Drive Exception	10	An drive event occurred causing output to be disabled. This bit is used on ION products to indicate a bus voltage fault, and with an attached Atlas amplifier to indicate any disabling drive event.
Commutation error	11	Set to 1 when a commutation error has occurred.
Current Foldback	12	Set to 1 when current foldback has occurred.
—	13	Reserved; not used; may be 0 or 1.
Breakpoint 2	14	Set to 1 when breakpoint 2 has been triggered.
—	15	Reserved; not used; may be 0 or 1.

Atlas

This command does not require any additional Atlas communication, all of the required data is transmitted in the Atlas SPI Status Word received when sending torque commands.

In the case of Drive Exception or Instruction Error, more precise information may be obtained by using the GetDriveFaultStatus command. It should be noted that the Overtemperature event bit is not used for Atlas axes.

Restrictions

Bits 8, 9, 10, and 12 are not implemented in products that do not include drive amplifier support. In this case, they are reserved—may be 0 or 1.

C-Motion API

```
PMDresult PMDGetEventStatus(PMDAxisInterface axis_intf,
                            PMDuint16* status)
```

VB-Motion API

```
Dim status as Short
status = MagellanAxis.EventStatus
```

see

GetActivityStatus (p. 25), **GetSignalStatus** (p. 55), **GetDriveStatus** (p. 38),
GetDriveFaultStatus (p. 36)

GetFOCValue 5Ah

Syntax

GetFOCValue *axis loop_node*

Motor Types

▨	Brushless DC	Microstepping	▨

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
loop	*Direct (D)*	0
	Quadrature (Q)	1
node	*Reference (D,Q)*	0
	Feedback (D,Q)	1
	Error (D,Q)	2
	Integrator Sum (D,Q)	3
	— (Reserved)	4
	Integrator Contribution (D,Q)	5
	Output (D,Q)	6
	FOC Output (Alpha,Beta)	7
	Actual Current (A,B)	8
	I^2t Energy	10

Returned data

	Type	Range/Scaling
value	signed 32 bits	see below

Packet Structure

GetFOCValue

	15 – 12	11 – 8	7 – 0
	0	*axis*	**5A**h

First data word

	15 – 12	11 – 8	7 – 0
write	0	*loop*	*node*

Second data word

	31 – 16
read	*value* (high-order part)

Third data word

	15 – 0
read	*value* (low-order part)

Description

GetFOCValue is used to read the value of a ***node*** of the FOC current control. See the product user guide for more information on the location of each ***node*** in the FOC current control algorithm.

Description (cont.)

Though the data returned is signed 32 bits regardless of the *node*, the range and format vary depending on the *node*, as follows:

Node	Range	Scaling	Units
Reference (D,Q)	-2^{15} to $2^{15}-1$	$100/2^{14}$	% max current
Feedback (D,Q)	-2^{18} to $2^{18}-1$	$100/2^{14}$	% max current
Error (D,Q)	-2^{15} to $2^{15}-1$	$100/2^{14}$	% max current
Integrator Sum (D,Q)	-2^{31} to $2^{31}-1$	$100/2^{14}$	(% max current)* current loop cycles
Integrator Contribution (D,Q)	-2^{31} to $2^{31}-1$	$100/2^{14}$	% max current
Output (D,Q)	-2^{15} to $2^{15}-1$	$100/2^{14}$	% PWM
FOC Output (Alpha,Beta)	-2^{15} to $2^{15}-1$	$100/2^{14}$	% PWM
Actual Current (A,B)	-2^{15} to $2^{15}-1$	$100/2^{14}$	% max current
I^2t Energy	-2^{31} to $2^{31}-1$	$100/2^{30}$	% max energy

Most of the *nodes* have units of % maximum current, and most have a scaling of $100/2^{14}$. That is, a value of 2^{14} corresponds to 100% maximum current. The range is extended to allow for overshoot in excess of maximum peak current, and thus values can be more than 100% of the maximum output current.

The *Integrator Sum* is a signed 32-bit number, with scaling of $100/2^{14}$. That is, a current of 100% maximum, present for 16 current loop cycles, will result in an integrator sum of $16*(100\%)*2^{14}/100 = 2^{18}$. Current loop cycles are not the same as position loop servo cycles. The current loop runs at 20 kHz, regardless of the servo cycle time.

Atlas

This command is relayed to an attached Atlas amplifier.

Restrictions

This command is only supported in products that include digital current control, and when the current control mode is set to FOC.

C-Motion API

```
PMDresult PMDGetFOCValue (PMDAxisInterface axis_intf,
                          PMDuint8 loop,
                          PMDuint8 node,
                          PMDint32* value)
```

VB-Motion API

```
MagellanAxis.FOCValue ( [in] loop,
                        [in] node,
                        [out] value )
```

see

Set/GetFOC (p. 141), **Set/GetCurrentControlMode** (p. 115)
Set/Get Current Foldback ((p. 117)

GetInstructionError A5h

Syntax

GetInstructionError

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

None

Returned data

	Type	Range
error	unsigned 16 bits	0 *to* 11h

Packet Structure

GetInstructionError

0	**A5**h
15 12 11 8	7 0

Data

read	*second error*	*first error*
	15 8	7 0

Description

GetInstructionError returns the code for the first instruction error since the last read operation, and then resets the error to zero (0). Generally, this command is issued only after the instruction error bit in the Event Status register indicates there was an instruction error. It also resets the Instruction error bit in the I/O status read word to zero (0).

The Atlas and MC58113 series products will return both the first and second errors after the last read operation. This is especially helpful in debugging initialization commands executed at startup from non-volatile RAM, since the first error is always a Processor reset (1). For other Magellan products the second error field will always be zero.

Description (cont'd)

The error codes are encoded as defined below:

Error Code	Encoding
No error	0
Processor reset	1
Invalid instruction	2
Invalid axis	3
Invalid parameter	4
Trace running	5
— (Reserved)	6
Block out of bounds	7
Trace buffer zero (0)	8
Bad serial checksum	9
— (Reserved)	10
Invalid negative value	11
Invalid parameter change	12
Invalid move after event-triggered stop	13
Invalid move into limit	14
Invalid Operating Mode restore after event-triggered change	16
Invalid Operating Mode for command	17
Invalid register state for command	18
ION/CME hard fault	19
Command invalid without Atlas amplifier	20
Incorrect Atlas command checksum	21
Invalid Atlas command protocol	22
Invalid Atlas command timing	23
Invalid Atlas torque command detected	24
— (Reserved)	25
Atlas command invalid in flash mode	26
— (Reserved)	27
Atlas command valid only for initialization	28
Wrong command data count	28
Attempted move with motion error event signaled	30
Wait timed out	31
NVRAM initialization busy	32
Invalid clock signal	33
NVRAM initialization skipped	34
Invalid interface for command	35
Encoder error	36
Value representation error	37
-- (Reserved)	38
NVRAM format error	39

Atlas

This command does not require any additional Atlas communication. In case a command error is signaled by an Atlas amplifier during the processing of a Magellan command the Magellan instruction error register will be set to the error code returned by Atlas. The error code is maintained separately by the Atlas amplifier and may be cleared by reading directly from Atlas; it is not reset by reading the Magellan instruction error code.

Restrictions

C-Motion API

```
PMDresult PMDGetInstructionError  (PMDAxisInterface axis_intf,
                                   PMDuint16* error)
```

VB-Motion API

```
Dim error as Short
error = MagellanObject.InstructionError
```

see **GetEventStatus** (p. 42), **ResetEventStatus** (p. 80)

GetInterruptAxis E1h

Syntax **GetInterruptAxis**

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments None

Returned data

Name	Instance	Encoding
mask	*None*	0
	Axis1 Mask	1
	Axis2 Mask	2
	Axis3 Mask	4
	Axis4 Mask	8

Packet Structure

GetInterruptAxis

0	**E1**h
15 … 8	7 … 0

Data

	0	*mask*
read	15 … 4	3 … 0

Description **GetInterruptAxis** returns a field that identifies all axes with pending interrupts. Axis numbers are assigned to the low-order four bits of the returned word, with bits corresponding to interrupting axes set to 1. If there are no pending interrupts, the returned word is zero (0). If any axis has a pending interrupt, the /HostInterrupt signal will be in an active state.

Restrictions This command is only useful for products with /HostInterrupt pin. When using CAN events for interrupt event notification, the interrupting axis is sent as part of the CAN event.

C-Motion API

```
PMDresult PMDGetInterruptAxis(PMDAxisInterface axis_intf,
                              PMDuint16* mask)
```

VB-Motion API

```
Dim mask as Short
mask = MagellanObject.InterruptAxis
```

see **ClearInterrupt** (p. 19), **Set/GetInterruptMask** (p. 146)

GetPhaseCommand EAh

Syntax

GetPhaseCommand *axis phase*

Motor Types

▨	**Brushless DC**	**Microstepping**	▨

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
phase	*Phase A*	0
	Phase B	1
	Phase C	2

Returned data

	Type	Range	Scaling	Units
command	signed 16 bits	-2^{15} *to* $2^{15}-1$	$100/2^{15}$	% output

Packet Structure

GetPhaseCommand

0	*axis*	**EA**h
15 12	11 8	7 0

First data word

write	0	*phase*
	15 2	1 0

Second data word

read	*command*
	15 0

Description

GetPhaseCommand returns the value of the commutated phase command for phase A, B, or C of the specified *axis*. These are the phase command values directly output to the current loop or motor after commutation.

Scaling example: If a value of –4,489 is retrieved (EE77h) for a given axis and phase, then this corresponds to –4,489*100/32,767 = –13.7% of full-scale output.

Restrictions

Phase C is only valid when the motor type has been set for a 3-phase commutation.

This command has no meaning when current control mode is set to FOC whether or not the current loops are enabled.

When the current control mode is set to *Phase A /B* current loops, the values are the inputs to the current loops. When current loops are disabled, the value is the motor output command.

C-Motion API

```
PMDresult PMDGetPhaseCommand(PMDAxisInterface axis_intf,
                             PMDuint16 phase,
                             PMDint16* command)
```

VB-Motion API

```
Dim command as Short
command = MagellanAxis.PhaseCommand( phase )
```

see

SetCurrentControlMode (p. 115)

GetPositionError 99h

Syntax

GetPositionError *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	Range	Scaling	Units
error	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts microsteps

Packet Structure

GetPositionError

0	*axis*	**99**h
15 12	11 8	7 0

First data word

	error (high-order part)
read	31 16

Second data word

	error (low-order part)
read	15 0

Description

GetPositionError returns the position error of the specified *axis*. The error is the difference between the actual position (encoder position) and the commanded position (instantaneous output of the trajectory generator). When used with the motor type set to microstepping or pulse & direction, the error is defined as the difference between the encoder position (represented in microsteps or steps) and the commanded position (instantaneous output of the trajectory generator).

Restrictions

C-Motion API

```
PMDresult PMDGetPositionError(PMDAxisInterface axis_intf,
                              PMDint32* error)
```

VB-Motion API

```
Dim error as Long
error = MagellanAxis.PositionError
```

see

Set/GetPosition (p. 172), **Set/GetPositionErrorLimit** (p. 173)

GetPositionLoopValue 55h

Syntax **GetPositionLoopValue** *axis node*

Motor Types

DC Brush	**Brushless DC**	▨	▨

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
node	*Integrator Sum*	0
	Integrator Contribution	1
	Derivative	2
	Biquad1 Input	3
	Biquad2 Input	4

Returned data

	Type	Range/Scaling
value	signed 32 bits	see below

Packet Structure

GetPositionLoopValue

0	*axis*	**55**h
15 12	11 8	7 0

First data word

write	*node*
	15 0

Second data word

read	*value* (high-order part)
	31 16

Third data word

read	*value* (low-order part)
	15 0

Description **GetPositionLoopValue** is used to find the value of a *node* in the position loop. See the product user guide for more information on the location of each *node* in the position loop processing. Though the data returned is signed 32 bits regardless of the *node*, the range and format varies depending on the *node*, as follows:

Node	Range	Scaling	Units
Integrator Sum	-2^{31} to $2^{31}-1$	unity	(counts)*cycles
Integrator Contribution	-2^{31} to $2^{31}-1$	$100*Kout/(2^{31})$	% Output
Derivative	-2^{15} to $2^{15}-1$	unity	(counts)/cycles
Biquad1 Input	-2^{15} to $2^{15}-1$	unity	counts
Biquad2 Input	-2^{15} to $2^{15}-1$	unity	counts

Restrictions

C-Motion API

```
PMDresult PMDGetPositionLoopValue (PMDAxisInterface axis_intf,
                                   PMDuint16 node,
                                   PMDint32* value)
```

VB-Motion API

```
Dim value as Long
value = MagellanAxis.PositionLoopValue( node )
```

see **Set/GetPositionLoop** (p. 174)

GetProductInfo 1h

Syntax

GetProductInfo *axis index*

Motor Types

DC Brush	Brushless DC	Microstepping

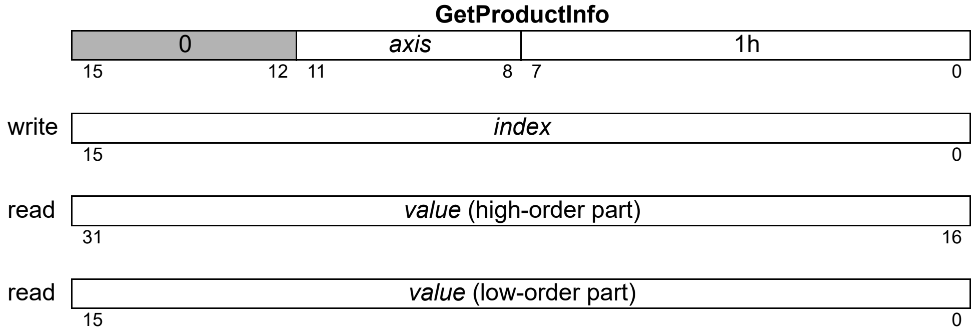
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
index	*firmware state*	0
	version	1
	product class	2
	checksum	3
	— (Reserved)	4
	part number 3:0	5
	part number 7:4	6
	part number 11:8	7
	part number 15:12	8
	— (Reserved)	9-12
	RAM size	13
	NVRAM size	14
	— (Reserved)	15-256
	boot version	257
	boot product class	258
	boot checksum	259
	boot part number 3:0	261
	boot part number 7:4	262
	boot part number 11:8	263
	boot part number 15:12	264

Returned Data

	Type
value	unsigned 32 bits

Packet Structure



Description

GetProductInfo is used to retrieve fixed information about the Magellan IC. All data is read in 32-bit units, most of the values are split into fields as explained below.

The *firmware state* is a an enumerated value, 0 means that the normal application firmware is running, and 1 indicates that the boot firmware, which is used for programming NVRAM, is running.

The *version,* and *boot version* consist of four 8-bit bytes, the least significant byte numbered zero. Byte 1 is the firmware major version, byte 0 is the minor version. Byte 2 is a custom code, zero for standard products. Byte 3 is reserved.

Description (cont.)

The *checksum* and *boot checksum* are 32 bit numbers that may be used to verify the identity of a product. The checksum values are documented in product release notes.

The *part number* and *boot part number* are 16 character strings indicating the IC and boot firmware part numbers . There is one ASCII character per 8-bit byte. The first character is stored in the least significant byte of *part number 3:0*, the second character in bits 15:8 of *part number 3:0*. The fourth character is stored in the least significant byte of *part number 7:4*, and so forth. Any unused characters at the end of the string are encoded as zero, ASCII null, but the string may not be null terminated.

The *RAM size* is the number of 32-bit words available for trace RAM.

The NVRAM size is the number of 16-bit words of non-volatile storage available.

GetProductInfo replaces and extends the Magellan commands **GetVersion** and **GetChecksum**. Magellan supports GetVersion, but that command always returns zero.

A value of zero returned by **GetVersion** should be taken to mean that **GetProductInfo** is supported.

Errors

Invalid parameter: index is not a supported value.

C-Motion API

```
PMDresult PMDGetProductInfo (PMDAxisInterface axis_intf, PMDuint16 index, PMDuint32* value);
```

Script API

```
GetProductInfo index
```

C# API

```
Int32 value = PMDAxis.GetProductInfo(PMDProductInfo index);
```

Visual Basic API

```
Int32 value = PMDAxis.GetProductInfo(ByVal index As PMDProductInfo)
```

see

NVRAM (p. 68), **SetBufferStart** (p. 104), **SetBufferLength** (p. 101), **ReadBuffer** (p. 72), **ReadBuffer16** (p. 73), **GetVersion** (p. 62)

GetSignalStatus A4h

Syntax

GetSignalStatus *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type
see below	unsigned 16 bits

Packet Structure

GetSignalStatus

0	*axis*	**A4**h
15 12	11 8	7 0

Data

read	0			0											
	15	14	13	12 11	10	9	8	7	6	5	4	3	2	1	0

Description

GetSignalStatus returns the contents of the Signal Status register for the specified ***axis***. The Signal Status register contains the value of the various hardware signals connected to each axis of the motion control IC. The value read is combined with the Signal Sense register (see **SetSignalSense** (p. 186)) and then returned to the user. For each bit in the Signal Sense register that is set to 1, the corresponding bit in the **GetSignalStatus** command will be inverted. Therefore, a low signal will be read as 1, and a high signal will be read as a 0. Conversely, for each bit in the Signal Sense register that is set to 0, the corresponding bit in the **GetSignalStatus** command is not inverted. Therefore, a low signal will be read as 0, and a high signal will be read as a 1.

All of the bits in the **GetSignalStatus** command are inputs, except for AxisOut and FaultOut. The value read for these bits is equal to the value output by the AxisOut and FaultOut mechanisms. See **SetAxisOutMask** (p. 92) and **SetFaultMask** (p. 137) for more information. The bit definitions are as follows:

Description	Bit Number	Description	Bit Number
Encoder A	0	Hall B	8
Encoder B	1	Hall C	9
Encoder Index	2	AxisOut	10
Capture Input	3	— (Reserved)	11–12
Positive Limit	4	/Enable In	13
Negative Limit	5	FaultOut	14
AxisIn	6	— (Reserved)	15
Hall A	7		

Atlas

Note that the */Enable In* and *FaultOut* signals are *not* the Atlas signals. In order to read the Atlas amplifier signal status the command must be directed to Atlas.

Restrictions

Depending on the product, some signals may not be present. See the product user guide. In ION products, when the capture source is set to Index, the Encoder Index input will be present as both the Encoder Index and the Capture Input bits. In MC58113 products the Capture Input bit is always used for the *Home* signal, regardless of the capture source.

C-Motion API

```
PMDresult PMDGetSignalStatus(PMDAxisInterface axis_intf,
                             PMDuint16* status)
```

VB-Motion API

```
Dim status as Short
status = MagellanAxis.SignalStatus
```

see **GetActivityStatus** (p. 25), **GetEventStatus** (p. 42), **GetSignalSense** (p. 186)

GetTemperature 53h

Syntax

GetTemperature *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned Data

	Type	Range	Scaling
temperature	signed 16 bits	-2^{15} *to* $2^{15}-1$	product specific

Packet Structure

GetTemperature

0	*axis*	**53**h
15 12	11 8	7 0

First data word

read	*temperature*
	15 0

Description

GetTemperature gets the most recent temperature reading from the temperature sensor(s) monitoring the ***axis***. Consult specific product documentation for scaling information.

Atlas

This command is relayed to an attached Atlas amplifier.

Restrictions

GetTemperature is only available in products equipped with temperature sensors. If ***axis*** has more than one temperature sensor, the temperature returned will be the average value of all sensor readings.

C-Motion API

```
PMDresult PMDGetTemperature(PMDAxisInterface axis_intf,
                            PMDint16* temperature)
```

VB-Motion API

```
Dim temperature as Short
temperature = MagellanAxis.Temperature
```

see

Get/SetOvertemperatureLimit (p. 159)

GetTime 3Eh

Syntax **GetTime**

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments None

Returned data

Name	Type	Range	Scaling	Units
time	unsigned 32 bits	0 *to* 2^{32}–1	unity	cycles

Packet Structure

GetTime

0	3Eh
15 8	7 0

First data word

read	*time* (high-order part)
	31 16

Second data word

read	*time* (low-order part)
	15 0

Description **GetTime** returns the number of cycles which have occurred since the motion control IC was last reset. The time per cycle is determined by **SetSampleTime**. The time count is not recalculated when setting the sample time. The time count is reset to zero whenever the synchronization mode is set.

Restrictions Time stops advancing when no axes are enabled.

C-Motion API

```
PMDresult PMDGetTime(PMDAxisInterface axis_intf,
                     PMDuint32* time)
```

VB-Motion API

```
Dim time as Long
time = MagellanObject.Time
```

see

Set/GetSampleTime (p. 180)
Set/GetSynchronizationMode (p. 192)

GetTraceCount BBh

Syntax

GetTraceCount

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

None

Returned data

Name	Type	Range	Scaling	Units
count	unsigned 32 bits	0 *to* 2^{32}–1	unity	samples

Packet Structure

GetTraceCount

	0	BBh
	15 — 8	7 — 0

First data word

	count (high-order part)
read	31 — 16

Second data word

	count (low-order part)
read	15 — 0

Description

GetTraceCount returns the number of points (variable values) stored in the trace buffer since the beginning of the trace.

Restrictions

In non-MC58113 products, if the trace mode is set to “rolling” and the buffer wraps, **GetTraceCount** returns the number of samples in the filled buffer.

C-Motion API

```
PMDresult PMDGetTraceCount(PMDAxisInterface axis_intf,
                              PMDuint32* count)
```

VB-Motion API

```
Dim count as Long
count = MagellanObject.TraceCount
```

see

ReadBuffer (p. 72), **Set/GetTraceStart** (p. 196), **Set/GetTraceStop** (p. 199), **Set/GetBufferLength** (p. 101)

GetTraceStatus BAh

Syntax

GetTraceStatus

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

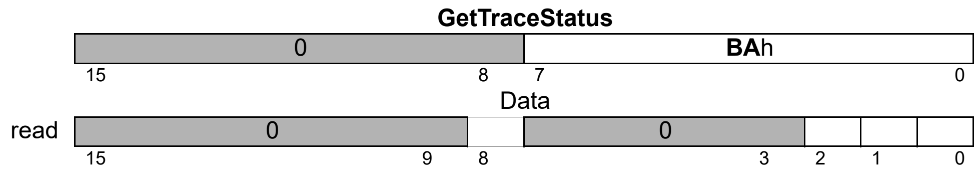
Arguments

None

Returned data

Name	Type
see below	unsigned 16 bits

Packet Structure



Description

GetTraceStatus returns the trace status. The definitions of the individual status bits are as follows:

Name	Bit Number	Description
Wrap Mode	0	Set to 0 when trace is in one-time mode, 1 when in rolling mode.
Activity	1	Set to 1 when trace is active (currently tracing), 0 if trace not active.
Data Wrap	2	Set to 1 when trace has wrapped, 0 if it has not wrapped. If 0, the buffer has not yet been filled, and all recorded data is intact. If 1, the trace has wrapped to the beginning of the buffer; any previous data may have been overwritten if not explicitly retrieved by the host using the **ReadBuffer** command while the trace is active.
—	3-7	— (Reserved)
Trigger Mode	8	Set to 0 when in Internal Trigger mode, 1 when in External Trigger mode. See **SetTraceMode** (p. 193) for explanation.
—	9-15	— (Reserved)

Restrictions

C-Motion API

```
PMDresult PMDGetTraceStatus(PMDAxisInterface axis_intf,
                                  PMDuint16* status)
```

VB-Motion API

```
Dim status as Short
status = MagellanObject.TraceStatus
```

see

Set/GetTraceStart (p. 196), **Set/GetTraceMode** (p. 193)

GetTraceValue 28h

Syntax

GetTraceValue *variableID*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

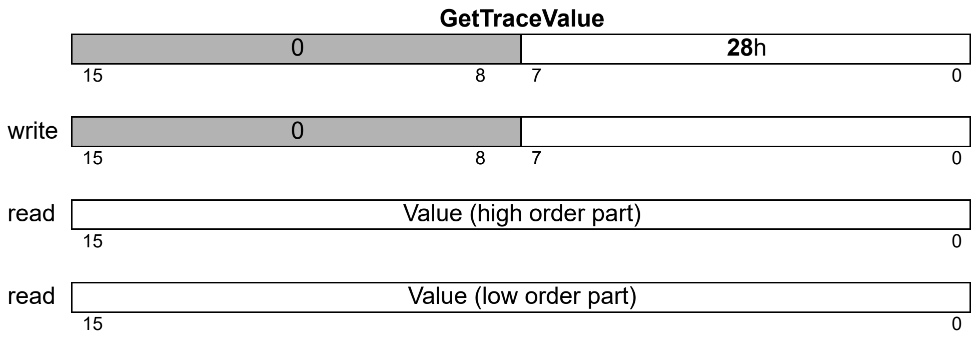
Arguments

Name	Type	Encoding
variableID	unsigned 8 bit	see below

Returned data

Value	Type	Range/Scaling
	32 bit	see below

Packet Structure



Description

GetTraceValue returns a single sample of any trace variable, without using the trace mechanism. The variableID encoding is the same as for **SetTraceVariable**. The use of this command does not change or depend upon any of the trace parameters.

C-Motion API

```
PMDresult PMDGetTraceValue(PMDAxisInterface axis_intf,
                           PMDuint8 variable, PMDuint32 *value)
```

VB-Motion API

```
MagellanAxis.TraceValue([in] variable
[out] value)
```

see

PMDSetTraceVariable (p. 202)

GetVersion 8Fh

Syntax

GetVersion

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

None

Returned data

Name	Type
version	unsigned 32 bits

Packet Structure

GetVersion

	0	8Fh
	15 8	7 0

First data word

read	product family	motor type	number of axes	special	# chips
	31 28	27 24	23 20	19 18	17 16

Second data word

read	customization code	product version	major version	minor version
	15 8	7 6	5 4	3 0

Description

GetVersion returns product information encoded as shown in the preceding packet structure diagram. Individual data fields are encoded as defined in the following table.

Name	Description	Encoding
product family	Navigator	2
	Pilot	3
	Magellan	5
	ION	9
motor type	Servo	1
	Brushless	3
	Microstepping	4
	Pulse & Direction	5
	All Motor Types	8
	ION–Any Motor Type	9
number of axes	Maximum number of supported axes	1 to 15
special	(Reserved)	0 to 3
# chips		0 to 3
customization code	None Other	0 1 to 255
product version		0 to 3
major s/w version		0 to 3
minor s/w version		0 to 15

As a special case, when the number of chips field is zero, the number of axes field should be interpreted as a sub-product specification. This scheme is used for the MC58113 series motion control ICs, which use a number of axes/sub-product field of 1. For example, the version number returned by an MC58113 processor version 1.0 is 0x58100010.

Restrictions

Note that in the C-Motion function **PMDGetVersion**, the special attributes value and the chip count values are combined and returned in a single parameter (*`special_and_chip_count`*). Chip count is encoded in bits 0–1 of this value; special is encoded in bits 2–3. Likewise for the *`major`* parameter. The *`major`* version is encoded in bits 0-1 and the product version is encoded in bits 2-3.

C-Motion API

```
PMDresult PMDGetVersion(PMDAxisInterface axis_intf,
                        PMDuint16* family,
                        PMDuint16* motorType,
                        PMDuint16* numberAxes,
                        PMDuint16* special_and_chip_count,
                        PMDuint16* custom,
                        PMDuint16* major,
                        PMDuint16* minor)
```

VB-Motion API

```
Dim version as Long
version  = MagellanObject.Version
```

see

InitializePhase 7Ah

Syntax

InitializePhase *axis*

Motor Types

	Brushless DC		

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

None

Packet Structure

InitializePhase

15–12	11–8	7–0
0	*axis*	**7A**h

Description

InitializePhase initializes the phase angle for the specified ***axis*** using the mode (Hall-based or algorithmic) specified by the **SetPhaseInitializationMode** command.

Restrictions

Warning: If the phase initialization mode has been set to algorithmic, then, after this command is sent, the motor may suddenly move in an uncontrolled manner.

C-Motion API

```
PMDresult PMDInitializePhase(PMDAxisInterface axis_intf)
```

VB-Motion API

```
MagellanAxis.InitializePhase()
```

see

GetPhaseCommand (p. 50), **Set/GetCommutationMode** (p. 109)

MultiUpdate 5Bh

Syntax

MultiUpdate *mask*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
mask	*None*	0
	Axis1 Mask	1
	Axis2 Mask	2
	Axis3 Mask	4
	Axis4 Mask	8

Returned data

None

Packet Structure

MultiUpdate

0	**5B**h
15 ... 8	7 ... 0

Data

	0	*mask*
write	15 ... 4	3 ... 0

Description

MultiUpdate causes an update to occur on all axes whose corresponding bit is set to 1 in the ***mask*** argument. After this command is executed, all axes which are selected using the mask will perform an **Update**. The paramater groups that are copied from their buffered versions into the corresponding run-time registers is determined by the update mask of each ***axis***, as shown in the table below.

Group	Command/Parameter
Trajectory	Acceleration Deceleration Gear Ratio Jerk Position Profile Mode Stop Mode Velocity ClearPositionError
Position Servo	Derivative Time Integrator Sum Limit Kaff Kd Ki Kp Kvff Kout Motor Command
Current Loops	Integrator Sum Limit
	Ki
	Kp

Each axis will be updated in turn, from the lowest numbered to the highest. If an error occurs during the update of an axis, for example a move into an active limit switch, then that update will be aborted, the error code returned, and no higher-numbered axes will be updated. The InstructionError bit of the event status register for each axis may be tested to discover which axis had an update failure.

Atlas

This command does not require any additional Atlas communication. It may cause an Atlas update by using the update bit in the Atlas torque command, see *Atlas Digital Amplifier Complete Technical Reference* for more information.

Restrictions

C-Motion API

```
PMDresult PMDMultiUpdate(PMDAxisInterface axis_intf,
                         PMDuint16 mask)
```

VB-Motion API

```
MagellanObject.MultiUpdate( [in] mask )
```

see

GetEventStatus (p. 42), **Update** (p. 215), **Set/GetUpdateMask** (p. 211)

NoOperation 00h

Syntax **NoOperation**

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments None

Returned data None

Packet Structure

NoOperation

0	**00**h
15 8	7 0

Description The **NoOperation** command has no effect on the motion control IC.

Restrictions

C-Motion API

```
PMDresult PMDNoOperation(PMDAxisInterface axis_intf)
```

VB-Motion API

```
MagellanObject.NoOperation()
```

see

NVRAM

Syntax

NVRAM *axis option value*

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
option	*Atlas NVRAM mode*	0
	Magellan NVRAM mode	256
	Erase NVRAM	1
	Write	2
	Block Write Begin	3
	Block Write End	4
	Skip	8
	— (Reserved)	9
	— (Reserved)	10
	Open NVRAM	11

	Type	Range
value	unsigned 16 bit	see below

Packet Structure

NVRAM

0	*axis*	**30**h
15 12	11 8	7 0

write

option
15 0

write

value
15 0

Description

The **NVRAM** command is used to write the non-volatile RAM (NVRAM) used for initialization on products that support it, including MC58113 series motion controllers, N-series ION digital drives, and Atlas digital amplifiers. The **NVRAM** command is first used to put the processor to be programmed into NVRAM mode, which supports only the commands necessary for its purpose. Once the processor is in NVRAM mode more **NVRAM** commands are used to erase and re-program NVRAM. NVRAM mode is exited by using the reset command; when programming Atlas this command must be sent to the Atlas axis. Even when programming Atlas all **NVRAM** commands should be sent to the Magellan axis, otherwise spurious SPI checksum errors will be signaled.

Changing to NVRAM mode, erasing, or writing NVRAM data may take more time than the other commands. When programming the MC58113 NVRAM the timeout period should be increased to at least 10 seconds; after each operation fully completes the return status may be read to confirm that the operation succeeded.

Description (cont'd)

When programming Atlas a different procedure is required. Atlas will return command status after checking arguments but before beginning an NVRAM operation, and will not respond to SPI commands while busy programming NVRAM. The Magellan controlling Atlas should be polled using **GetDriveStatus** after sending a **NVRAM** command, until the Atlas Not Connected bit is clear. If a flash error has occurred then the Instruction Error bit of the Event Status register will be set, and the **GetInstructionError** command may be sent to Atlas for more information. When writing NVRAM data one word at a time it is not necessary to check for error status after each write, the error status is latched, and may be checked periodically.

The option argument to **NVRAM** specifies the particular operation to perform:

NVRAM mode (256) will put an MC58113 series motion control IC into NVRAM mode. Motor output must be disabled.

Atlas NVRAM mode (0) will put an attached Atlas amplifier into NVRAM mode. Motor output must be disabled. All erase or program commands are sent to the Atlas amplifier unless the Magellan processor itself is in NVRAM mode. The value argument should be zero for this command.

The remaining operations will succeed only if either the Magellan processor itself or an attached Atlas amplifier is in NVRAM mode, otherwise an Invalid register state for command error will be raised. The value argument should be zero for this command.

Erase NVRAM (1) will erase the entire non-volatile memory, meaining that all bits will be set. NVRAM must be completely erased before any words may be written. The value argument should be zero for this command.

Open NVRAM (11) will allow writing to the non-volatile memory without erasing it. In this case the Skip option must be used to begin writing after the last previously written word. The memory may be read using **ReadBuffer16** in order to determine what has been written, but this must be done before entering NVRAM mode, which does not support the buffer commands.

Write (2) will write a single word of NVRAM, which is specified by the value argument. Words are written in sequence, from the beginning.

Skip (8) may be used to leave the number of words specified in the value argument unwritten, that is, with a value of 0xFFFF. Writing may resume afterwards. It is not necessary to use this command in the usual case.

Block Write Begin (3) and Block Write End (4) may be used to speed up NVRAM operations that are limited by communication bandwidth; their use is not required.

A block write operation is begun by using the **BlockWriteBegin** command, with the number of words that will be sent as a block specified in the value argument. A block may be at most 32 words. No polling procedure is required after a Block Write Begin command.

The next step is to send the data words. These are sent without the usual Magellan command format, therefore no other commands may be sent until the entire block is transmitted.

If using serial communications the words are sent as is, high byte first.

If using CANBus, the words are sent without any additional formatting. At most four words may be sent per CAN packet.

If using SPI communications, the words are sent without any additional formatting. at most four words may be sent for each cycle of the *~HostSPIEnable* signal.

Description (cont'd)

If using parallel communications the words are sent without any additional formatting, with the *~HostWrite* signal high, that is, as though they were command words. At most one word may be sent per *~HostWrite* cycle.

The block write operation is concluded by sending a **BlockWriteEnd** comamnd. The value argument to this command must be the 16-bit ones complement checksum of all words sent since the **BlockWriteBegin** command. If the checksum matches then the processor will write all words to NVRAM, in order. When programming MC58113 NVRAM a long wait may be required. When programming Atlas NVRAM the polling procedure described above for NVRAM writes should be followed.

Atlas

This command will be relayed to an attached Atlas amplifier unless the NVRAM mode (256) option is selected or it is sent to an MC58113 series motion control IC which is in NVRAM mode.

Restrictions

Once put in NVRAM mode an Atlas amplifier or MC58113 series motion control IC will accept only a restricted set of commands. There is no way to enable motor output, and Atlas will not accept torque commands.

VB-Motion API

```
MagellanAxis.NVRAM( [in] option, [in] value )
```

C-Motion API

```
PMDresult PMDNVRAM      (PMDAxisInterface axis_intf,
                                     PMDuint16 option,
                                     PMDuint16 value);
```

see

GetDriveStatus (p. 38), **GetEventStatus** (p. 42), **GetInstructionError** (p. 46), **Reset** (p. 75)

ReadAnalog EFh

Syntax

ReadAnalog *portID*

Motor Types

DC Brush	**Brushless DC**	**Microstepping**	**Pulse & Direction**

Arguments

Name	**Type**	**Range**	**Scaling**	**Units**
portID	unsigned 16 bits	0 *to* 7	unity	-

Returned data

	Type	**Range**	**Scaling**	**Units**
value	unsigned 16 bits	0 *to* 2^{16}–1	$100/2^{16}$	% input

Packet Structure

ReadAnalog

	0	**EF**h
	15 – 8	7 – 0

First data word

write	0	*portID*
	15 – 3	2 – 0

Second data word

read	*value*
	15 – 0

Description

ReadAnalog returns a 16-bit value representing the voltage presented to the specified analog input. See the product user guide for more information on analog input and scaling.

Restrictions

C-Motion API

```
PMDresult PMDReadAnalog(PMDAxisInterface axis_intf, PMDuint16 portID,
                        PMDuint16* value)
```

VB-Motion API

```
Dim value as Short
value = MagellanObject.Analog( portID )
```

see

ReadBuffer C9h

Syntax

ReadBuffer *bufferID*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range
bufferID	unsigned 16 bits	0 *to* 31

Returned data

	Type	Range
data	signed 32 bits	-2^{31} *to* $2^{31}-1$

Packet Structure

ReadBuffer

15 … 8	7 … 0
0	**C9**h

First data word

	15 … 5	4 … 0
write	0	*bufferID*

Second data word

	31 … 16
read	*data* (high-order part)

Third data word

	15 … 0
read	*data* (low-order part)

Description

ReadBuffer returns the 32-bit contents of the location pointed to by the read buffer index in the specified buffer. After the contents have been read, the read index is incremented by 1. If the result is equal to the buffer length (set by **SetBufferLength**), the index is reset to zero (0). The read index for buffer zero is automatically changed at the completion of a trace when in rolling trace mode.

Restrictions

C-Motion API

```
PMDresult PMDReadBuffer16(PMDAxisInterface axis_intf, PMDuint16 buffer-
ID,
                         PMDint32* data)
```

VB-Motion API

```
Dim data as Long
Data = MagellanObject.ReadBuffer16( bufferID )
```

see

Set/GetBufferReadIndex (p. 103), **Set/GetBufferStart** (p. 104), **Set/GetBufferLength** (p. 101)

ReadBuffer16 CDh

Syntax

ReadBuffer16 *bufferID*

Motor Types

DC Brush	**Brushless DC**	**Microstepping**	**Pulse & Direction**

Arguments

Name	**Type**	**Range**
bufferID	unsigned 16 bits	0 *to* 31

Returned data

	Type	**Range**
data	signed 16 bits	-2^{15} *to* $2^{15}-1$

Packet Structure

ReadBuffer

0	**CDh**
15 … 8	7 … 0

First data word

write	0	*bufferID*
	15 … 5	4 … 0

Second data word

read	*data*
	31 … 16

Description

ReadBuffer16 returns the 16-bit contents of the location pointed to by the read buffer index in the specified buffer. After the contents have been read, the read index is incremented by 1. If the result is equal to the buffer length (set by **SetBufferLength**), the index is reset to zero (0). This command is intended to read from a buffer located in non-volatile RAM, which has a 16-bit word size. ReadBuffer should be used for all other buffers.

Restrictions

This command is only available on products that support non-volatile RAM.

C-Motion API

```
PMDresult PMDReadBuffer(PMDAxisInterface axis_intf, PMDuint16 bufferID,
                        PMDint32* data)
```

VB-Motion API

```
Dim data as Long
Data = MagellanObject.ReadBuffer( bufferID )
```

see

Set/GetBufferReadIndex (p. 103), **WriteBuffer** (p. 216), **Set/GetBufferStart** (p. 104), **Set/GetBufferLength** (p. 101)

ReadIO 83h

Syntax

ReadIO *address*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

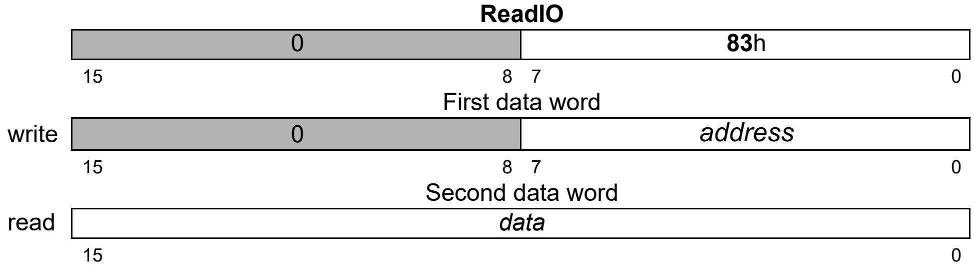
Arguments

Name	Type	Range
address	unsigned 16 bits	0 *to* 255

Returned data

	Type	Range
data	unsigned 16 bits	0 *to* 2^{16}–1

Packet Structure



Description

ReadIO reads one 16-bit word of data from the device at ***address***. The ***address*** is an offset from location 1000h of the motion control IC's peripheral device address space.

The format and interpretation of the 16-bit data word are dependent on the user-defined device being addressed. User-defined I/O can be used to implement a number of features, including additional parallel I/O, flash memory for non-volatile configuration information storage, or display devices such as LED arrays.

Restrictions

This command is not available in products without a parallel I/O port.

C-Motion API

```
PMDresult PMDReadIO(PMDAxisInterface axis_intf, PMDuint16 address,
                    PMDuint16* data);
```

VB-Motion API

```
Dim data as Short
data = MagellanObject.IO( address )
```

see

WriteIO (p. 217)

Syntax

Reset

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

None

Returned data

None

Packet Structure

Reset

15 … 8	7 … 0
0	**39**h

Description

Reset restores the motion control IC to its initial condition, setting all motion control IC variables to their default values. Most variables are motor-type independent; however several default values depend upon the configured motor type of the axis. Some of the default values also depend on the state of Magellan pin OutputMode0 when power is applied, if this pin is grounded, Magellan will be in an "Atlas-compatible" state, if it is floating, "backwards-compatible." MC58113 series products always behave in an Atlas-compatible way. The motor-type independent values are listed here.

	Default Value	Buffered
Breakpoints and Interrupts		
Breakpoint 1	0	NO
Breakpoint 2	0	NO
Breakpoint Value 1	0	NO
Breakpoint Value 2	0	NO
Breakpoint 1 Update Mask	0Bh (products with current control) 03h (products without)	NO
Breakpoint 2 Update Mask	0Bh (products with current control) 03h (products without)	NO
Interrupt Mask	0	NO
Commutation		
Commutation Mode	motor dependent	NO
Phase Angle	0	NO
Phase Counts	motor dependent	NO
Phase Offset	–1	NO
Phase Prescale	0	NO
Phase Initialize Mode	0	NO
Phase Initialize Time	0	NO
Phase Correction Mode	motor dependent	NO
Current Control		
Current Control Mode	0	YES-Current Loop
Current Loop Kp (both A and B loops)	0	YES-Current Loop
Current Loop Ki (both A and B loops)	0	YES-Current Loop
Current Loop Integrator Sum Limit (both A and B loops)	0	YES-Current Loop
FOC Kp (both D and Q loops)	0	YES-Current Loop
FOC Ki (both D and Q loops)	0	YES-Current Loop

	Default Value	Buffered
Current Control (cont.)		
FOC Integrator Sum Limit	0	YES-Current Loop
Holding Motor Limit	32767	NO
Holding Delay	motor dependent	NO
Digital Servo Filter		
Position Error Limit	65535	NO
Position Loop Biquad Coeffs	All 0	YES-PositionLoop
Position Loop Biquad Enables	Both 0	YES-Position Loop
Position Loop Kvff	0	YES-Position Loop
Position Loop Kaff	0	YES-Position Loop
Position Loop Kp	0	YES-Position Loop
Position Loop Ki	0	YES-Position Loop
Position Loop Kd	0	YES-Position Loop
Position Loop Integrator Sum Limit	0	YES-Position Loop
Position Loop Derivative Time	1	YES-Position Loop
Position Loop Kout	65535	YES-Position Loop
Motor Limit	32767	NO
Motor Bias	0	NO
Motor Command	0	YES-Position Loop
Auxiliary Encoder Source	0	NO
Encoder		
Actual Position	0	NO
Actual Position Units	motor dependent	NO
Capture Source	0	NO
Encoder Modulus	0	NO
Encoder Source	motor dependent	NO
Encoder To Step Ratio	00010001h	NO
Motor Output		
Operating Mode	0033h (Magellan backwards-compatible) 0001h (ION, Magellan Atlas-compatible)	NO
Active Operating Mode	0033h (Magellan backwards-compatible) 0001h (ION, Magellan Atlas-compatible)	NO
Output Mode	motor dependent	NO
Motor Type	product dependent	NO
PWM Frequency	motor dependent	NO
Step Range	1	NO
Position Servo Loop Control		
Motion Complete Mode	0	NO
Sample Time	see Notes	NO
Settle Time	0	NO
Settle Window	0	NO
Tracking Window	0	NO
Profile Generation		
Acceleration	0	YES-Trajectory
Deceleration	0	YES-Trajectory
Gear Master	0	NO
Gear Ratio	0	YES-Trajectory
Jerk	0	YES-Trajectory
Position	0	YES-Trajectory

	Default Value	Buffered
Profile Generation (cont.)		
Profile Mode	0	YES-Trajectory
Start Velocity	0	NO
Stop Mode	0	YES-Trajectory
Velocity	0	YES-Trajectory
RAM Buffer		
Buffer Length	0-Magellan 0180h-ION	NO
Buffer Read Index	0	NO
Buffer Start	0	NO
Buffer Write Index	0	NO
Safety		
Positive Limit Event Action	8	NO
Negative Limit Event Action	8	NO
Motion Error Event Action	motor dependent	NO
Current Foldback Event Action	7	NO
OvervoltageThreshold	see *specific product manual*	NO
Undervoltage Threshold	see *specific product manual*	NO
OvertemperatureThreshold	see *specific product manual*	NO
FaultOut Mask	0600h	NO
Continuous Current Limit	see *specific product manual*	
Energy Limit	see *specific product manual*	
Status Registers and AxisOut Indicator		
AxisOut Source Axis	0	NO
AxisOut Register	0	NO
AxisOut Selection Mask	0	NO
AxisOut Sense Mask	0	NO
Signal Sense	motor dependent	NO
Traces		
Trace Mode	0	NO
Trace Period	1	NO
Trace Start	0	NO
Trace Stop	0	NO
Trace Variables	all are 0	NO
Miscellaneous		
Update Mask	0Bh (products with current control) 03h (products without)	NO
CAN Mode	C000h (see Notes)	NO
Serial Port Mode	0004h (see Notes)	NO

The motor-type dependent default values are listed in the following tables.

Variable	DC Brush	Brushless DC (3 phase)	Brushless DC (2 phase)
Actual Position Units	0	0	0
Commutation Mode	-	0	0
Encoder Source	0	0	0
Motion Error Event Action	5	5	5
Output Mode	1-Magellan 2-ION 10-MC58113	2	2
Phase Correction Mode	-	1	1
PWM Frequency (kHz)	20	20	20
SPI Mode	0	-	-
Phase Counts	-	1	1
Holding Delay	-	-	-
Signal Sense	0 (backwards-compatible), 0800h (Atlas-compatible)	0 (backwards-compatible), 0800h (Atlas-compatible) 0 (MC58113)	0

Variable	Microstepping (3 phase)	Microstepping (2 phase)	Pulse & Direction
Actual Position Units	1	1	1
Commutation Mode	0	0	-
Encoder Source	2	2	3
Motion Error Event Action	0	0	0
Output Mode	2	1-Magellan 2-ION	-
Phase Correction Mode	-	-	-
PWM Frequency (kHz)	20	80-Magellan 20-ION 20-MC58113	-
SPI Mode	-	-	-
Phase Counts	256	256	-
Holding Delay	32767	32767	20
Signal Sense	0	0	0800h

Notes

All axes supported by the motion control IC are enabled at reset.

In some products, CAN Mode and Serial Port Mode defaults are defined at reset by a parallel bus read.

In ION products, the reset defaults for CAN Mode and the Serial Port Mode used for RS485 can be over-ridden using the **SetDefault** command. See the *ION Digital Drive User Manual.*

See **Set/GetSampleTime** (p. 180) for more information regarding SampleTime.

Atlas

The Magellan reset command does *not* cause any attached Atlas amplifiers to be reset. When Magellan re-connects to any such Atlas amplifiers it will check their motor types, set their operating mode, and set their current foldback event actions.

Restrictions

The typical time before the device is ready for communication after a reset is 20ms for Magellan products, and 100ms for ION products.

The MC55110 and the MC58110 have a maximum Step Range of 100 ksteps/sec, which cannot be changed.

Not all of the listed variables are available on all products. See the product user guide.

C-Motion API `PMDresult PMDReset(PMDAxisInterface axis_intf)`

VB-Motion API `MagellanObject.Reset()`

see **SetDefault** (p. 123)

ResetEventStatus 34h

Syntax

ResetEventStatus *axis mask*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mask	*Motion Complete*	FFFEh
	Wrap-around	FFFDh
	Breakpoint 1	FFFBh
	Capture Received	FFF7h
	Motion Error	FFEFh
	Positive Limit	FFDFh
	Negative Limit	FFBFh
	Instruction Error	FF7Fh
	Disable	FEFFh
	Overtemperature Fault	FDFFh
	Drive Exception	FBFFh
	Commutation Error	F7FFh
	Current Foldback	EFFFh
	Breakpoint 2	BFFFh

Returned data

None

Packet Structure

ResetEventStatus

15–12	11–8	7–0
0	*axis*	**34**h

Data

	15–0
write	*mask*

Description

ResetEventStatus clears (sets to 0), for the specified ***axis***, each bit in the Event Status register that has a value of 0 in the ***mask*** sent with this command. All other Event Status register bits (bits that have a mask value of 1) are unaffected.

Events that cause changes in operating mode or trajectory require, in general, that the corresponding bit in Event Status be cleared prior to returning to operation. That is, prior to restoring the operating mode (in cases where the event caused a change in it) or prior to performing another trajectory move (in cases where the event caused a trajectory stop). The one exception to this is *Motion Error*, which is not required to be cleared if the event action for it includes disabling of the position loop.

Atlas

When clearing bits 10 (Drive Exception), or 12 (Current Foldback), this command will be sent to an attached Atlas amplifier before being applied to the local Magellan register.

Note that bit 9 (Overtemperature Fault) is not used for Atlas axes.

Restrictions

Not all bits in **ResetEventStatus** are supported in some products. See the product user manual.

C-Motion API

```
PMDresult PMDResetEventStatus(PMDAxisInterface axis_intf,
                              PMDuint16 status)
```

VB-Motion API

```
MagellanAxis.ResetEventStatus( mask )
```

see

GetEventStatus (p. 42)

RestoreOperatingMode 2Eh

Syntax

RestoreOperatingMode *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet Structure

RestoreOperatingMode

0	*axis*	**2E**h
15 12	11 8	7 0

Description

RestoreOperatingMode is used to command the *axis* to return to its static operating mode. It should be used when the active operating mode has changed due to actions taken from safety events or other programmed events. Calling **RestoreOperatingMode** will re-enable all loops that were disabled as a result of events.

Atlas

This command will be sent to an attached Atlas amplifier before being applied to the local Magellan register.

Restrictions

Before using **RestoreOperatingMode** to return to the static operating mode, the event status bits should all be cleared. If a bit in event status that caused a change in operating mode is not cleared, this command will return an error. The exceptions to this are Motion Error and the breakpoint events, which do not have to be cleared prior to restoring the operating mode.

Though **RestoreOperatingMode** will re-enable the trajectory generator (if it was disabled as a result of an event action), it will not resume a move. This must be done through an **Update** or **MultiUpdate**.

C-Motion API

```
PMDresult PMDRestoreOperatingMode(PMDAxisInterface axis_intf)
```

VB-Motion API

```
MagellanAxis.RestoreOperatingMode()
```

see

GetActiveOperatingMode (p. 24), **Set/GetOperatingMode** (p. 156), **Set/GetEventAction** (p. 135), **Set/GetBreakpoint** (p. 94)

SetAcceleration buffered 90h
GetAcceleration 4Ch

Syntax

SetAcceleration *axis acceleration*
GetAcceleration *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
acceleration	unsigned 32 bits	0 *to* 2^{31}–1	$1/2^{16}$	counts/cycle^2 microsteps/cycle^2

Packet Structure

SetAcceleration

0	*axis*	**90**h
15 – 12	11 – 8	7 – 0

First data word

write	*acceleration* (high-order part)
	31 – 16

Second data word

write	*acceleration* (low-order part)
	15 – 0

GetAcceleration

0	*axis*	**4C**h
15 – 12	11 – 8	7 – 0

First data word

read	*acceleration* (high-order part)
	31 – 16

Second data word

read	*acceleration* (low-order part)
	15 – 0

Description

SetAcceleration loads the maximum acceleration buffer register for the specified *axis*. This command is used with the Trapezoidal, Velocity Contouring, and S-curve profiling modes.

GetAcceleration reads the maximum acceleration buffer register.

Scaling example: To load a value of 1.750 counts/cycle^2, multiply by 65,536 (given 114,688) and load the resultant number as a 32-bit number, giving 0001 in the high word and C000h in the low word. Values returned by **GetAcceleration** must correspondingly be divided by 65,536 to convert to units of counts/cycle^2 or steps/cycle^2.

Restrictions

SetAcceleration may not be issued while an axis is in motion with the S-curve profile.

SetAcceleration is not valid in Electronic Gear profile mode.

SetAcceleration is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetAcceleration(PMDAxisInterface axis_intf,
                             PMDuint32 acceleration)
PMDresult PMDGetAcceleration(PMDAxisInterface axis_intf,
                             PMDuint32* acceleration)
```

VB-Motion API

```
Dim acceleration as Long
MagellanAxis.Acceleration = acceleration
acceleration = MagellanAxis.Acceleration
```

see

Set/GetDeceleration (p. 122), **Set/GetJerk** (p. 148), **Set/GetPosition** (p. 172), **Set/GetVelocity** (p. 213), **MultiUpdate** (p. 65), **Update** (p. 215)

SetActualPosition 4Dh
GetActualPosition 37h

Syntax

SetActualPosition *axis position*
GetActualPosition *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
position	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts microsteps

Packet Structure

SetActualPosition

15–12	11–8	7–0
0	*axis*	**4D**h

First data word

	31–16
write	*position* (high-order part)

Second data word

	15–0
write	*position* (low-order part)

GetActualPosition

15–12	11–8	7–0
0	*axis*	**37**h

First data word

	31–16
read	*position* (high-order part)

Second data word

	15–0
read	*position* (low-order part)

Description

SetActualPosition loads the position register (encoder position) for the specified ***axis***. At the same time, the commanded position is replaced by the loaded value minus the position error. This prevents a servo "bump" when the new axis position is established. The destination position (see **SetPosition** (p. 172)) is also modified by this amount so that no trajectory motion will occur when the **Update** instruction is issued. In effect, this instruction establishes a new reference position from which subsequent positions can be calculated. It is commonly used to set a known reference position after a homing procedure.

Note: For axes configured as pulse & direction or microstepping motor types, actual position units determines if the position is specified and returned in units of counts or steps. Additionally, for these motor types, the position error is zeroed when the **SetActualPosition** command is sent. **SetActualPosition** takes effect immediately, it is not buffered.

GetActualPosition reads the contents of the encoder's actual position register. This value will be accurate to within one cycle (as determined by **Set/GetSampleTime**).

Restrictions

C-Motion API

```
PMDresult PMDSetActualPosition(PMDAxisInterface axis_intf,
                               PMDint32 position)
PMDresult PMDGetActualPosition(PMDAxisInterface axis_intf,
                               PMDint32* position)
```

VB-Motion API

```
Dim position as Long
MagellanAxis.ActualPosition = position
position = MagellanAxis.ActualPosition
```

see **GetPositionError** (p. 51), **Set/GetActualPositionUnits** (p. 87), **AdjustActualPosition** (p. 16)

SetActualPositionUnits BEh
GetActualPositionUnits BFh

Syntax

SetActualPositionUnits *axis mode*
GetActualPositionUnits *axis*

Motor Types

▨	▨	**Microstepping**	**Pulse & Direction**

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Counts*	0
	Steps	1

Packet Structure

SetActualPositionUnits

15 – 12	11 – 8	7 – 0
0	*axis*	**BE**h

Data

write

15 – 1	0
0	*mode*

GetActualPositionUnits

15 – 12	11 – 8	7 – 0
0	*axis*	**BF**h

Data

read

15 – 1	0
0	*mode*

Description

SetActualPositionUnits determines the units used by the **Set/GetActualPosition**, **AdjustActualPosition** and **GetCaptureValue** for the specified *axis*. It also affects the trace variable Actual Position. When set to *Counts*, position units are in encoder counts. When set to *Steps*, position units are in steps/microsteps. The step position is calculated using the ratio as set by the **SetEncoderToStepRatio** command.

GetActualPositionUnits returns the position units for the specified *axis*.

Restrictions

The trace variable, capture value, is not affected by this command. The value is always in counts.

C-Motion API

```
PMDresult PMDSetActualPositionUnits(PMDAxisInterface axis_intf,
                                     PMDuint16 mode)
PMDresult PMDGetActualPositionUnits(PMDAxisInterface axis_intf,
                                     PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.ActualPositionUnits = mode
mode = MagellanAxis.ActualPositionUnits
```

see

Set/GetActualPosition (p. 85), **Set/GetEncoderToStepRatio** (p. 134), **AdjustActualPosition** (p. 16), **GetCaptureValue** (p. 29), **Set/GetTraceVariable** (p. 202)

SetAnalogCalibration 29h
GetAnalogCalibration 2Ah

Syntax

SetAnalogCalibration *axis channel offset*
GetAnalogCalibration *axis channel*

Motor Types

DC Brush	**Brushless DC**	**Microstepping**	▨

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
coefficient	*current leg A offset*	0
	current leg B *offset*	1
	current leg C offset	2
	current leg D offset	3
	encoder cos offset	0x300
	encoder sin offset	0x301
	encoder cos gain	0x302
	encoder sin gain	0x303
	encoder phase gain	0x304
	encoder sin/cos interpolation factor	0x30D
	encoder sin/cos correction table enable	0x30F
	encoder sin/cos correction table	0x310 - 0x32F
value	see below	

Packet Structure

SetAnalogCalibration

	▨	*axis*	**29**h
	15 — 12	11 — 8	7 — 0

First data word

write	*coefficient*	
	15	0

Second data word

write	*value*	
	15	0

GetAnalogCalibration

	▨	*axis*	**2A**h
	15 — 12	11 — 8	7 — 0

write	*coefficient*	
	15	0

read	*value*	
	15	0

Description

The **SetAnalogCalibration** command sets offsets and gains that are used to correct analog inputs for the vagaries of external amplification circuitry.

It is frequently convenient to use the **CalibrateAnalog** command to compute the appropriate coefficients, which may later be read using **GetAnalogCalibration**, and modified if needed by **SetAnalogCalibration**.

Description (cont.)

The four leg current offsets are subtracted from the raw analog readings, as returned by the **ReadAnalog** command. They are signed 16-bit numbers ranging from -2^{15} to 2^{15} - 1

The encoder calibration coefficients are currently used for the N-series ION, which supports analog sin/cos encoders. Please consult the users guide for an overview of sin/cos encoder operation.

The encoder sin and cos offsets are signed 16-bit numbers that are added to the raw analog readings after subtracting a bias of 0x8000 (32768). The raw analog sin and cos values may be read using trace or the **GetTraceValue** command.

The encoder sin and cos gains are unsigned 16-bit numbers scaled by 2^{14}, that is 16384 corresponds to 1.0. The sin gain multiplies the raw sin input after the offset and bias are added, the cos gain multiplies the raw cos input after the offset and bias are added.

The phase gain is a signed 16-bit number scaled by 2^{14}, that is 16384 corresponds to 1.0. This coefficient is used to compute a correction for non-orthogonality between sin and cos signals.

The encoder sin, cos, and phase gains are signed 16-bit numbers scaled by 2^{14}, that is 16384 corresponds to 1.0. The sin gain multiplies the raw sin input after the offset and bias are added, the cos gain multiplies the raw cos input after the offset and bias are added. The phase gain is used to compute a correction for non-orthogonality between sin and cos signals.

The encoder sin/cos interpolation factor controls the scaling and precision of the analog interpolation between digital quadrature positions. This coefficient is an unsigned 16-bit number with units of counts/electrical revolution.The minimum value is 4, or no interpolation, the maximum value is 16384. Currently only interpolation by a factor of 2 is supported; if the specified interpolation factor is not a factor of 2 it will be rounded down.

The N-series ION supports a 32-entry table for fine correction of the angle computed using sin and cos analog readings. This table is enabled by setting the encoder sin/cos correction table enable to 1, the default value of 0 means disabled. The table should be disabled whenever entries are being written.

Each table entry may be written using the encoder sin/cos correction table coefficient codes, 0x310 means entry 0, 0x311 means entry 1, and so forth. Each table entry is a signed 16-bit number scaled by 360°/16384 that is added to the raw angle computed from the sin and cos analog inputs. Linear interpolation is used to compute the actual value added.

Restrictions

This command is not supported by all Magellan products. MC58113 and N-series ION support the leg current offsets. N-series ION supports the sin/cos encoder calibration coefficients.

C-Motion API

```
PMDresult PMDSetAnalogCalibration(PMDAxisInterface axis_intf,
                  PMDuint16 channel,
                  PMDint16 offset);
PMDresult PMDGetAnalogCalibration(PMDAxisInterface axis_intf,
                  PMDuint16 channel,
                  PMDint16 *offset);
```

VB-Motion API

```
MagellanAxis.AnalogCalibrationSet( [in] channel, [in] offset )
MagellanAxis.AnalogCalibrationGet( [in] channel, [out] offset )
```

see

ReadAnalog (p. 71), **CalibrateAnalog** (p. 17)

SetAuxiliaryEncoderSource 08h
GetAuxiliaryEncoderSource 09h

Syntax

SetAuxiliaryEncoderSource *axis mode_auxiliaryAxis*
GetAuxiliaryEncoderSource *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*None*	0
	Quadrature	1
	Pulse & Direction	2
auxiliaryAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet Structure

SetAuxiliaryEncoderSource

0	*axis*	**08**h
15 12	11 8	7 0

Data

write

0	*mode*	0	*auxiliaryAxis*
15 9	8	7 2	1 0

GetAuxiliaryEncoderSource

0	*axis*	**09**h
15 12	11 8	7 0

Data

read

0	*mode*	0	*auxiliaryAxis*
15 9	8	7 2	1 0

Description

SetAuxiliaryEncoderSource controls the motion control IC's dual encoder loop feature. The *mode* either disables or specifies the format for the secondary encoder loop for *axis*. The *auxilaryAxis* selects which axis encoder input is to be interpreted as the damping term (Kd) of the servo equation for *axis*. To determine the actual position of the auxiliary encoder, use **GetActualPosition**(*auxiliaryAxis*). The auxiliary axis encoder input is used for commutation of brushless DC motors. The **SetPhaseOffset**, **SetPhaseAngle**, and **SetPhaseCounts** commands should still be applied to the main axis, even when dual encoder loop is enabled.

Restrictions

To avoid a potentially unstable operating condition in dual loop mode, the auxiliary encoder should have resolution greater than or equal to that of the main encoder. Not all products support pulse & direction input.

C-Motion API

```
PMDresult PMDSetAuxiliaryEncoderSource(PMDAxisInterface axis_intf,
                                       PMDuint8 mode,
                                       PMDAxis auxiliaryAxis)
PMDresult PMDGetAuxiliaryEncoderSource(PMDAxisInterface axis_intf,
                                       PMDuint8* mode,
                                       PMDAxis* auxiliaryAxis)
```

VB-Motion API

```
MagellanAxis.AuxiliaryEncoderSourceSet( [in] mode, [in] auxiliaryAxis )
MagellanAxis.AuxiliaryEncoderSourceGet( [out] mode, [out] auxiliaryAxis )
```

SetAxisOutMask 45h
GetAxisOutMask 46h

Syntax

SetAxisOutMask *axis sourceAxis_sourceRegister selectionMask senseMask*
GetAxisOutMask *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
sourceAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
sourceRegister	*Disabled*	0
	Event Status	1
	Activity Status	2
	Signal Status	3
	Drive Status	4
selectionMask	see below	bitmask
senseMask	see below	bitmask

Packet Structure

SetAxisOutMask

0	*axis*	**45**h
15 12	11 8	7 0

First Data Word

write	0	*sourceRegister*	*sourceAxis*
	15 12	11 8	7 0

Second Data Word

write	*selectionMask*
	15 0

Third Data Word

write	*senseMask*
	15 0

GetAxisOutMask

0	*axis*	**46**h
15 12	11 8	7 0

First Data Word

read	0	*sourceRegister*	*sourceAxis*
	15 12	11 8	7 0

Second Data Word

read	*selectionMask*
	15 0

Third Data Word

read	*senseMask*
	15 0

Description

SetAxisOutMask configures what will drive the AxisOut pin of the ***axis***. The ***sourceRegister*** and ***sourceAxis*** arguments specify which register, from which axis, will be used to drive AxisOut of the specified axis.

SetAxisOutMask (cont.) 45h
GetAxisOutMask 46h

Description (cont.)

For each bit in the *selectionMask* that is set to 1, the corresponding bit of the specified *sourceRegister* is selected to set AxisOut active. The *senseMask* bit determines which state of each bit causes AxisOut to be active—a zero (0) in the *senseMask* means that a 0 in the corresponding bit will cause AxisOut to be active, and a 1 in the *senseMask* means that a 1 in the corresponding bit will cause AxisOut to be active. If multiple bits are selected in the *sourceRegister*, AxisOut will be active if any of the selected bits, combined with their sense, require it to be. The following table shows the available bits in each register.

Bit	Event Status Register	Activity Status Register	Signal Status Register	Drive Status Register
0	Motion Complete	Phasing Initialized	Encoder A	
1	Wrap-around	At Maximum Velocity	Encoder B	In Foldback
2	Breakpoint 1	Tracking	Encoder Index	Overtemperature
3	Position Capture		Capture Input	
4	Motion Error		Positive Limit	In Holding
5	Positive Limit		Negative Limit	Overvoltage
6	Negative Limit		AxisIn	Undervoltage
7	Instruction Error	Axis Settled	Hall Sensor A	
8	Disable	Motor Mode	Hall Sensor B	
9	Overtemperature Fault	Position Capture	Hall Sensor C	
0Ah	Bus Voltage Fault	In Motion		
0Bh	Commutation Error	In Positive Limit		
0Ch	Current Foldback	In Negative Limit		
0Dh			/Enable Input	
0Eh	Breakpoint 2		FaultOut	
0Fh				

For example, assume it is desired to have the AxisOut pin of *Axis1* driven active whenever motion complete of *Axis2* is 1, or commutation error of *Axis2* is 0. In this case, *axis* would be 0 (*Axis1*), *sourceAxis* would be 1 (*Axis2*), *sourceRegister* would be 1 (*Event Status*), *selectionMask* would be 0801h (commutation error and motion complete) and *senseMask* would be 0001h.

When the source register is set to Disabled, AxisOut will be active.

GetAxisOutMask returns the mapping of the AxisOut pin of *axis*.

Restrictions

Depending on the product features, some bits may not be supported. See the product user guide.

C-Motion API

```
PMDresult PMDSetAxisOutMask(PMDAxisInterface axis_intf,
                            PMDAxis sourceAxis,
                            PMDuint8 sourceRegister,
                            PMDuint16 selectionMask,
                            PMDuint16 senseMask)
PMDresult PMDGetAxisOutMask(PMDAxisInterface axis_intf,
                            PMDAxis* sourceAxis,
                            PMDuint8* sourceRegister,
                            PMDuint16* selectionMask,
                            PMDuint16* senseMask)
```

VB-Motion API

```
MagellanAxis.AxisOutMaskSet( [in] sourceAxis,
                             [in] sourceRegister,
                             [in] selectionMask,
                             [in] senseMask )
MagellanAxis.AxisOutMaskGet( [out] sourceAxis,
                             [out] sourceRegister,
                             [out] selectionMask,
                             [out] senseMask )
```

see

SetBreakpoint D4h
GetBreakpoint D5h

Syntax

SetBreakpoint *axis breakpointID sourceAxis_action_trigger*
GetBreakpoint *axis breakpointID*

Motor Types

DC Brush	**Brushless DC**	**Microstepping**	**Pulse & Direction**

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
breakpointID	*Breakpoint1*	0
	Breakpoint2	1
sourceAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
action	*None*	0
	Update	1
	Abrupt Stop	2
	Smooth Stop	3
	— (Reserved)	4
	Disable Position Loop & Higher Modules	5
	Disable Current Loop & Higher Modules	6
	Disable Motor Output & Higher Modules	7
	Abrupt Stop with Position Error Clear	8
trigger	*None*	0
	Greater Or Equal Commanded Position	1
	Lesser Or Equal Commanded Position	2
	Greater Or Equal Actual Position	3
	Lesser Or Equal Actual Position	4
	Commanded Position Crossed	5
	Actual Position Crossed	6
	Time	7
	Event Status	8
	Activity Status	9
	Signal Status	Ah
	Drive Status	Bh

Packet Structure

SetBreakpoint

0	*axis*	**D4**h
15 12	11 8	7 0

First data word

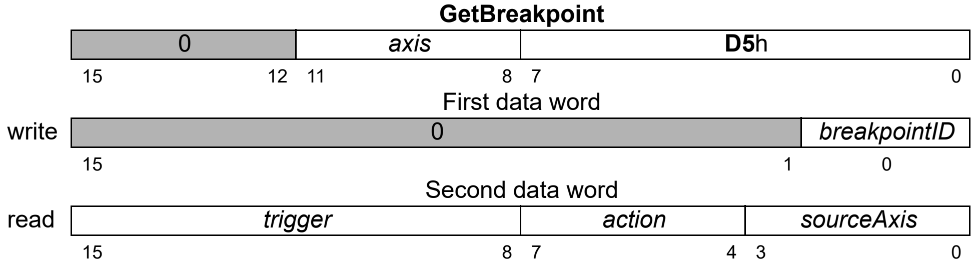
write	0	*breakpointID*
	15 1	0

Second data word

write	*trigger*	*action*	*sourceAxis*
	15 8	7 4	3 0

SetBreakpoint (cont.) D4h
GetBreakpoint D5h

Packet Structure (cont.)



Description

SetBreakpoint establishes a breakpoint for the specified *axis* to be triggered by a condition or event on *sourceAxis*, which may be the same as or different from *axis*. Up to two concurrent breakpoints can be set for each axis, each of which may have its own breakpoint type and comparison value. The *breakpointID* field specifies which breakpoint the **SetBreakpoint** and **GetBreakpoint** commands will address.

The six position breakpoints are threshold-triggered; the breakpoint occurs when the indicated value reaches or crosses a threshold. The status breakpoints are level-triggered; the breakpoint occurs when a specific bit or combination of bits in the indicated status register changes state. The time breakpoint is triggered when the current time, which may be read using GetTime, is equal to the breakpoint value. Thresholds and bit specifications are both set by the **SetBreakpointValue** instruction.

The *action* determines what the motion control IC does when the breakpoint occurs, as follows:

Action	Description
None	No action
Update	Transfer the double buffered registers specified by the Breakpoint Update Mask into the active registers.
Abrupt Stop	Causes an instantaneous halt of the trajectory generator. Trajectory velocity is zeroed.
Smooth Stop	Causes a smooth stop to occur at the active deceleration rate.
Abrupt Stop with Position Error Clear	Abrupt stop of the trajectory, and additionally zero the position error to the servo.
Disable Position Loop & Higher Modules	Disables Trajectory generator and position loop modules.
Disable Current Loop & Higher Modules	Disables Trajectory generator, position loop, and current loop modules.
Disable Motor Output & Higher Modules	Disables all modules, including the motor output.

GetBreakpoint returns the *trigger*, *action*, and *sourceAxis* for the specified breakpoint (1 or 2) of the indicated *axis*. When a breakpoint occurs, the trigger value will be reset to zero (0). The *Commanded Position Crossed* and the *Actual Position Crossed* triggers are converted to one of the position trigger types 1–4, depending on the current position when the command is issued.

Restrictions

Always load the breakpoint comparison value (**SetBreakpointValue** command) before setting a new breakpoint condition (**SetBreakpoint** command). Failure to do so will likely result in unexpected behavior.

Breakpoint trigger options may be limited depending on the resources of the *sourceAxis*. See the product user guide.

SetBreakpoint (cont.) D4h
GetBreakpoint D5h

C-Motion API

```
PMDresult PMDSetBreakpoint(PMDAxisInterface axis_intf,
                           PMDuint16 breakpointID, PMDAxis sourceAxis,
                           PMDuint8 action, PMDuint8 trigger)
PMDresult PMDGetBreakpoint(PMDAxisInterface axis_intf,
                           PMDuint16 breakpointID, PMDAxis* sourceAxis,
                           PMDuint8* action, PMDuint8* trigger)
```

VB-Motion API

```
MagellanAxis.BreakpointSet( [in] breakpointID,
                            [in] sourceAxis,
                            [in] action,
                            [in] trigger )
MagellanAxis.BreakpointGet( [in] breakpointID,
                            [out] sourceAxis,
                            [out] action,
                            [out] trigger )
```

see **Set/GetBreakpointValue** (p. 99), **Set/GetBreakpointUpdateMask** (p. 97)

SetBreakpointUpdateMask 32h
GetBreakpointUpdateMask 33h

Syntax

SetBreakpointUpdateMask *axis breakPointID mask*
GetBreakpointUpdateMask *axis breakPointID*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
breakpointID	*Breakpoint1*	0
	Breakpoint2	1
	Type	**Scaling**
mask	unsigned 16 bit	bitmask

Packet Structure

SetBreakpointUpdateMask

	15–12	11–8	7–0
	0	*axis*	**32**h

First data word

	15–1	0
write	0	*breakpointID*

Second data word

	15–0
write	*mask*

GetBreakpointUpdateMask

	15–12	11–8	7–0
	0	*axis*	**33**h

First data word

	15–1	0
write	0	*breakpointID*

Second data word

	15–0
read	*mask*

Description

SetBreakpointUpdateMask configures what loops are updated upon the update action of a breakpoint. If the bitmask for a given loop is set in the *mask*, the operating parameters for that loop will be updated from the buffered values when the breakpoint is hit, and update is the breakpoint action. Each breakpoint has its own update mask. The bitmask encoding is given below.

Name	Bit(s)	Description
Trajectory	0	Set to 1 to update trajectory from buffered parameters.
Position Loop	1	Set to 1 to update position loop from buffered parameters.
—	2	Reserved
Current Loop	3	Set to 1 to update current loop from buffered parameters.
—	4–15	Reserved

For example, if the update mask for breakpoint 1 is set to hexadecimal 0001h, and the action for breakpoint 1 is set to update, the trajectory for the given *axis* will be updated from its buffered parameters when breakpoint 1 is hit.

SetBreakpointUpdateMask (cont.) 32h
GetBreakpointUpdateMask 33h

Description (cont.)

The Current Loop bit applies regardless of the active current control mode. When it is set, a breakpoint action of update will update either the active FOC parameters or the active digital current loop parameters, depending on which Current Control mode is active.

GetBreakpointUpdateMask gets the update mask for the indicated breakpoint.

Restrictions

The Current Loop bit is only valid for products that include a current loop.

C-Motion API

```
PMDresult PMDSetBreakpointUpdateMask(PMDAxisInterface axis_intf,
                                     PMDuint16 breakPointID,
                                     PMDuint16 mask)
PMDresult PMDGetBreakpointUpdateMask(PMDAxisInterface axis_intf,
                                     PMDuint16 breakPointID,
                                     PMDuint16* mask)
```

VB-Motion API

```
Dim mask as Short
MagellanAxis.BreakpointUpdateMask( breakpointID ) = mask
mask = MagellanAxis.BreakpointUpdateMask( breakpointID )
```

see

Set/GetBreakpoint (p. 94), **Set/GetUpdateMask** (p. 211)

SetBreakpointValue D6h
GetBreakpointValue D7h

Syntax

SetBreakpointValue *axis breakpointID value*
GetBreakpointValue *axis breakpointID*

Motor Types

DC Brush	**Brushless DC**	**Microstepping**	**Pulse & Direction**

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
breakpointID	*Breakpoint1*	0
	Breakpoint2	1

value (see below)

Packet Structure

SetBreakpointValue

0	*axis*	**D6**h
15 12	11 8	7 0

First data word

write

0	*breakpointID*
15 1	0

Second data word

write

value (high-order part)
31 16

Third data word

write

value (low-order part)
15 0

GetBreakpointValue

0	*axis*	**D7**h
15 12	11 8	7 0

First data word

write

0	*breakpointID*
15 1	0

Second data word

read

value (high-order part)
31 16

Third data word

read

value (low-order part)
15 0

Description

SetBreakpointValue sets the breakpoint comparison value for the specified *axis*. For the position breakpoints, this is a threshold comparison value. For the time breakpoint it is an equality comparison value.

SetBreakpointValue (cont.) D6h
GetBreakpointValue D7h

Description (cont.)

The *value* parameter is interpreted according to the trigger condition for the selected breakpoint; see **SetBreakpoint** (p. 94). The data format for each trigger condition is as follows:

Breakpoint Trigger	Value Type	Range	Units
Greater Or Equal Commanded Position	signed 32-bit	-2^{31} to $2^{31}-1$	counts
Lesser Or Equal Commanded Position	signed 32-bit	-2^{31} to $2^{31}-1$	counts
Greater Or Equal Actual Position	signed 32-bit	-2^{31} to $2^{31}-1$	counts
Lesser Or Equal Actual Position	signed 32-bit	-2^{31} to $2^{31}-1$	counts
Commanded Position Crossed	signed 32-bit	-2^{31} to $2^{31}-1$	counts
Actual Position Crossed	signed 32-bit	-2^{31} to $2^{31}-1$	counts
Time	unsigned 32-bit	0 to $2^{32}-1$	cycles
Event Status	2 word mask	-	boolean status values
Activity Status	2 word mask	-	boolean status values
Signal Status	2 word mask	-	boolean status values
Drive Status	2 word mask	-	boolean status values

For level-triggered breakpoints, the high-order part of *value* is the selection mask, and the low-order word is the sense mask. For each selection bit that is set to 1, the corresponding bit of the specified status register is conditioned to cause a breakpoint when it changes state. The sense mask bit determines which state causes the break. If it is 1, the corresponding status register bit will cause a break when it is set to 1. If it is 0, the status register bit will cause a break when it is set to 0.

For example, assume it is desired that the breakpoint type will be set to Event Status and that a breakpoint should be recognized whenever the motion complete bit (bit 0 of Event Status register) is set to 1, or the commutation error bit (bit 11 of Event Status register) is set to 0. In this situation the high and low words for value would be high word: 0801h and low word: 0001h.

GetBreakpointValue returns the breakpoint value for the specified *breakpointID*.

Two completely separate breakpoints are supported, each of which may have its own breakpoint type and comparison value. The *breakpointID* field specifies which breakpoint the **SetBreakpointValue** and **GetBreakpointValue** commands will address.

Restrictions

Always load the breakpoint comparison value (**SetBreakpointValue** command) before setting a new breakpoint condition (**SetBreakpoint** command). Failure to do so will likely result in unexpected behavior.

Depending on the product features, not all bits of all registers are supported. See the product user guide.

C-Motion API

```
PMDresult PMDSetBreakpointValue(PMDAxisInterface axis_intf,
                                PMDuint16 breakpointID,
                                PMDint32 value)
PMDresult PMDGetBreakpointValue(PMDAxisInterface axis_intf,
                                PMDuint16 breakpointID,
                                PMDint32* value)
```

VB-Motion API

```
MagellanAxis.BreakpointValueSet( [in] breakpointID, [in] value )
MagellanAxis.BreakpointValueGet( [in] breakpointID, [out] value )
```

see

Set/GetBreakpoint (p. 94)

SetBufferLength C2h
GetBufferLength C3h

Syntax

SetBufferLength *bufferID length*
GetBufferLength *bufferID*

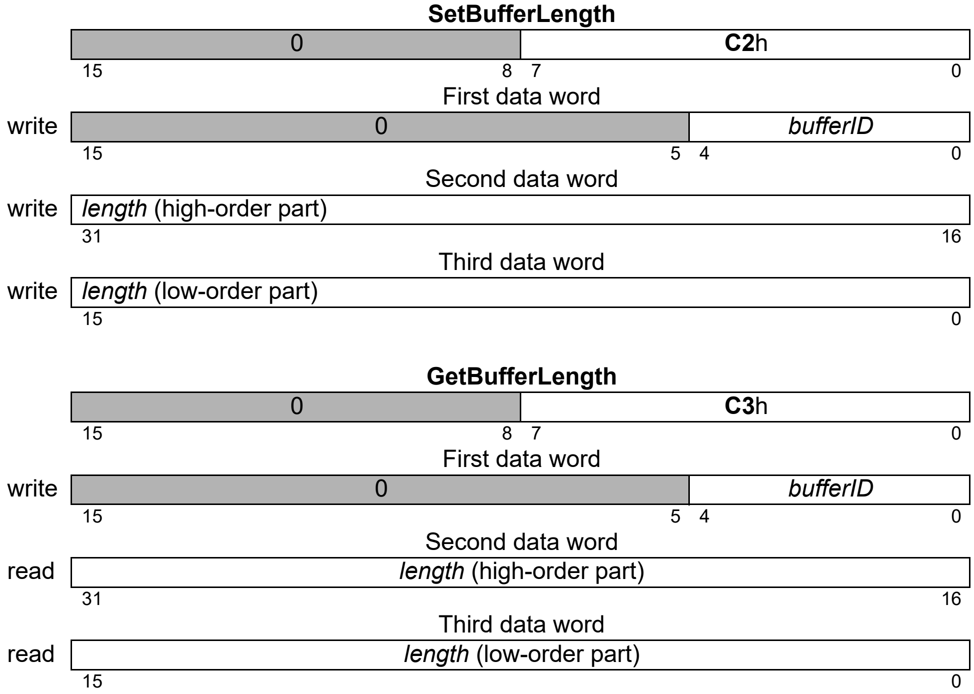
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range
bufferID	unsigned 16 bits	0 *to* 31
length	unsigned 32 bits	1 *to* $2^{30} - 1$

Packet Structure



Description

SetBufferLength sets the *length*, in numbers of 32-bit elements, of the buffer in the memory block identified by *bufferID*. For buffers pointing to non-volatile RAM, the length should be specified in 16-bit words.

Note: The **SetBufferLength** command resets the buffers read and write indexes to 0.

The **GetBufferLength** command returns the *length* of the specified buffer.

Restrictions

The buffer length plus the buffer start address cannot exceed the memory size of the product. See the product user guide.

When the buffer start is changed in such a way that the word size changes, the buffer length will change. An error may result if the new buffer start plus the new buffer length is outside the legal range. If the current state of the buffer is not known it is safer to set the buffer length to zero before changing the buffer start.

C-Motion API

```
PMDresult PMDSetBufferLength(PMDAxisInterface axis_intf,
                             PMDuint16 bufferID, PMDuint32 length)
PMDresult PMDGetBufferLength(PMDAxisInterface axis_intf,
                             PMDuint16 bufferID, PMDuint32* length)
```

SetBufferLength (cont.) C2h
GetBufferLength C3h

VB-Motion API

```
Dim length as Long
MagellanObject.BufferLength( bufferID ) = length
length = MagellanObject.BufferLength( bufferID )
```

see **Set/GetBufferReadIndex** (p. 103), **Set/GetBufferStart** (p. 104), **Set/GetBufferWriteIndex** (p. 106)

SetBufferReadIndex C6h
GetBufferReadIndex C7h

Syntax

SetBufferReadIndex *bufferID index*
GetBufferReadIndex *bufferID*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range	Scaling	Units
bufferID	unsigned 16 bits	0 *to* 31	unity	-
index	unsigned 32 bits	0 *to* buffer length - 1	unity	double words

Packet Structure

SetBufferReadIndex

	0	**C6h**
	15 8	7 0

First data word

write	0	*bufferID*
	15 5	4 0

Second data word

write	*index* (high-order part)
	31 16

Third data word

write	*index* (low-order part)
	15 0

GetBufferReadIndex

	0	**C7h**
	15 8	7 0

First data word

write	0	*bufferID*
	15 5	4 0

Second data word

read	*index* (high-order part)
	31 16

Third data word

read	*index* (low-order part)
	15 0

Description

SetBufferReadIndex sets the address of the read *index* for the specified **bufferID**. For buffers pointing to non-volatile RAM, the read index should be specified in 16-bit words.

GetBufferReadIndex returns the current read *index* for the specified **bufferID**.

Restrictions

If the read index is set to an address beyond the length of the buffer, the command will not be executed and will return host I/O error code 7, buffer bound exceeded.

C-Motion API

```
PMDresult PMDSetBufferReadIndex(PMDAxisInterface axis_intf,
                                PMDuint16 bufferID,
                                PMDuint32 index)
PMDresult PMDGetBufferReadIndex(PMDAxisInterface axis_intf,
                                PMDuint16 bufferID,
                                PMDuint32* index)
```

VB-Motion API

```
Dim index as Long
MagellanObject.BufferReadIndex( bufferID ) = index
index = MagellanObject.BufferReadIndex( bufferID )
```

see

Set/GetBufferLength (p. 101), **Set/GetBufferStart** (p. 104), **Set/GetBufferWriteInde**x (p. 106)

SetBufferStart C0h
GetBufferStart C1h

Syntax

SetBufferStart *bufferID address*
GetBufferStart *bufferID*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range	Units
bufferID	unsigned 16 bits	0 *to* 31	-
address	unsigned 32 bits	0 *to* $2^{31} - 1$	double words

Packet Structure

SetBufferStart

	0	**C0**h
	15 8	7 0
	First data word	
write	0	*bufferID*
	15 5	4 0
	Second data word	
write	*address* (high-order part)	
	31 16	
	Third data word	
write	*address* (low-order part)	
	15 0	

GetBufferStart

	0	**C1**h
	15 8	7 0
	First data word	
write	0	*bufferID*
	15 5	4 0
	Second data word	
read	*address* (high-order part)	
	31 16	
	Third data word	
read	*address* (low-order part)	
	15 0	

Description

SetBufferStart sets the starting *address* for the specified buffer, in double-words, of the buffer in the memory block identified by ***bufferID.*** In products with non-volatile RAM (NVRAM), the address range beginning at 20000000h is used for NVRAM. Buffers pointing to NVRAM use a word size of 16 bits, unlike buffers pointing to DRAM, which use a word size of 32 bits. For NVRAM buffers the start should be specified in 16-bit words pluse 20000000h.

Note: The **SetBufferStart** command resets the buffers read and write indexes to 0.

The **GetBufferStart** command returns the starting *address* for the specified *bufferID*.

Restrictions

The buffer start address plus the buffer length cannot exceed the memory size of the product. See the product user guide.

When the buffer start is changed in such a way that the word size changes, the buffer length will change. An error may result if the new buffer start plus the new buffer length is outside the legal range. If the current state of the buffer is not known it is safer to set the buffer length to zero before changing the buffer start.

SetBufferStart (cont.) C0h
GetBufferStart C1h

C-Motion API

```
PMDresult PMDSetBufferStart(PMDAxisInterface axis_intf, PMDuint16 bufferID,
                            PMDuint32 address)
PMDresult PMDGetBufferStart(PMDAxisInterface axis_intf, PMDuint16 bufferID,
                            PMDuint32* address)
```

VB-Motion API

```
Dim address as Long
MagellanObject.BufferStart( bufferID ) = address
address = MagellanObject.BufferStart( bufferID )
```

see **Set/GetBufferLength** (p. 101), **Set/GetBufferReadIndex** (p. 103), **Set/GetBufferWriteIndex** (p. 106)

SetBufferWriteIndex C4h
GetBufferWriteIndex C5h

Syntax

SetBufferWriteIndex *bufferID index*
GetBufferWriteIndex *bufferID*

Motor Types

DC Brush	**Brushless DC**	**Microstepping**	**Pulse & Direction**

Arguments

Name	**Type**	**Range**	**Scaling**	**Units**
bufferID	unsigned 16 bits	0 *to* 31	unity	-
index	unsigned 32 bits	0 *to* buffer length - 1	unity	double words

Packet Structure

SetBufferWriteIndex

0	**C4**h
15 … 8	7 … 0

First data word

	0	*bufferID*
write	15 … 4	3 … 0

Second data word

	index (high-order part)
write	31 … 16

Third data word

	index (low-order part)
write	15 … 0

GetBufferWriteIndex

0	**C5**h
15 … 8	7 … 0

First data word

	0	*bufferID*
write	15 … 4	3 … 0

Second data word

	index (high-order part)
read	31 … 16

Third data word

	index (low-order part)
read	15 … 0

Description

SetBufferWriteIndex sets the write *index* for the specified *bufferID*. For buffers pointing to non-volatile RAM, the write index should be specified in 16-bit words.

GetBufferWriteIndex returns the write *index* for the specified *bufferID*.

Restrictions

C-Motion API

```
PMDresult PMDSetBufferWriteIndex(PMDAxisInterface axis_intf,
                                 PMDuint16 bufferID, PMDuint32 index);
PMDresult PMDGetBufferWriteIndex(PMDAxisInterface axis_intf,
                                 PMDuint16 bufferID, PMDuint32* index)
```

VB-Motion API

```
Dim index as Long
MagellanObject.BufferWriteIndex( bufferID ) = index
index = MagellanObject.BufferWriteIndex( bufferID )
```

see

Set/GetBufferLength (p. 101), **Set/GetBufferReadIndex** (p. 103), **Set/GetBufferStart** (p. 104)

SetCANMode 12h
GetCANMode 15h

Syntax

SetCANMode *mode*
GetCANMode

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Encoding
mode	unsigned 16 bits	see below

Packet Structure

SetCANMode

	15–8	7–0
	0	**12**h

Data

	15–13	12–7	6–0
write	*transmission rate*	0	*nodeID*

GetCANMode

	15–8	7–0
	0	**15**h

Data

	15–13	12–7	6–0
read	*transmission rate*	0	*nodeID*

Description

SetCANMode sets the CAN 2.0B communication parameters for the motion control IC. After completion of this command, the motion control IC will respond to a CAN receive message addressed to 600h + nodeID. CAN responses are sent to 580h + nodeID. The CAN transmission rate will be as specified in the ***transmission rate*** parameter. Note that when this command is used to change to a new nodeID, the command response (for this command) will be sent to the new nodeID. The following table shows the encoding of the data used by this command.

Bits	Name	Instance	Encoding
0–6	CAN NodeID	*Address 0*	0
		Address 1	1
		…	…
		Address 127	127
7–12	— (Reserved)		
13–15	Transmission Rate	*1,000,000 baud*	0
		800,000	1
		500,000	2
		250,000	3
		125,000	4
		50,000	5
		20,000	6
		10,000	7

Restrictions

C-Motion API

```
PMDresult PMDSetCANMode(PMDAxisHandle axis_handle, PMDuint8 nodeID,
                        PMDuint8 transmission_rate)
PMDresult PMDGetCANMode(PMDAxisHandle axis_handle, PMDuint8* nodeID,
                        PMDuint8* transmission_rate)
```

VB-Motion API

```
CommunicationCAN.CANModeSet( [ in ] nodeID, [ in ] transmission_rate )
```

see

SetCaptureSource D8h
GetCaptureSource D9h

Syntax

SetCaptureSource *axis source*
GetCaptureSource *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
source	*Index*	0
	Home	1
	High Speed Capture	2

Packet Structure

SetCaptureSource

0	*axis*	**D8**h
15 12	11 8	7 0

Data

write	0	*source*
	15 3	2 0

GetCaptureSource

0	*axis*	**D9**h
15 12	11 8	7 0

Data

read	0	*source*
	15 3	2 0

Description

SetCaptureSource determines which of three signals—*Index*, *Home*, or *High Speed Capture*—is used to trigger the capture of the actual axis position for the specified *axis*. **GetCaptureSource** returns the capture signal *source* for the selected *axis*.

Restrictions

High Speed Capture is not available as a capture source in all products. See the product user guide.

C-Motion API

```
PMDresult PMDSetCaptureSource(PMDAxisInterface axis_intf,
                              PMDuint16 source)
PMDresult PMDGetCaptureSource(PMDAxisInterface axis_intf,
                              PMDuint16* source)
```

VB-Motion API

```
Dim source as Short
MagellanAxis.CaptureSource = source
source = MagellanAxis.CaptureSource
```

see

GetCaptureValue (p. 29)

SetCommutationMode E2h
GetCommutationMode E3h

Syntax

SetCommutationMode *axis mode*
GetCommutationMode *axis*

Motor Types

	Brushless DC		

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Sinusoidal*	0
	Hall-based	1

Packet Structure

SetCommutationMode

0	*axis*	**E2**h
15 12	11 8	7 0

Data

write	0	*mode*
	31 1	0

GetCommutationMode

0	*axis*	**E3**h
15 12	11 8	7 0

Data

read	0	*mode*
	31 1	0

Description

SetCommutationMode sets the phase commutation *mode* for the specified *axis*.

When set to *Sinusoidal*, as the motor turns, the encoder input signals are used to calculate the phase angle. This angle is in turn used to generate sinusoidally varying outputs to each motor winding.

When set to *Hall-based,* the Hall effect sensor inputs are used to commutate the motor windings using a "six-step" or "trapezoidal" waveform method.

When using FOC current control, this command is used to define the method used for motor phase determination.

GetCommutationMode returns the value of the commutation mode.

Restrictions

C-Motion API

```
PMDresult PMDSetCommutationMode(PMDAxisInterface axis_intf,
                                PMDuint16 mode)
PMDresult PMDGetCommutationMode(PMDAxisInterface axis_intf,
                                PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.CommutationMode = mode
mode = MagellanAxis.CommutationMode
```

see

Set/GetPhasePrescale (p. 171), **Set/GetPhaseCounts** (p. 164)

SetCommutationParameter 63h
GetCommutationParameter 64h

Syntax

SetCommutationParameter *axis parameter value*
GetCommutationParameter *axis parameter value*

Motor Types

	Brushless DC	Microstepping

Arguments

Name	Instance	Encoding
axis	Axis1	0
parameter	phase counts	0
	phase angle	1
	phase offset	2
	phase denominator3	

	Type	Range	Scaling/Units
value	unsigned 32-bits	0 to 2^{31}-1	counts

Packet Structure

SetCommutationParameter

	15 14 13 12	11 10 9 8	7 6 5 4 3 2 1 0
	0	*axis*	33h
write	*parameter* (15–0)		
write	*value* (high-order part) (15–0)		
write	*value* (low-order part) (15–0)		

GetCommutationParameter

	15 12	11 8	7 0
	0	*axis*	64h
write	*parameter* (15–0)		
read	*value* (high-order part) (15–0)		
read	*value* (low-order part) (15–0)		

Description

SetCommutationParameter is used to set several 32-bit quantities used for motor commutation or microstep generation.

For brushless DC motors, the PhaseCounts and PhaseDenominator registers specify the number of encoder counts per electrical revolution. If this number is an integer, PhaseDenominator may be left at its default value of 1, and PhaseCounts set to the counts per electrical revolution. Alternatively, PhaseDenominator may be set to the number of motor pole pairs, and PhaseCounts to the number of encoder counts per mechanical revolution.

For example, for a six pole motor using an encoder with 1024 counts per revolution there are 341 1/3 encoder counts per electrical revolution, PhaseCounts may be set to 1024, and PhaseDenominator to 3.

PhaseAngle and PhaseOffset are both values that may be set by command but are normally altered by the commutation process. PhaseAngle gives the current position in the electrical cycle; to convert to degrees divide PhaseAngle by PhaseCounts and multiply by 360. For example, for the motor in the example above, a PhaseAngle of 256 corresponds to an angle of (256/1024)*360 = 90 degrees.

PhaseOffset is the non-negative offset from the index mark to the internal zero phase angle. Setting PhaseOffset has no immediate effect, but, if phase correction is enabled, sets the phase angle when an index pulse is detected. The default value of PhaseOffset is -1, which means that at the first index pulse the PhaseOffset should be set equal to the current phase angle. If phase initialization is correctly set up it is normally not necessary to set PhaseOffset.PhaseOffset may be read to determine whether an index pulse has been detected since phase initialization.

Setting the PhaseAngle has the side-effect of setting PhaseOffset to the default value of -1.

The maximum value for PhaseOffset is 2^{31}- 1, any value with bit 31 set is interpreted as negative, and equivalent to -1. If set by command PhaseOffset should be less than PhaseCounts, but that condition is not checked.

For microstep motors PhaseCounts sets the number of microsteps per electrical revolution, and PhaseAngle the current position in the electrical cycle. Each electrical revolution is four full steps. The maximum supported value is 1024 microsteps per electrical revolution, or 256 microsteps per full step. The PhaseDenominator parameter is ignored for microstep motors.

For microstep motors PhaseOffset, which is zero by default, specifies an offset to be added to PhaseAngle to produce the current electrical phase angle. 08000h corresponds to 360 degrees for PhaseOffset.

To obtain traditional full-stepping both phases are always driven at full output, either positive or negative, set PhaseCounts to 4, and set Offset to 01000h or 45 degrees.

The minimum value for PhaseCounts, for either step or BLDC motors, is 4. The minimum value for PhaseDenominator is 1, and the maximum possible value is 32767. For proper commutation PhaseCounts must be greater than PhaseDenominator, although that condition is not checked.

Restrictions

Not all Magellan products support these commands, or any 32-bit interface to the commutation parameters. The older 16-bit interface uses the commands **SetPhaseCounts**, **SetPhaseAngle**, **SetPhaseOffset**, and **SetPhasePrescale**. Products supporting the 32-bit interface may not support **SetPhasePrescale**.

It is possible to specify commutation parameters using the 32-bit interface that may not be represented using the 16-bit interface. In this case, if a 16-bit get command is invoked then a value representation error (37) will be raised. It is recommended that the 16-bit and 32-bit interfaces not be used together.

Errors

Invalid Parameter: Unrecognized parameter or value out of bounds.

C-Motion API

```
PMDresult PMDGetCommutationParameter (PMDAxisInterface axis_intf,
                                      PMDuint16 parameter,
                                      PMDint32* value);
PMDresult PMDSetCommutationParameter (PMDAxisInterface axis_intf,
                                      PMDuint16 parameter,
                                      PMDint32 value);
```

Script API

```
GetCommutationParameter parameter
SetCommutationParameter paramter value
```

C# API

```
Int32 value = PMDAxis.GetCommutationParameter(PMDCommutationParameter
                                              parameter);
PMDAxis.SetCommutationParameter(PMDCommutationParameter parameter,
                                Int32 value);
```

Visual Basic API

```
Int32 value = PMDAxis.GetCommutationParameter(ByVal parameter
                                              As PMDCommutationParame-
ter)
PMDAxis.SetCommutationParameter(ByVal parameter
                                As PMDCommutationParameter,
                                ByVal value As Int32)
```

see

Set/GetPhaseAngle (p. 161), **Set/GetPhaseCorrectionMode** (p. 163), **Set/GetPhaseCounts** (p. 164), **Set/GetPhaseOffset** (p. 168), **Set/GetPhasePrescale** (p. 171)

SetCurrent 5Eh
GetCurrent 5Fh

Syntax

SetCurrent *axis parameter value*
GetCurrent *axis parameter*

Motor Types

		Microstepping	**Pulse & Direction**

Arguments

Name	**Instance**	**Encoding**	**Units**
axis	*Axis1*	0	
	Axis2	1	
	Axis3	2	
	Axis4	3	
parameter	*Holding Motor Limit*	0	%
	Holding Delay	1	trajectory generator cycles
	Drive Current	2	%

	Type	**Range/Scaling**
value	unsigned 16-bit	see below

Packet Structure

SetCurrent

0	*axis*	**5E**h
15 12	11 8	7 0

First Data Word

write	*parameter*
	15 0

Second Data Word

write	*value*
	15 0

GetCurrent

0	*axis*	**5F**h
15 12	11 8	7 0

First Data Word

write	*parameter*
	15 0

Second Data Word

read	*value*
	15 0

Description

SetCurrent configures some aspects of step motor current control. The Holding Motor Limit is the maximum commanded current when in holding. The Holding Delay is the number of cycles to wait after end of move before going into holding. The Drive Current is the commanded current when not in holding.

The ***Holding Motor Limit*** is in units of % maximum current, with scaling of $100/2^{15}$. Its range is 0 to 2^{15}–1. It defines the value to which the current will be limited when in the holding state. This limit is applied as an additional limit to the motor limit, so the lower of the two will affect the true limit.

The ***Holding Delay*** is in units of trajectory generator cycles, with unity scaling and a range of 0 to 2^{15}–2. It defines the wait time between ending a move and switching to the holding current limit. That is, there will be a delay of ***Holding Delay*** trajectory cycles after Motion Complete, after which the In Holding bit in the Drive Status register will be set, and the motor command will be limited by the ***Holding Motor Limit.*** When the ***Holding Delay*** is set to 2^{15}–1 (its default), the axis will never go into holding current.

Description (cont.)

The Drive Current is in units of % maximum current, with a scaling of $100/2^{15}$. Its range is 0 to 2^{15}- 1. It defines the value used for the active motor command when driving a step motor, that is, when not in a holding state. This setting is used only by Atlas amplifiers driving step motors. It is not used by ION or MC58113, which use **SetMotorCommand** instead.

GetCurrent gets the indicated holding current parameter.

These commands were documented as **SetHoldingCurrent** and **GetHoldingCurrent** in previous versions of this manual. The name has been changed for clarity, but the command remains backwards compatible.

Atlas

When setting Holding Current or Drive Current this command will be relayed to an attached Atlas amplifier.

Restrictions

For pulse & direction motor types, only the *Holding Delay* is used. It delays the assertion of the At Rest output by the indicated number of cycles after a move is complete.

C-Motion API

```
PMDresult PMDSetCurrent       (PMDAxisInterface axis_intf,
                               PMDuint16 parameter,
                               PMDuint16 value)
PMDresult PMDGetCurrent       (PMDAxisInterface axis_intf,
                               PMDuint16 parameter,
                               PMDuint16* value)
```

VB-Motion API

```
MagellanAxis.CurrentSet( [in] parameter, [in] value )
MagellanAxis.CurrentGet( [in] parameter, [out] value )
```

see

GetDriveStatus (p. 38), **Set/GetSampleTime** (p. 180), **SetMotorCommand** (p. 151)

SetCurrentControlMode buffered 43h
GetCurrentControlMode 44h

Syntax

SetCurrentControlMode *axis mode*
GetCurrentControlMode *axis*

Motor Types

▨	Brushless DC	Microstepping	▨

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Phase A /B Current Loops*	0
	FOC	1
	Third leg floating	2

Packet Structure

SetCurrentControlMode

	15 … 12	11 … 8	7 … 0
	0	*axis*	**43**h

First data word

	15 … 0
write	*mode*

GetCurrentControlMode

	15 … 12	11 … 8	7 … 0
	0	*axis*	**44**h

First data word

	15 … 0
read	*mode*

Description

SetCurrentControlMode configures the axis to use either the Phase A/B method or the FOC method for current control.

For three-phase brushless DC motors some products also support the third leg floating method, in which only two of the three motor terminals is actively driven at any time, the remaining terminal being left floating. This method may be appropriate for motors intended for commutation by Hall effect sensors.

GetCurrentControlMode gets the buffered current loop *mode* for the indicated axis.

Atlas

These commands will be relayed to an attached Atlas amplifier. Atlas does not buffer the current control mode.

Restrictions

This command is only available on products that include a digital current loop.

SetCurrentControlMode is a buffered command. It will not take effect until an update is done on the current loop (through **Update** command, **MultiUpdate** command, or update action on breakpoint). The value read by **GetCurrentControlMode** is the buffered setting.

C-Motion API

```
PMDresult PMDSetCurrentControlMode(PMDAxisInterface axis_intf,
                                   PMDuint16 mode)
PMDresult PMDGetCurrentControlMode(PMDAxisInterface axis_intf,
                                   PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.CurrentControlMode = mode
mode = MagellanAxis.CurrentControlMode
```

see

Update (p. 215), **Set/GetUpdateMask** (p. 211), **MultiUpdate** (p. 65), **Set/GetBreakpointUpdateMask** (p. 97), **GetFOCValue** (p. 44), **Get/SetFOC** (p. 141), **GetCurrentLoopValue** (p. 34), **Get/SetCurrentLoop** (p. 120)

SetCurrentFoldback 41h
GetCurrentFoldback 42h

Syntax

SetCurrentFoldback *axis parameter value*
GetCurrentFoldback *axis parameter*

Motor Types

DC Brush	Brushless DC	Microstepping	▨

Arguments

Name	Instance	Encoding	Units
axis	*Axis1*	0	
	Axis2	1	
	Axis3	2	
	Axis4	3	
parameter	*Continuous Current Limit*	0	A
	Energy Limit	1	A^2s

	Type	Range/Scaling
value	unsigned 16-bit	see below

Packet Structure

SetCurrentFoldback

	15 … 12	11 … 8	7 … 0
	0	axis	**41**h

First data word

	15 … 0
write	*parameter*

Second data word

	15 … 0
write	*value*

GetCurrentFoldback

	15 … 12	11 … 8	7 … 0
	0	axis	**42**h

First data word

	15 … 0
write	*parameter*

Second data word

	15 … 0
read	*value*

Description

SetCurrentFoldback is used to set various I^2t foldback-related parameters. Two parameters can be set, the Continuous Current Limit, and the Energy Limit. The units of Continuous Current Limit are convertible to milliAmps, and represent percentage of maximum peak current, with scaling of $100/2^{15}$. The range is from 0% to the factory default continuous current limit setting.

The maximum current is the largest current that can be represented rather than the maximum that can be sourced or sensed. The maximum current can be calculated via the formula

Max = Current Scaling * 0x8000

For example for the high power Altas, using the scale factor from Section 3.11, "Atlas Conversion Factors," of the *Atlas Complete Technical Reference* the maximum current = 1.526mA * 0x8000 = 50A.

When using this command with the ION drive, check the *ION Digital Drive User Manual* for exact scaling values. Different drives have different scaling values and default limit settings.

SetCurrentFoldback (cont.) 41h
GetCurrentFoldback 42h

Description (cont.)

When using this command with the MC58113, the current scaling depends on the circuit used to sense current, the current limit range extends to 100%.

The units of Energy Limit are convertible to Amp^2s. The range is from 0% to the factory default energy limit setting. When using this command with the ION drive, check the *ION Digital Drive User Manual* for exact scaling values. For Atlas, use the *Atlas Complete Technical Reference.* Different drives have different scaling values and default limit settings.

For MC58113, the time unit is one current control period of 51.2 μs, and an additional scaling factor of 2^{31} is applied. If the current conversion factor is k A/count, then the energy conversion factor is:

$k^2 A^2 * 51.2e\text{-}6\ s * 2^{31}$

For example, for a current conversion factor of 1.526 mA/count, the energy conversion factor is:

$1.526e\text{-}3\ A * 1.526e\text{-}3\ A * 51.2e\text{-}6\ s * 2^{31} = 0.2560\ A^2s/count$

The ***Continuous Current Limit*** is used by the current foldback algorithm. When the current output of the drive exceeds this setting, accumulation of the I^2 energy above this setting begins. Once the accumulated excess I^2 energy exceeds the value specified by the ***Energy Limit*** parameter, a current foldback condition exists and the commanded current will be limited to the specified ***Continuous Current Limit***. When this occurs, the Current Foldback bit in the Event Status and Drive Status registers will be set. When the accumulated I^2 energy above the ***Continuous Current Limit*** drops to zero (0), the limit is removed, and the Current Foldback bit in the Drive Status register is cleared.

SetEventAction can be used to configure a change in operating mode when current foldback occurs. Doing this does not interfere with the basic operation of Current Foldback described above. If this is done, the Current Foldback bit in the Event Status register must be cleared prior to restoring the operating mode, regardless of whether the system is in current foldback or not.

When current control is not active, a current foldback event always causes a change to the disabled state (all loops and motor output are disabled), regardless of the programmed Event Action. Changing the operating mode from disabled requires clearing of the Current Foldback bit in Event Status.

GetCurrentFoldback gets the maximum continuous current setting.

Atlas

These commands will be relayed to an attached Atlas amplifier.

Restrictions

This command is only available on products that support digital current control.

Values of ***Continuous Current Limit*** greater than the factory setting for maximum continuous current are not allowed.

C-Motion API

```
PMDresult PMDSetCurrentFoldback(PMDAxisInterface axis_intf,
                                PMDuint16 parameter,
                                PMDuint16 value)
PMDresult PMDGetCurrentFoldback(PMDAxisInterface axis_intf,
                                PMDuint16 parameter,
                                PMDuint16* value)
```

VB-Motion API

```
MagellanAxis.CurrentFoldbackSet( [in] parameter, [in] value )
MagellanAxis.CurrentFoldbackGet( [in] parameter, [out] value )
```

see **GetActiveOperatingMode** (p. 24), **GetEventStatus** (p. 42), **GetDriveStatus** (p. 38), **ResetEventStatus** (p. 80), **RestoreOperatingMode** (p. 82), **Set/GetEventAction** (p. 135)

SetCurrentLoop buffered 73h
GetCurrentLoop 74h

Syntax

SetCurrentLoop *axis phase_parameter value*
GetCurrentLoop *axis phase_parameter*

Motor Types

DC Brush	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
phase	*Phase A*	0
	Phase B	1
	Both (A and B)	2
parameter	*Proportional Gain (KpCurrent)*	0
	Integrator Gain (KiCurrent)	1
	Integrator Sum Limit (ILimitCurrent)	2

	Type	Range/Scaling
value	unsigned 16 bits	see below

Packet Structure

SetCurrentLoop

15–12	11–8	7–0
0	*axis*	**73**h

First data word

	15–12	11–8	7–0
write	0	*phase*	*parameter*

Second data word

	15–0
write	*value*

GetCurrentLoop

15–12	11–8	7–0
0	*axis*	**74**h

First data word

	15–12	11–8	7–0
write	0	*phase*	*parameter*

Second data word

	15–0
read	*value*

Description

Set/GetCurrentLoop is used to configure the operating parameters of the Phase A/B PI digital current loops. See the product user guide for more information on how each ***parameter*** is used in the current loop processing. The ***value*** written/read is always an unsigned 16-bit value, with the parameter-specific scaling shown below:

Parameter	Range	Scaling	Units
Proportional Gain (KpCurrent)	0 to 2^{15}–1	1/64	gain
Integer Gain (KiCurrent)	0 to 2^{15}–1	1/256	gain/cycles
Integrator Sum Limit (ILimitCurrent)	0 to 2^{15}–1	1/100	% current * cycles

A setting of 64 for ***KpCurrent*** corresponds to a gain of 1. That is, an error signal of 30% maximum current will cause the proportional contribution of the current loop output to be 30% of maximum output. Similarly, setting ***KiCurrent*** to 256 gives it a gain of 1, and the value of the integrator sum would become the integrator contribution to the output. The units of time for the integrator sum are cycles.

Description (cont.)

ILimitCurrent is used to limit the contribution of the integrator sum at the output. Its effect depends on the value of *KiCurrent*. Setting *ILimitCurrent* to 1000 when *KiCurrent* is 10 means that the maximum contribution to the output is 1000 x 10 = 10,000 out of 2^{15} - 1 or approximately 30.5%

The *phase* argument can be used to set the operating parameters for the A and B loops independently. In most cases, the A and B loops will not require different operating parameters, so **SetCurrentLoop** can be used with a *phase* of 2, which sets both the A and B loops in a single API command. For **GetCurrentLoop**, a *phase* of 2 is not valid.

Atlas

These commands will be relayed to an attached Atlas amplifier.

Restrictions

Set/GetCurrentLoop are buffered commands. All parameters set are buffered, and will not take effect until an update is done on the current loop (through **Update** command, **MultiUpdate** command, or update action on breakpoint). The values read by **GetCurrentLoop** are the buffered settings.

This command is only supported in products that include digital current control, and when the current control mode is Phase A/B.

C-Motion API

```
PMDresult PMDSetCurrentLoop(PMDAxisInterface axis_intf,
                            PMDuint8 phase,
                            PMDuint8 parameter,
                            PMDuint16 value)
PMDresult PMDGetCurrentLoop(PMDAxisInterface axis_intf,
                            PMDuint8 phase,
                            PMDuint8 parameter,
                            PMDuint16* value)
```

VB-Motion API

```
MagellanAxis.CurrentLoopSet ( [in] phase,
                              [in] parameter,
                              [in] value )
MagellanAxis.CurrentLoopGet ( [in] phase,
                              [in] parameter,
                              [out] value )
```

see

Update (p. 215), **Set/GetUpdateMask** (p. 211), **MultiUpdate** (p. 65), **Set/GetBreakpointUpdateMask** (p. 97), **GetCurrentLoopValue** (p. 34), **Set/GetCurrentControlMode** (p. 115)

SetDeceleration buffered 91h
GetDeceleration 92h

Syntax

SetDeceleration *axis deceleration*
GetDeceleration *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
deceleration	unsigned 32 bits	0 *to* 2^{31}–1	$1/2^{16}$	counts/cycle^2 microsteps/cycle^2

Packet Structure

SetDeceleration

15–12	11–8	7–0
0	*axis*	**91**h

First data word

	31–16
write	*deceleration* (high-order part)

Second data word

	15–0
write	*deceleration* (low-order part)

GetDeceleration

15–12	11–8	7–0
0	*axis*	**92**h

First data word

	31–16
read	*deceleration* (high-order part)

Second data word

	15–0
read	*deceleration* (low-order part)

Description

SetDeceleration loads the maximum deceleration buffer register for the specified *axis*.

GetDeceleration returns the value of the maximum deceleration buffer.

Scaling example: To load a value of 1.750 counts/cycle^2 multiply by 65,536 (giving 114,688) and load the resultant number as a 32-bit number, giving 0001 in the high word and C000h in the low word. Retrieved numbers (**GetDeceleration**) must correspondingly be divided by 65,536 to convert to units of counts/cycle^2 or steps/cycle^2

Restrictions

This is a buffered command. The new value set will not take effect until the next **Update** or **MultiUpdate** command is entered, with the Trajectory Update bit set in the update mask.

These commands are used with the Trapezoidal and Velocity Contouring profile modes. They are not used with the Electronic Gear or S-curve profile mode.

Note: If *deceleration* is set to zero (0), then the value specified for acceleration (**SetAcceleration**) will automatically be used to set the magnitude of deceleration.

C-Motion API

```
PMDresult PMDSetDeceleration(PMDAxisInterface axis_intf,
                             PMDuint32 deceleration)
PMDresult PMDGetDeceleration(PMDAxisInterface axis_intf,
                             PMDuint32* deceleration)
```

VB-Motion API

```
Dim deceleration as Long
MagellanAxis.Deceleration = deceleration
deceleration = MagellanAxis.Deceleration
```

see

Set/GetAcceleration (p. 83), **Set/GetPosition** (p. 172), **Set/GetVelocity** (p. 213), **MultiUpdate** (p. 65), **Update** (p. 215)

SetDefault 89h
GetDefault 8Ah

Syntax

SetDefault *axis variable value*
GetDefault *axis variable*

Motor Type

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
variable	*CanMode*	0
	SerialPortMode485	1
	Type	**Range/Scaling**
value	32 bits	see below

Packet Structure

SetDefault

0	*axis*	**89**h
15 12	11 8	7 0

	First data word
write	*variable*
	15 0

	Second data word
write	*value* (high-order part)
	31 16

	Third data word
write	*value* (low-order part)
	15 0

GetDefault

0	*axis*	**8A**h
15 12	11 8	7 0

	First data word
write	*variable*
	15 0

	Second data word
read	*value* (high-order part)
	31 16

	Third data word
read	*value* (low-order part)
	15 0

Description

SetDefault is used to override the reset default settings of system variables. When **SetDefault** is invoked to change the reset default of a ***variable***, it stores the ***value*** sent by the user in non-volatile memory. It does not modify the value of the variable in active use. On subsequent system power cycles or resets, this ***value*** will become the default for the selected ***variable***.

The value for each variable is the value that would be used normally by the "Set/Get" command for that variable. When configuring variables that are 16-bit values, the value should be sent as the low order part of the 32-bit ***value***.

The ***axis*** sent with **Set/GetDefault** may or may not be relevent, depending on whether the parameter is an axis-specific parameter or not.

GetDefault gets the reset default value of the indicated ***variable*** from non-volatile memory.

Restrictions

This command is only available in ION products.

The **SetDefault** command can only be executed when motor output is disabled (e.g., immediately after power-up or reset).

C-Motion API

```
PMDresult PMDSetDefault (PMDAxisInterface axis_intf,
                         PMDuint16 variable,
                         PMDuint32 value)
PMDresult PMDGetDefault (PMDAxisInterface axis_intf,
                         PMDuint16 variable,
                         PMDuint32* value)
```

VB-Motion API

```
MagellanAxis.DefaultSet( [in] variable, [in] value )
MagellanAxis.DefaultGet( [in] variable, [out] value )
```

see

Reset (p. 75)

SetDriveCommandMode 7Eh
GetDriveCommandMode 7Fh

Syntax

SetDriveCommandMode *mode*
GetDriveCommandMode *mode*

Motor Type

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Encoding
mode	16-bit unsigned	see below

Packet Structure

SetDriveCommandMode

	0	7Eh
	15 8	7 0
write	*mode*	
	15	0

GetDriveCommandMode

	0	7Fh
	15 8	7 0
read	*mode*	
	15	0

Description

SetDriveCommandMode is used to change the command format for drive motor torque. Currently it may be used to put an attached Atlas amplifier into pulse and direction input mode, by using a mode value of 14h. After setting an Atlas amplifier to pulse and direction mode it will not be possible for Magellan to communicate with it, except by electrically connecting the Magellan pulse and direction outputs and changing the Magellan output mode. **SetDriveCommandMode** does not change Magellan output mode.

GetDriveCommandMode returns the current Atlas command mode, see *Atlas Digital Amplifier Complete Technical Reference* for more detail.

Atlas

These commands are relayed to an attached Atlas amplifier.

C-Motion API

```
PMDresult PMDSetDriveCommandMode(PMDAxisInterface axis_intf,
                                 PMDuint16 mode,
PMDresult PMDGetDriveCommandMode(PMDAxisInterface axis_intf,
                                 PMDuint16* mode)
```

VB-Motion API

```
MagellanAxis.DriveCommandMode = mode
       mode = MagellanAxis.DriveCommandMode
```

SetDriveFaultParameter 62h
GetDriveFaultParameter 60h

Syntax

SetDriveFaultParameter *axis parameter value*
GetDriveFaultParameter *axis parameter*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding	Units
axis	*Axis1*	0	
	Axis2	1	
	Axis3	2	
	Axis4	3	
parameter	*Overvoltage Limit*	0	V
	Undervoltage Limit	1	V
	Event Recovery Mode	2	N/A
	Watchdog Limit	3	s
	Temperature Limit	4	°C
	Temperature Hysteresis	5	°C
	— (Reserved)	6	
	— (Reserved)	7	
	Shunt voltage limit	8	V
	Shunt duty	9	%
	Bus current supply limit	10	A
	Bus current return limit	11	A

	Type	Range	Scaling
value	unsigned 16 bits	see below	see below

Packet Structure

SetDriveFaultParameter

15–12	11–8	7–0
0	*axis*	**62**h

First data word

	15–0
write	*parameter*

Second data word

	15–0
write	*value*

GetDriveFaultParameter

15–12	11–8	7–0
0	*axis*	**60**h

First data word

	15–0
write	*parameter*

Second data word

	15–0
read	*value*

Description

SetDriveFaultParameter sets various drive operation limits. The particular limit set depends on the parameter argument. When an operation limit is exceeded, motor output will be disabled and either a Drive Exception or Overtemperature event will be raised, and a bit set in the Drive Fault Status register to indicate the fault.

Description (cont'd)

Not all products support all limits, consult product-specific documentation for more detail.

GetDriveFaultParameter returns the limits set by **SetDriveFaultParameter**.

The Overvoltage and Undervoltage limit parameters set the thresholds for determination of overvoltage and undervoltage conditions. If the bus voltage exceeds the Overvoltage Limit value, an overvoltage condition occurs. If the bus voltage is less than the Undervoltage Limit value, an undervoltage condition occurs. Both the Overvoltage Limit and Undervoltage Limit have ranges of 0 to 2^{16} - 1; the scaling is product-dependent.

For example, to set the overvoltage threshold on Atlas to 30V, Overvoltage Limit should be set to 30V/1.3612 mv = 22039. On an MC58113 system with a maximum readable voltage of 90V, Overvoltage Limit should be set to (30V / 90V) * 65535 = 21845.

GetDriveFaultParameter reads the indicated limit.

The Event Recovery mode and Watchdog Limit are relevant only to an axis driving an Atlas amplifier, see *Atlas Digital Amplifier Complete Technical Reference* scaling and use. These commands were previously documented as Set/GetBusVoltageLimits. The names have been changed for clarity as more fault parameter options were added.

Temperature Limit and Temperature Hysteresis are used either with an attached Atlas amplifier or with a motion control IC with a temperature input. In the case of the motion control IC the temperature scaling depends on external hardware. Because the input thermistor voltage may either rise or fall with actual temperature the sign of the temperature limit is used to indicate the sign of the gain: With a positive sign the internal temperature reading is just the input voltage. With a negative sign, the internal temperature reading is the input voltage subtracted from 3.3V, and the limit applied to that reading is the absolute value of the argument. In both cases 08000h corresponds to 3.3V.

Shunt voltage limit and Shunt duty are used with motion control ICs that support a shunt PWM output to control bus voltage rise due to regeneration. As long as the bus voltage remains below the shunt voltage limit the shunt PWM will remain inactive, when bus voltage rises above the limit, the shunt PWM will become active, with a duty cycle specified by Shunt duty. Shunt duty is scaled so that 08000h corresponds to 100%. The shunt PWM will remain active until bus voltage falls below the shunt voltage limit by a fixed hysteresis of 2.5%.

The bus current supply and bus current return limits are limits on the measured bus current supply and the computed bus current return values. When either current exceeds the specified limit motor output will be disabled, a DriveException event raised, and the Overcurrent Fault bit set in the Drive Fault status register.

Atlas

These commands will be relayed to an attached Atlas amplifier.

Restrictions

Get/SetDriveFaultParameter is only available in products equipped with Bus voltage sensors.

The ***Overvoltage Limit*** cannot be set to a value greater than the reset default setting, and the ***Undervoltage Limit*** cannot be set to a value less than the reset default setting.

Motion API

```
PMDresult PMDSetDriveFaultParameter(PMDAxisInterface axis_intf,
                                    PMDuint16 parameter,
                                    PMDuint16 value)
PMDresult PMDGetDriveFaultParameter(PMDAxisInterface axis_intf,
                                    PMDuint16 parameter,
                                    PMDuint16* value)
```

VB-Motion API

```
MagellanAxis.DriveFaultParameterSet( [in] parameter, [in] value )
MagellanAxis.DriveFaultParameterGet( [in] parameter, [out] value )
```

see

Set/GetFaultOutMask (p. 137), **GetBusVoltage** (p. 28), **GetDriveFaultStatus** (p. 36), **ClearDriveFaultStatus** (p. 18), **GetEventStatus** (p. 42), **ResetEventStatus** (p. 80)

SetDrivePWM 23h
GetDrivePWM 24h

Syntax

SetDrivePWM *parameter value*
GetDrivePWM *parameter*

Motor Type

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding	Units
parameter	Limit	0	%
	Dead Time	1	ns
	Signal Sense	2	N/A
	Frequency	3	Hz
	Refresh Period	4	ns
	Refresh Time	5	ns
	Minimum Current Read Time	6	ns
	Type	**Range/Scaling**	
value	16-bit unsigned	see below	

Packet Structure

SetDrivePWM

	15 … 8	7 … 0
	0	**23**h
write	0	*parameter*
write	*value* (15 … 0)	

GetDrivePWM

	15 … 8	7 … 0
	0	**24**h
write	0	*parameter*
read	*value* (15 … 0)	

Description

SetDrivePWM sets parameters used for controlling amplifier PWM output. The PWM Limit register limits the maximum PWM duty cycle, and hence the effective output voltage. The range is from 0 to 2^{14}, 2^{14} corresponding to 100% PWM modulation.

The PWM Dead Time option controls the dead time added for High/Low PWM output between turning off the high side switch and turning on the low side, or vice versa. It has units of ns.

The PWM Frequency option controls the frequency for all PWM signals, the value is approximately the actual frequency, in Hz, scaled by 1/4. The available options are shown in the table below. Not all products support all frequencies.

Approximate Frequency	PWM bit Resolution	Actual Frequency	SetPWMFrequency Value
20 kHz	10	19.531 kHz	5,000
40 kHz	9	39.062 kHz	10,000
80 kHz	8	78.124 kHz	20,000

The PWM Signal Sense register controls whether an individual PWM signal is active high, encoded by a set bit, or active low, encoded by a clear bit. The PWM signal sense is not applied in the case of the sign signal for sign/magnitude PWM. The register layout is shown below:

Signal	Bit
PWM A High/PWM A Mag	0
PWM A Low	1
PWM B High/PWM B Mag	2
PWM B Low	3
PWM C High/PWM C Mag	4
PWM C Low	5
PWM D High/PWM D Mag	6
PWM D Low	7
reserved	8-14
PWM shunt	15

The PWM Refresh Period and PWM Refresh Time options are used to specify a minimum amount of off time when in High/Low PWM output mode. This may be required in order to allow charge pump capacitors to recharge. The Refresh Time is specified in ns, and the Refresh Period in commutation cycles. The low side of each PWM channel will be guaranteed to be on for at least the Refresh Time for every Refresh Period cycles.

The PWM Minimum Current Read time option is used to specify a minimum amount of off time for two out of the three PWM output channels for three phase output in PWM High/Low output mode. For motion control ICs supporting leg current sensing this may be required in order to get accurate current measurement. It has units of ns.

GetDrivePWM returns the parameters set by **SetDrivePWM**.

Atlas

These commands are relayed to an attached Atlas amplifier.

C-Motion API

```
PMDresult PMDSetDrivePWM(PMDAxisInterface axis_intf,
                         PMDuint16 option,
                         PMDuint16 value);
PMDresult PMDGetDrivePWM(PMDAxisInterface axis_intf,
                         PMDuint16 option,
                         PMDuint16* value)
```

VB-Motion API

```
Magellan.DrivePWMSet( [in] parameter, [in] value )
Magellan.DrivePWMGet( [in] parameter, [out] value )
```

SetEncoderModulus 8Dh
GetEncoderModulus 8Eh

Syntax

SetEncoderModulus *axis modulus*
GetEncoderModulus *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
modulus	unsigned 16 bits	0 *to* 2^{15}–1	unity	counts

Packet Structure

SetEncoderModulus

15–12	11–8	7–0
0	*axis*	**8D**h

Data

	15–0
write	*modulus*

GetEncoderModulus

15–12	11–8	7–0
0	*axis*	**8E**h

Data

	15–0
read	*modulus*

Description

SetEncoderModulus sets the parallel word range for the specified *axis* when parallel-word feedback is used. The *modulus* determines the range of the connected device. For multi-turn systems, this value is used to determine when a position wrap condition has occurred. The value provided should be one half of the actual range of the axis. For example, if the parallel-word input is used with a linear potentiometer connected to an external A/D (Analog to Digital converter) which has 12 bits of resolution, then the total range is 4,096 and a value of 2,048 should be loaded with this command.

GetEncoderModulus returns the encoder modulus.

Restrictions

A value for encoder modulus is only required when the encoder source is set to parallel.

C-Motion API

```
PMDresult PMDSetEncoderModulus(PMDAxisInterface axis_intf,
                               PMDuint16 modulus)
PMDresult PMDGetEncoderModulus(PMDAxisInterface axis_intf,
                               PMDuint16* modulus)
```

VB-Motion API

```
Dim modulus as Short
MagellanAxis.EncoderModulus = modulus
modulus = MagellanAxis.EncoderModulus
```

see

Set/GetEncoderSource (p. 132)

SetEncoderSource DAh
GetEncoderSource DBh

Syntax

SetEncoderSource *axis source*
GetEncoderSource *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
source	*Incremental*	0
	Parallel	1
	None	2
	Loopback	3
	Pulse and Direction	4
	Hall Sensors	5
	32 bit parallel	6
	(Reserved)	7
	sin/cos	8
	SSI	9
	(Reserved)	10-12
	BiSS	13

Packet Structure

SetEncoderSource

0	*axis*	**DA**h
15 … 12	11 … 8	7 … 0

Data

	0	*source*
write	15 … 3	2 … 0

GetEncoderSource

0	*axis*	**DB**h
15 … 12	11 … 8	7 … 0

Data

	0	*source*
read	15 … 3	2 … 0

Description

SetEncoderSource sets the type of encoder feedback (*Incremental* quadrature encoder or *Parallel*-word) for the specified *axis*. When incremental quadrature is selected the motion control IC expects A and B quadrature signals to be input at the QuadA and QuadB axis inputs. When parallel-word is selected the motion control IC expects user-defined external circuitry connected to the motion control IC's external bus to load a 16-bit word containing the current position value for the selected axis. External feedback devices with less than 16 bits may be used but the unused bits must be sign extended or zeroed.

When motor type (see **SetMotorType** (p. 154)) is set to *Pulse and Direction* and the encoder source is set to *Loopback*, the step output is internally fed back into the quadrature counters. This allows for position capture of the step position when a physical encoder is not present.

Description (cont.)

When the encoder source is set to *Pulse and Direction,* then Magellan expects the incoming position encoding to correspond to a pulse & direction encoding scheme rather than a quadrature encoding scheme. This feature is most commonly used with electronic gear mode, so that the Magellan processor can be driven by a motion controller that outputs pulse & direction signals.

GetEncoderSource returns the code for the current type of feedback.

Restrictions

A *Loopback* source is only supported for pulse & direction motors. *Loopback* is not supported in single-chip versions (MC58110 & MC55110). In order for the loopback option to work correctly the step invert bit of the signal sense register must be set. This bit is set as a side-effect of setting the loopback encoder source.

A source value of *None* is typically only used with microstepping and pulse & direction motors.

Not all products support all types of encoders. See the product user guide.

When using a parallel word encoder with the MotorType set to *Pulse&Direction* or *MicroStepping*, the **SetCountToStepRatio** command must be used prior to this command.

When using BiSS or SSI encoders with N-Series ION, setting encoder parameters using PRP commands is required before **SetEncoderSource**. See the product user guide.

When using sin/cos encoders calibration may be required. For more information see **Set/GetAnalogCalibration** and **CalibrateAnalog**, and consult the product user guide.

C-Motion API

```
PMDresult PMDSetEncoderSource(PMDAxisInterface axis_intf, PMDuint16 source)
PMDresult PMDGetEncoderSource(PMDAxisInterface axis_intf, PMDuint16* source)
```

VB-Motion API

```
Dim source as Short
MagellanAxis.EncoderSource = source
source = MagellanAxis.EncoderSource
```

see

CalibrateAnalog (p. 17)**, Set/GetAnalogCalibration** (p. 88)**, Set/GetEncoderModulus** (p. 131)
Set/GetSignalSense (p. 186)

SetEncoderToStepRatio DEh
GetEncoderToStepRatio DFh

Syntax

SetEncoderToStepRatio *axis counts steps*
GetEncoderToStepRatio *axis*

Motor Types

▨	▨	**Microstepping**	**Pulse & Direction**

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
counts	unsigned 16 bits	1 *to* 2^{15}–1	unity	counts
steps	unsigned 16 bits	1 *to* 2^{15}–1	unity	microsteps

Packet Structure

SetEncoderToStepRatio

15–12	11–8	7–0
0	*axis*	**DE**h

First data word

	15–0
write	*counts*

Second data word

	15–0
write	*steps*

GetEncoderToStepRatio

15–12	11–8	7–0
0	*axis*	**DF**h

First data word

	15–0
read	*counts*

Second data word

	15–0
read	*steps*

Description

SetEncoderToStepRatio sets the ratio of the number of encoder counts to the number of output steps per motor rotation used by the motion control IC to convert encoder counts into steps. *Counts* is the number of encoder counts per full rotation of the motor. *Steps* is the number of steps output by the motion control IC per full rotation of the motor. Since this command sets a ratio, the parameters do not have to be for a full rotation as long as they correctly represent the encoder count to step ratio. **GetEncoderToStepRatio** returns the ratio of the number of encoder counts to the number of output steps per motor rotation.

C-Motion API

```
PMDresult PMDSetEncoderToStepRatio(PMDAxisInterface axis_intf,
                                   PMDuint16 counts, PMDuint16 steps)
PMDresult PMDGetEncoderToStepRatio(PMDAxisInterface axis_intf,
                                   PMDuint16* counts, PMDuint16* steps)
```

VB-Motion API

```
MagellanAxis.EncoderToStepRatioSet( [in] counts, [in] steps )
MagellanAxis.EncoderToStepRatioGet( [out] counts, [out] steps )
```

see

Set/GetActualPositionUnits (p. 87)

SetEventAction 48h
GetEventAction 49h

Syntax

SetEventAction *axis event action*
GetEventAction *axis event*

Motor Types

DC Brush	**Brushless DC**	**Microstepping**	**Pulse & Direction**

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
event	*Immediate*	0
	Positive Limit	1
	Negative Limit	2
	Motion Error	3
	Current Foldback	4
action	*None*	0
	— (Reserved)	1
	Abrupt Stop	2
	Smooth Stop	3
	— (Reserved)	4
	Disable Position Loop & Higher Modules	5
	Disable Current Loop & Higher Modules	6
	Disable Motor Output & Higher Modules	7
	Abrupt Stop with Position Error Clear	8

Packet Structure

SetEventAction

15–12	11–8	7–0
0	*axis*	**48**h

First data word

	15–0
write	*event*

Second data word

	15–0
write	*action*

GetEventAction

15–12	11–8	7–0
0	*axis*	**49**h

First data word

	15–0
write	*event*

Second data word

	15–0
read	*action*

Description

SetEventAction configures what actions will be taken by the *axis* in response to a given *event*. The *action* can be either to modify the operating mode by disabling some or all of the loops, or, in the case of all loops remaining on, to perform an abrupt or smooth stop. The *Abrupt Stop* action can be done with or without a clearing of the position error.

SetEventAction (cont.) 48h
GetEventAction 49h

Description (cont.)

When, through **SetEventAction**, one of the *events* causes an *action*, the event bit in the Event Status register must be cleared prior to returning to operation. For trajectory stops, this means that the bit must be cleared prior to performing another trajectory move. For changes in operating mode, this means that the bit must be cleared prior to restoring the operating mode, either by **RestoreOperatingMode** or **SetOperatingMode**.

An exception is the Motion Error event, which only needs to be cleared in Event Status if its *action* is *Abrubt Stop* or *Smooth Stop*. If it causes changes in operating mode, the operating mode can be restored without clearing the bit in Event Status first.

GetEventAction gets the action that is currently programmed for the given event with the exception of the *Immediate* event, which cannot be read back.

Atlas

For the Current Foldback event this command will be sent to an attached Atlas amplifier before being applied to the local Magellan register. The foldback event action is set automatically on Atlas by Magellan when first establishing SPI communication.

Restrictions

If a *Smooth Stop* action occurs while the trajectory mode is S-curve, the trajectory cannot be restarted until the smooth stop is complete. If a *Smooth Stop* action occurs while the trajectory mode is electronic gearing, an abrupt stop will occur.

C-Motion API

```
PMDresult PMDSetEventAction (PMDAxisInterface axis_intf,
                             PMDuint16 event,
                             PMDuint16 action)
PMDresult PMDGetEventAction (PMDAxisInterface axis_intf,
                             PMDuint16 event,
                             PMDuint16* action)
```

VB-Motion API

```
Dim action as Short
MagellanAxis.EventAction( event ) = action
action = MagellanAxis.EventAction( event )
```

see

GetActiveOperatingMode (p. 24), **RestoreOperatingMode** (p. 82), **Set/GetOperatingMode** (p. 156)

SetFaultOutMask FBh
GetFaultOutMask FCh

Syntax

SetFaultOutMask *axis mask*
GetFaultOutMask *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mask	see below	bitmask

Packet Structure

SetFaultOutMask

0	*axis*	**FB**h
15 12	11 8	7 0

First data word

write	*mask*
	15 0

GetFaultOutMask

0	*axis*	**FC**h
15 12	11 8	7 0

First data word

read	*mask*
	15 0

Description

SetFaultOutMask configures the mask on Event Status register bits that will be ORed together on the FaultOut pin. The FaultOut pin is active high, as are the bits in Event Status. Thus, FaultOut will go high when any of the enabled bits in Event Status are set (1). The ***mask*** parameter is used to determine what bits in the Event Status register can cause FaultOut high, as follows:

Name	Bit
Motion Complete	0
Wrap-around	1
Breakpoint 1	2
Position Capture	3
Motion Error	4
Positive Limit	5
Negative Limit	6
Instruction Error	7
Disable	8
Overtemperature Fault	9
Drive Exception	10
Commutation Error	11
Current Foldback	12
— (Reserved)	13
Breakpoint 2	14
— (Reserved)	15

SetFaultOutMask (cont.) FBh
GetFaultOutMask FCh

Description (cont.)

For example, a *mask* setting of hexadecimal 0610h will configure the FaultOut pin to go high upon a motion error, Overtemperature Fault, or Bus Voltage Fault. The FaultOut pin stays high until all Fault enabled bits in Event Status are cleared. The default value for the FaultOut *mask* is 0600h – Overtemperature Fault and Bus Voltage Fault enabled.

GetFaultOutMask gets the current *mask* for the indicated *axis*.

Atlas

The Magellan version of this command does *not* apply to an Atlas amplifier. In order to control Atlas behavior it is necessary to send a command directly, see *Atlas Digital Amplifier Complete Technical Reference* for more detail.

Restrictions

This command is only available on products that include a FaultOut pin.

Depending on the product, all of the specified bits in Event Status may not be available.

In addition to the FaultOut *mask* on the Event Status register, the FaultOut pin is driven by a mask on the Drive Fault Status register (bits 4, 2, 1, and 0) which cannot be changed, and is internally ORed with the FaultOut *mask* on Event Status.

C-Motion API

```
PMDresult PMDSetFaultOutMask (PMDAxisInterface axis_intf,
                              PMDuint16 mask)
PMDresult PMDGetFaultOutMask (PMDAxisInterface axis_intf,
                              PMDuint16* mask)
```

VB-Motion API

```
Dim mask as Short
MagellanAxis.FaultOutMask = mask
mask = MagellanAxis.FaultOutMask
```

see

Set/GetInterruptMask (p. 146)

SetFeedbackParameter 21h
GetFeedbackParameter 22h

Syntax

SetFeedbackParameter *parameter value*
GetFeedbackParameter *parameter value*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
parameter	*Encoder Modulus*	0
	Type	**Range/Scaling**
value	32-bit unsigned	see below

Packet Structure

SetFeedbackParameter

	0	**21**h
	15 8	7 0
write	0	*parameter*
	15 8	7 0
	value (high order parameter)	
write	15 8	7 0
write	*value* (low order parameter)	
	15	0

GetFeedbackParameter

	0	**22**h
	15 8	7 0
write	0	*parameter*
	15 8	7 0
	value (high order parameter)	
read	15 8	7 0
read	*value* (low order parameter)	
	15	0

Description

SetFeedbackParameter sets parameters used to configure position feedback devices. Encoder modulus is a 32 bit parallel encoder modulus, its least significant 16 bit word is identical with the parameter set by **SetEncoderModulus**.

The Encoder Modulus sets the parallel word range for the specified axis when 32 bit parallel-word feedback is used. The modulus determines the range of the connected device. For multi-turn systems, this value is used to determine when a position wrap condition has occurred. The value provided should be one half of the actual range of the axis. For example, if the parallel-word input is used with an SSI encoder which has 24 bits of resolution, then the total range is 16777216 and a value of 8388608 should be used as the encoder modulus.

GetFeedbackParameter returns the value of parameters set by **SetFeedbackParameter**.

SetFeedbackParameter (cont.) 21h
GetFeedbackParameter 22h

C-Motion API

```
PMDresult PMDSetFeedbackParameter (PMDAxisInterface axis_intf,
                    PMDuint8 parameter,
                    PMDuint32 value);
PMDresult PMDGetFeedbackParameter (PMDAxisInterface axis_intf,
                    PMDuint8 parameter,
                    PMDuint32* value)
```

VB-Motion API

```
MagellanAxis.FeedbackParameter( [in] parameter
             [out] value )
```

see

SetEncoderModulus (p. 131)

SetFOC buffered F6h
GetFOC F7h

Syntax

SetFOC *axis loop_parameter value*
GetFOC *axis loop_parameter*

Motor Types

	Brushless DC	**Microstepping**	

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
loop	*Direct(D)*	0
	Quadrature(Q)	1
	Both(D and Q)	2
parameter	*Proportional Gain (KpDQ)*	0
	Integrator Gain (KiDQ)	1
	Integrator Sum Limit (ILimitDQ)	2
	Type	**Range/Scaling**
value	unsigned 16 bits	see below

Packet Structure

SetFOC

	15–12	11–8	7–0
	0	*axis*	**F6**h

First data word

	15–12	11–8	7–0
write	0	*loop*	*parameter*

Second data word

	15–0
write	*value*

GetFOC

	15–12	11–8	7–0
	0	*axis*	**F7**h

First data word

	15–12	11–8	7–0
write	0	*loop*	*parameter*

Second data word

	15–0
read	*value*

Description

Set/GetFOC is used to configure the operating parameters of the FOC-Current control. See the product user guide for more information on how each ***parameter*** is used in the current loop processing. The ***value*** written/read is always an unsigned 16-bit value, with the parameter-specific scaling shown below:

Parameter	**Range**	**Scaling**	**Units**
Proportional Gain (KpDQ)	0 to 2^{15}–1	1/64	gain
Integrator Gain (KiDQ)	0 to 2^{15}–1	1/256	gain/cycles
Integrator Sum Limit (ILimitDQ)	0 to 2^{15}–1	1/100	% current * cycles

A setting of 64 for ***KpDQ*** corresponds to a gain of 1. That is, an error signal of 30% maximum current will cause the proportional contribution of the current loop output to be 30% of maximum output.

Description (cont.)

Similarly, setting *KiDQ* to 256 gives it a gain of 1; the value of the integrator sum would become the integrator contribution to the output.

ILimitDQ is used to limit the contribution of the integrator sum at the output. Its effect depends on the value of *KiDQ*. Setting *IlimitDQ* to 1000 when *KiDQ* is 10 means that the maximum contribution to the output is 1000 x 10 = 10,000 out of 2^{15} - 1 or approximately 30.5%. The units of time for the integrator sum are cycles.

The *loop* argument allows individual configuration of the parameters for the D and Q current loops. Alternately, a *loop* of 2 can be used with **SetFOC** to set the D and Q loops with a single API command. A *loop* of 2 is not valid for **GetFOC**.

Atlas

These commands are relayed to an attached Atlas amplifier.

Restrictions

Set/GetFOC are buffered commands. All parameters set are buffered, and will not take effect until an update is done on the current loop (through **Update** command, **MultiUpdate** command, or update action on breakpoint). The values read by **GetFOC** are the buffered settings.

These commands are only supported in products that include digital current control, and when the current control mode is set to FOC.

C-Motion API

```
PMDresult PMDSetFOC (PMDAxisInterface axis_intf,
                     PMDuint8 loop,
                     PMDuint8 parameter,
                     PMDuint16 value)
PMDresult PMDGetFOC (PMDAxisInterface axis_intf,
                     PMDuint8 loop,
                     PMDuint8 parameter,
                     PMDuint16* value)
```

VB-Motion API

```
MagellanAxis.FOCSet( [in] loop, [in] parameter, [in] value )
MagellanAxis.FOCGet( [in] loop, [in] parameter, [out] value )
```

see

Update (p. 215), **Set/GetUpdateMask** (p. 211), **MultiUpdate** (p. 65), **Set/GetBreakpointUpdateMask** (p. 98), **GetFOCValue** (p. 44), **Set/GetCurrentControlMode** (p. 115)

SetGearMaster AEh
GetGearMaster AFh

Syntax

SetGearMaster *axis masterAxis_source*
GetGearMaster *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
masterAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
source	*Actual*	0
	Commanded	1

Packet Structure

SetGearMaster

	0	*axis*	**AE**h
	15 12	11 8	7 0

Data

write	0	*source*	*masterAxis*
	15 9	8	7 0

GetGearMaster

	0	*axis*	**AF**h
	15 12	11 8	7 0

Data

read	0	*source*	*masterAxis*
	15 9	8	7 0

Description

SetGearMaster establishes the slave (*axis*) and master (*masterAxis*) axes for the electronic-gearing profile, and sets the *source*, *Actual* or *Commanded*, of the master axis position data to be used.

The *masterAxis* determines the axis that will drive the slave axis. Both the slave and the master axes must be enabled (**SetOperatingMode** command). The source determines whether the master axis' commanded position as determined by the trajectory generator will be used to drive the slave axis, or whether the master axis' encoder position will be used to drive the slave.

GetGearMaster returns the value for the geared axes and position source.

Restrictions

SetGearMaster (cont.) AEh
GetGearMaster AFh

C-Motion API

```
PMDresult PMDSetGearMaster(PMDAxisInterface axis_intf,
                           PMDAxis masterAxis, PMDuint8 source)
PMDresult PMDGetGearMaster(PMDAxisInterface axis_intf,
                           PMDAxis* masterAxis, PMDuint8* source)
```

VB-Motion API

```
MagellanAxis.GearMasterSet( [in] masterAxis, [in] source )
MagellanAxis.GearMasterGet( [out] masterAxis, [out] source )
```

see **Set/GetGearRatio** (p. 145)

SetGearRatio GetGearRatio

buffered **14h** **59h**

Syntax

SetGearRatio *slaveAxis ratio*
GetGearRatio *slaveAxis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding		
slaveAxis	*Axis1*	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	**Range**	**Scaling**	**Units**
ratio	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	SlaveCts/MasterCts

Packet Structure

SetGearRatio

0	*slaveAxis*	**14**h
15 12	11 8	7 0

First data word

write	*ratio* (high-order part)
	31 16

Second data word

write	*ratio* (low-order part)
	15 0

GetGearRatio

0	*slaveAxis*	**59**h
15 12	11 8	7 0

First data word

read	*ratio* (high-order part)
	31 16

Second data word

read	*ratio* (low-order part)
	15 0

Description

SetGearRatio sets the ratio between the master and slave axes for the Electronic Gear profile for the given *slaveAxis*. Positive ratios cause the slave to move in the same direction as the master, negative ratios in the opposite direction. The specified ratio has a unity scaling of 65,536.

GetGearRatio returns the gear ratio set for the specified *slaveAxis*.

Scaling examples:

ratio value	resultant ratio
–32,768	.5 negative slave counts for each positive master count
1,000,000	15.259 positive slave counts for each positive master count
123	.0018 positive slave counts for each positive master count

Restrictions

This is a buffered command. The new value set will not take effect until the next **Update** or **MultiUpdate** command is entered, with the Trajectory Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetGearRatio(PMDAxisInterface axis_intf, PMDint32 ratio)
PMDresult PMDGetGearRatio(PMDAxisInterface axis_intf, PMDint32* ratio)
```

VB-Motion API

```
Dim ratio as Long
MagellanAxis.GearRatio = ratio
ratio = MagellanAxis.GearRatio
```

see

Set/GetGearMaster (p. 143), **MultiUpdate** (p. 65), **Update** (p. 215)

SetInterruptMask 2Fh
GetInterruptMask 56h

Syntax

SetInterruptMask *axis mask*
GetInterruptMask *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mask	*Motion Complete*	0001h
	Wrap-around	0002h
	Breakpoint 1	0004h
	Capture Received	0008h
	Motion Error	0010h
	Positive Limit	0020h
	Negative Limit	0040h
	Instruction Error	0080h
	Disable	0100h
	Overtemperature Fault	0200h
	Drive Exception	0400h
	Commutation Error	0800h
	Current Foldback	1000h
	Breakpoint 2	4000h

Packet Structure

SetInterruptMask

15–12	11–8	7–0
0	*axis*	**2F**h

Data

	15–0
write	*mask*

GetInterruptMask

15–12	11–8	7–0
0	*axis*	**56**h

Data

	15–0
read	*mask*

Description

SetInterruptMask determines which bits in the Event Status register of the specified *axis* will cause a host interrupt. For each interrupt *mask* bit that is set to 1, the corresponding Event Status register bit will cause an interrupt when that status register bit goes active (is set to 1). Interrupt mask bits set to 0 will not generate interrupts.

GetInterruptMask returns the *mask* for the specified *axis*.

SetInterruptMask also controls CAN event notification when using the motion control IC's CAN 2.0B interface. Whenever a host interrupt is activated, a CAN message is generated using message ID 180h + nodeID, notifying interested CAN nodes of the change in the Event Status register.

Example: The interrupt *mask* value 28h will generate an interrupt when either the Positive Limit bit or the Capture Received bit of the Event Status register goes active (set to 1).

Restrictions

C-Motion API

```
PMDresult PMDSetInterruptMask(PMDAxisInterface axis_intf,
                              PMDuint16 mask)
PMDresult PMDGetInterruptMask(PMDAxisInterface axis_intf,
                              PMDuint16* mask)
```

VB-Motion API

```
Dim mask as Short
MagellanAxis.InterruptMask = mask
mask = MagellanAxis.InterruptMask
```

see

ClearInterrupt (p. 19), **GetInterruptAxis** (p. 49), **Set/GetFaultOutMask** (p. 137)

SetJerk GetJerk buffered 13h 58h

Syntax

SetJerk *axis jerk*
GetJerk *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding		
axis	*Axis1*	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	**Range**	**Scaling**	**Units**
jerk	unsigned 32 bits	0 *to* 2^{31}–1	1/2^{32}	counts/cycle^3 microsteps/cycle^3

Packet Structure

SetJerk

15–12	11–8	7–0
0	*axis*	**13**h

First data word

	31–16
write	*jerk* (high-order part)

Second data word

	15–0
write	*jerk* (low-order part)

GetJerk

15–12	11–8	7–0
0	*axis*	**58**h

First data word

	31–16
read	*jerk* (high-order part)

Second data word

	15–0
read	*jerk* (low-order part)

Description

SetJerk loads the Jerk register in the parameter buffer for the specified *axis*.

GetJerk reads the contents of the Jerk register.

Scaling example: To load a jerk value (rate of change of acceleration) of 0.012345 counts/cycle^3 (or steps/cycle^3) multiply by 2^{32} or 4,294,967,296. In this example this gives a value to load of 53,021,371 (decimal) which corresponds to a high word of 0329h and a low word of 0ABBh when loading each word in hexadecimal.

Restrictions

SetJerk is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory Update bit set in the update mask.

This command is used only with the S-curve profile mode. It is not used with the Trapezoidal, Velocity Contouring, or Electronic Gear profile modes.

C-Motion API

```
PMDresult PMDSetJerk(PMDAxisInterface axis_intf, PMDuint32 jerk)
PMDresult PMDGetJerk(PMDAxisInterface axis_intf, PMDuint32* jerk)
```

VB-Motion API

```
Dim jerk as Long
MagellanAxis.Jerk = jerk
jerk = MagellanAxis.Jerk
```

see

Set/GetAcceleration (p. 83), **Set/GetDeceleration** (p. 122), **Set/GetPosition** (p. 172), **Set/GetVelocity** (p. 213), **MultiUpdate** (p. 65), **Update** (p. 215)

SetMotionCompleteMode **EB**h
GetMotionCompleteMode **EC**h

Syntax

SetMotionCompleteMode *axis mode*
GetMotionCompleteMode *axis*

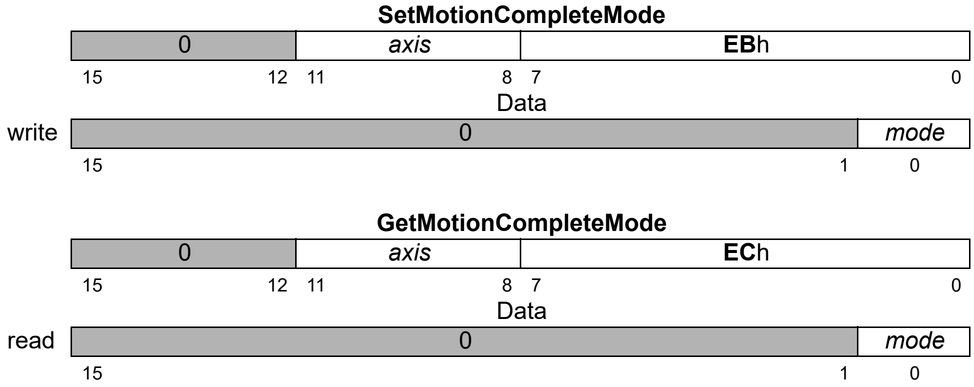
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*commanded*	0
	actual	1

Packet Structure



Description

SetMotionCompleteMode establishes the source for the comparison which determines the motion-complete status for the specified ***axis***. When set to ***commanded***, the motion is considered complete when the profile velocity reaches zero (0) and no further motion will occur without an additional host command. This mode is unaffected by the actual encoder location.

When set to ***actual*** mode the motion complete bit will be set when the above condition is true, and when the actual encoder position has been within the settle window (**SetSettleWindow** command) for the number of cycles specified by the **SetSettleTime** command. The settle timer is started at zero (0) at the end of the trajectory profile motion, so a minimum delay of settle time cycles will occur after the trajectory profile motion is complete.

GetMotionCompleteMode returns the value for the motion-complete mode.

Restrictions

C-Motion API

```
PMDresult PMDSetMotionCompleteMode(PMDAxisInterface axis_intf,
                                   PMDuint16 mode)
PMDresult PMDGetMotionCompleteMode(PMDAxisInterface axis_intf,
                                   PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.MotionCompleteMode = mode
mode = MagellanAxis.MotionCompleteMode
```

see

Set/GetSettleTime (p. 184), **Set/GetSettleWindow** (p. 185)

SetMotorBias 0Fh
GetMotorBias 2Dh

Syntax

SetMotorBias *axis bias*
GetMotorBias *axis*

Motor Types

DC Brush	Brushless DC		

Arguments

Name	Instance	Encoding		
axis	*Axis1*	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	**Range**	**Scaling**	**Units**
bias	signed 16 bits	-2^{15} *to* $2^{15}-1$	$100/2^{15}$	% output

Packet Structure

SetMotorBias

	0	*axis*	**0F**h
	15 12	11 8	7 0

	Data
write	*bias*
	15 0

GetMotorBias

	0	*axis*	**2D**h
	15 12	11 8	7 0

	Data
read	*bias*
	15 0

Description

SetMotorBias sets the output *bias* of the digital servo filter for the specified *axis*.

GetMotorBias reads the value of the *bias* of the digital servo filter.

Scaling example: If it is desired that a motor bias value of –2.5% of full scale be placed on the servo filter output, then this register should be loaded with a value of –2.5*32,768/100 = –819 (decimal). This corresponds to a loaded hexadecimal value of 0FCCDh.

Restrictions

C-Motion API

```
PMDresult PMDSetMotorBias(PMDAxisInterface axis_intf, PMDint16 bias)
PMDresult PMDGetMotorBias(PMDAxisInterface axis_intf, PMDint16* bias)
```

VB-Motion API

```
Dim bias as Short
MagellanAxis.MotorBias = bias
bias = MagellanAxis.MotorBias
```

see

Set/GetMotorCommand (p. 151), **Set/GetMotorLimit** (p. 153)

SetMotorCommand buffered 77h
GetMotorCommand 69h

Syntax

SetMotorCommand *axis command*
GetMotorCommand *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
command	signed 16 bits	-2^{15} *to* $2^{15}-1$	$100/2^{15}$	% output

Packet Structure

SetMotorCommand

15–12	11–8	7–0
0	*axis*	**77**h

Data

	15–0
write	*command*

GetMotorCommand

15–12	11–8	7–0
0	*axis*	**69**h

Data

	15–0
read	*command*

Description

SetMotorCommand loads the Motor Command buffer register of the specified ***axis***. For axes configured for microstepping motors, this command is used to control the magnitude of the output waveform. For DC brush and brushless DC motors, this command directly sets the Motor Output register when the Position Loop and Trajectory Generator modules are disabled in the operating mode.

GetMotorCommand reads the contents of the motor command buffer register.

Scaling example: If it is desired that a Motor Command value of 13.7% of full scale be output to the motor, then this register should be loaded with a value of 13.7 * 32,768/100 = 4,489 (decimal). This corresponds to a hexadecimal value of 1189h.

Atlas

Note that **SetMotorCommand** is not used to set step motor drive current when using an Atlas amplifier, **SetCurrent** should be used instead.

Restrictions

SetMotorCommand is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Position Loop Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetMotorCommand(PMDAxisInterface axis_intf,
                             PMDint16 command)
PMDresult PMDGetMotorCommand(PMDAxisInterface axis_intf,
                             PMDint16* command)
```

VB-Motion API

```
Dim command as Short
MagellanAxis.MotorCommand = command
command = MagellanAxis.MotorCommand
```

see

SetCurrent (p. 113), **Set/GetMotorBias** (p. 150), **Set/GetMotorLimit** (p. 153), **Set/GetOperatingMode** (p. 156), **MultiUpdate** (p. 65), **Update** (p. 215)

SetMotorLimit 06h
GetMotorLimit 07h

Syntax

SetMotorLimit *axis limit*
GetMotorLimit *axis*

Motor Types

DC Brush	Brushless DC	▨	▨

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
limit	unsigned 16 bits	0 *to* 2^{15}–1	$100/2^{15}$	% output

Packet Structure

SetMotorLimit

0	*axis*	**06h**
15 12	11 8	7 0

Data

write	*limit*
	15 0

GetMotorLimit

0	*axis*	**07h**
15 12	11 8	7 0

Data

read	*limit*
	15 0

Description

SetMotorLimit sets the maximum value for the motor output command allowed by the digital servo filter of the specified ***axis***. Motor command values beyond this value will be clipped to the specified motor command limit. For example if the motor limit was set to 1,000 and the servo filter determined that the current motor output value should be 1,100, the actual output value would be 1,000. Conversely, if the output value was –1,100, then it would be clipped to –1,000. This command is useful for protecting amplifiers, motors, or system mechanisms when it is known that a motor command exceeding a certain value will cause damage.

GetMotorLimit reads the motor limit value.

Scaling example: If it is desired that a motor limit of 75% of full scale be established, then this register should be loaded with a value of 75.0 * 32,767/100 = 24,576 (decimal). This corresponds to a hexadecimal value of 06000h.

Restrictions

This command only affects the motor output when the position loop or trajectory generator is enabled. When the motion control IC is in open loop mode, this command has no effect.

C-Motion API

```
PMDresult PMDSetMotorLimit(PMDAxisInterface axis_intf,
                              PMDuint16 limit);
PMDresult PMDGetMotorLimit(PMDAxisInterface axis_intf,
                              PMDuint16* limit)
```

VB-Motion API

```
Dim limit as Short
MagellanAxis.MotorLimit = limit
limit = MagellanAxis.MotorLimit
```

see

Set/GetMotorBias (p. 150), **Set/GetMotorCommand** (p. 151), **Set/GetOperatingMode** (p. 156)

SetMotorType 02h
GetMotorType 03h

Syntax

SetMotorType *axis type*
GetMotorType *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
type	*Brushless DC (3 phase)*	0
	Closed-loop stepper	1
	Microstepping (3 phase)	2
	Microstepping (2 phase)	3
	Pulse & Direction	4
	DC Brush	7

Packet Structure

SetMotorType

	15 … 12	11 … 8	7 … 0
	0	*axis*	**02**h

Data

	15 … 3	2 … 0
write	0	*type*

GetMotorType

	15 … 12	11 … 8	7 … 0
	0	*axis*	**03**h

Data

	15 … 3	2 … 0
read	0	*type*

Description

SetMotorType sets type of motor being driven by the selected ***axis***. This operation sets the number of phases for commutation on the axis, as well as internally configuring the motion control IC for the motor type.

The following table describes each motor type, and the number of phases to be commutated.

Motor type	Commutation
Brushless DC (3 phase)	3 phase
Closed-loop stepper*	2 phase
Microstepping (3 phase)	3 phase
Microstepping (2 phase)	2 phase
Pulse & Direction	None
DC Brush	None

* Also called Brushless DC (2-phase)

GetMotorType returns the configured motor type for the selected ***axis***.

Restrictions

The motor type should only be set once for each axis, either via the motor configuration word during device startup, or immediately after reset using **SetMotorType**. Once it has been set, it should not be changed. Executing **SetMotorType** will reset many variables to their motor type specific default values.

Not all motor types are available on all products. See the product user guide.

C-Motion API

```
PMDresult PMDSetMotorType (PMDAxisInterface axis_intf, PMDuint8 type)
PMDresult PMDGetMotorType (PMDAxisInterface axis_intf, PMDuint8* type)
```

VB-Motion API

```
Dim type as Short
MagellanAxis.MotorType = type
type = MagellanAxis.MotorType
```

see **Reset** (p. 75)

SetOperatingMode 65h
GetOperatingMode 66h

Syntax

SetOperatingMode *axis mode*
GetOperatingMode *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range/Scaling
mode	unsigned 16-bit	see below

Packet Structure

SetOperatingMode

0	*axis*	**65**h
15 12	11 8	7 0

First data word

write

0	*mode*
15 4	3 0

GetOperatingMode

0	*axis*	**66**h
15 12	11 8	7 0

First data word

read

0	*mode*
15 4	3 0

Description

SetOperatingMode configures the operating mode of the *axis*. Each bit of the *mode* configures whether a feature/loop of the *axis* is active or disabled, as follows:

Name	Bit	Description
Axis Enabled	0	0: No *axis* processing, *axis* outputs in reset state. 1: *axis* active.
Motor Output Enabled	1	0: *axis* motor outputs disabled. 1: *axis* motor outputs enabled.
Current Control Enabled	2	0: *axis* current control bypassed. 1: *axis* current control active.
—	3	Reserved
Position Loop Enabled	4	0: *axis* position loop bypassed. 1: *axis* position loop active.
Trajectory Enabled	5	0: trajectory generator disabled. 1: trajectory generator enabled.
—	6–15	Reserved

When the *axis* is disabled, no processing will be done on the axis, and the axis outputs will be at their reset states. When the axis motor output is disabled, the axis will function normally, but its motor outputs will be in their disabled state. When a loop is disabled (position or current loop), it operates by passing its input directly to its output, and clearing all internal state variables (such as integrator sums, etc.). When the trajectory generator is disabled, it operates by commanding 0 velocity.

Description (cont.)

For example, to configure an axis for Torque mode, (trajectory and position loop disabled) the operating mode would be set to hexadecimal 0007h.

This command should be used to configure the static operating mode of the *axis*. The actual current operating mode may be changed by the axis in response to safety events, or user-programmable events. In this case, the present operating mode is available using **GetActiveOperatingMode**. **GetOperatingMode** will always return the static operating mode set using **SetOperatingMode**. Executing the **SetOperatingMode** command sets both the static operating mode and the active operating mode to the desired state.

GetOperatingMode gets the operating mode of the *axis*.

Atlas

The SetOperatingMode command will be relayed to an attached Atlas amplifier before being applied to the local Magellan register. GetOperatingMode does not require any additional Atlas communication.

Restrictions

The possible operating modes of an axis is product specific, and in some cases axis specific. See the product user guide for a description of what operating modes are supported on each axis.

C-Motion API

```
PMDresult PMDSetOperatingMode(PMDAxisInterface axis_intf,
                              PMDuint16 mode)
PMDresult PMDGetOperatingMode(PMDAxisInterface axis_intf,
                              PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.OperatingMode = mode
mode = MagellanAxis.OperatingMode
```

see

GetActiveOperatingMode (p. 24), **RestoreOperatingMode** (p. 82)

SetOutputMode E0h
GetOutputMode 6Eh

Syntax

SetOutputMode *axis mode*
GetOutputMode *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	▨

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Parallel DAC Offset Binary*	0
	PWM Sign Magnitude	1
	PWM 50/50 Magnitude	2
	SPI DAC Offset Binary	3
	Parallel DAC Sign Magnitude	4
	SPI DAC 2's Complement	5
	Atlas SPI	6
	PWM High/Low	7
	Pulse & Direction	8
	Atlas Recovery	9
	None	10

Packet Structure

SetOutputMode

0	*axis*	**E0**h
15 12	11 8	7 0

Data

write	0	*mode*
	15 4	3 0

GetOutputMode

0	*axis*	**6E**h
15 12	11 8	7 0

Data

read	0	*mode*
	15 4	3 0

Description

SetOutputMode sets the form of the motor output signal of the specified *axis*.

GetOutputMode returns the value for the motor output mode.

Restrictions

Not all output modes are available on all products. See the product user guide.

C-Motion API

```
PMDresult PMDSetOutputMode(PMDAxisInterface axis_intf, PMDuint16 mode)
PMDresult PMDGetOutputMode(PMDAxisInterface axis_intf, PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.OutputMode = mode
mode = MagellanAxis.OutputMode
```

see

SetOvertemperatureLimit 1Bh
GetOvertemperatureLimit 1Ch

Syntax

SetOvertemperatureLimit *axis limit*
GetOvertemperatureLimit *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding		
axis	*Axis1*	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	**Range**	**Scaling**	**Units**
limit	signed 16 bits	-2^{15} *to* $2^{15}-1$	2^8	°C

Packet Structure

SetOvertemperatureLimit

	0	*axis*	**1B**h
	15 12	11 8	7 0

First data word

write	*limit*
	15 0

GetOvertemperatureLimit

	0	*axis*	**1C**h
	15 12	11 8	7 0

First data word

read	*limit*
	15 0

Description

SetOvertemperatureLimit sets the temperature threshold upon which an overtemperature condition will occurr. For example, to set the overtemperature threshold at 60 degrees C, the value should be 60*256 = 15360. When the programmed threshold is exceeded, the Overtemperature Fault bit is set in the Event Status register, and the ***axis*** enters the overtemperature state.

GetOvertemperatureLimit gets the current overtemperature threshold setting.

This command is not used to set the temperature limit for MC58113. Use **SetDriveFaultParameter**.

Atlas

These commands are not used with Atlas.

Restrictions

Get/SetOvertemperatureLimit is only available in products equipped with temperature sensors.

If the ***axis*** has more than one temperature sensor, the temperature used to compare to the overtemperature threshold will be the highest value of all sensor readings.

The overtemperature threshold cannot be set to a value greater than the reset default setting.

C-Motion API

```
PMDresult PMDSetOvertemperatureLimit (PMDAxisInterface axis_intf,
                                      PMDint16 limit)
PMDresult PMDGetOvertemperatureLimit (PMDAxisInterface axis_intf,
                                      PMDint16* limit)
```

VB-Motion API

```
Dim limit as Short
MagellanAxis.OvertemperatureLimit = limit
limit = MagellanAxis.OvertemperatureLimit
```

see **GetTemperature** (p. 57), **GetEventStatus** (p. 42), **ResetEventStatus** (p. 80), **SetDriveFaultParameter** (p. 126)

SetPhaseAngle 84h
GetPhaseAngle 2Ch

Syntax

SetPhaseAngle *axis angle*
GetPhaseAngle *axis*

Motor Types

	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
angle	unsigned 16 bits	0 *to* 2^{15}–1	unity	counts microsteps

Packet Structure

SetPhaseAngle

0	*axis*	**84**h
15 12	11 8	7 0

Data

	angle
write	*angle*
	15 0

GetPhaseAngle

0	*axis*	**2C**h
15 12	11 8	7 0

Data

	angle
read	*angle*
	15 0

Description

SetPhaseAngle sets the instantaneous commutation angle for the specified ***axis***. For brushless DC motors, the phase angle is specified in units of encoder counts. For microstepping motors, it is specified in units of microsteps. **GetPhaseAngle** returns the value of the phase angle. To convert counts to an actual phase angle, divide by the number of encoder counts per electrical cycle and multiply by 360.

For example, if a value of 500 is retrieved using **GetPhaseAngle** and the counts per electrical cycle value has been set to 2,000 (**SetPhaseCounts** command), this corresponds to an angle of (500/2,000)*360 = 90 degrees current phase angle position. **SetPhaseAngle** resets the phase offset previously set by **SetPhaseOffset**.

Restrictions

The specified angle must not exceed the number set by the **SetPhaseCounts** command. Some Magellan products support a 32-bit commutation parameter interface using the commands **Set/GetCommutationParameter**. It is possible to set parameters through the 32-bit interface that cannot be represented using the 16-bit interface. If an attempt is made to read a non-representable value then a value representation error (37) will be raised.

C-Motion API

```
PMDresult PMDSetPhaseAngle(PMDAxisInterface axis_intf,
                           PMDuint16 angle)
PMDresult PMDGetPhaseAngle(PMDAxisInterface axis_intf,
                           PMDuint16* angle)
```

VB-Motion API

```
Dim angle as Short
MagellanAxis.PhaseAngle = angle
angle = MagellanAxis.PhaseAngle
```

see **Set/GetCommutationParameter** (p. 110), **Set/GetPhaseCounts** (p. 164)

SetPhaseCorrectionMode E8h
GetPhaseCorrectionMode E9h

Syntax

SetPhaseCorrectionMode *axis mode*
GetPhaseCorrectionMode *axis*

Motor Types

	Brushless DC		

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Disable*	0
	Index	1
	Hall	2

Packet Structure

SetPhaseCorrectionMode

15–12	11–8	7–0
0	*axis*	**E8h**

Data

	15–1	0
write	0	*mode*

GetPhaseCorrectionMode

15–12	11–8	7–0
0	*axis*	**E9h**

Data

	15–1	0
read	0	*mode*

Description

SetPhaseCorrectionMode controls the method used for phase correction on the specified axis. Phase correction is optional, and may be disabled by using mode 0. In mode 1 (Index) the encoder ***Index*** signal is used to update the commutation phase angle once per mechanical revolution. In mode 2 (Hall) a particular Hall sensor transition is used to update the commutation phase angle once every twelve electrical revolutions.

Phase correction ensures that the commutation angle will remain correct even if some encoder counts are lost due to electrical noise, or due to the number of encoder counts per electrical revolution not being an integer. Because Hall sensors normally have significant hysteresis index based correction is preferred if an index signal is available.

GetPhaseCorrectionMode returns the phase correction mode.

Restrictions

Hall phase correction mode is not supported by all products; it is supported by MC58113.

C-Motion API

```
PMDresult PMDSetPhaseCorrectionMode(PMDAxisInterface axis_intf,
                                    PMDuint16 mode)
PMDresult PMDGetPhaseCorrectionMode(PMDAxisInterface axis_intf,
                                    PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.PhaseCorrectionMode = mode
mode = MagellanAxis.PhaseCorrectionMode
```

see

GetPhaseCommand (p. 50), **InitializePhase** (p. 64), **Set/GetPhaseCounts** (p. 164)

SetPhaseCounts 75h
GetPhaseCounts 7Dh

Syntax

SetPhaseCounts *axis counts*
GetPhaseCounts *axis*

Motor Types

▨	**Brushless DC**	**Microstepping**	▨

Arguments

Name	Instance	Encoding		
axis	*Axis1*	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	**Range**	**Scaling**	**Units**
counts	unsigned 16 bits	1 *to* 2^{15}–1	unity	counts microsteps

Packet Structure

SetPhaseCounts

15–12	11–8	7–0
0	*axis*	**75**h

Data

	15–0
write	*counts*

GetPhaseCounts

15–12	11–8	7–0
0	*axis*	**7D**h

Data

	15–0
read	*counts*

Description

For axes configured for brushless DC motor types, **SetPhaseCounts** sets the number of encoder counts per electrical cycle of the motor. The number of electrical cycles is equal to 1⁄2 the number of motor poles. If this value is not an integer, then the closest integer value should be used, and phase correction mode should be enabled. See **SetPhaseCorrectionMode** (p. 163).

For axes configured for microstepping motor types, the number of microsteps per full step is set using the **SetPhaseCounts** command. The parameter used for this command represents the number of microsteps per electrical cycle (4 times the desired number of microsteps). For example, to set 64 microsteps per full step, the **SetPhaseCounts 256** command should be used. The maximum number of microsteps that can be generated per full step is 256, giving a maximum parameter value of 1024.

GetPhaseCounts returns the number of counts or microsteps per electrical cycle.

Restrictions

Some Magellan products support a 32-bit commutation parameter interface using the commands **Set/GetCommutationParameter**. It is possible to set parameters through the 32-bit interface that cannot be represented using the 16-bit interface. If an attempt is made to read a non-representable value then a value representation error (37) will be raised.

C-Motion API

```
PMDresult PMDSetPhaseCounts(PMDAxisInterface axis_intf, PMDuint16
counts)
PMDresult PMDGetPhaseCounts(PMDAxisInterface axis_intf, PMDuint16*
counts)
```

VB-Motion API

```
Dim counts as Short
MagellanAxis.PhaseCounts = counts
counts = MagellanAxis.PhaseCounts
```

see **Set/GetCommutationParameter** (p. 110), **Set/GetPhaseAngle** (p. 161)

SetPhaseInitializeMode E4h
GetPhaseInitializeMode E5h

Syntax

SetPhaseInitializeMode *axis mode*
GetPhaseInitializeMode *axis*

Motor Types

	Brushless DC		

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Algorithmic*	0
	Hall-based	1
	Pulse	2

Packet Structure

SetPhaseInitializeMode

15–12	11–8	7–0
0	*axis*	**E4**h

Data

	15–1	0
write	0	*mode*

GetPhaseInitializeMode

15–12	11–8	7–0
0	*axis*	**E5**h

Data

	15–1	0
read	0	*mode*

Description

In pulse mode the motion control IC briefly stimulates the motor windings and sets the initial phasing based on the observed motor response. **SetPhaseInitializeMode** establishes the mode in which the specified ***axis*** is to be initialized for commutation. The options are *Algorithmic* and *Hall-based*. In algorithmic mode the motion control IC briefly stimulates the motor windings and sets the initial phasing based on the observed motor response. In Hall-based initialization mode, the three Hall sensor signals are used to determine the motor phasing.

GetPhaseInitializeMode returns the value of the initialization mode.

Restrictions

Algorithmic mode should only be selected if it is known that the axis is free to move in both directions, and that a brief uncontrolled move can be tolerated by the motor, mechanism, and load.

Not all Magellan products support pulse phase initialization. N-series ION does.

C-Motion API

```
PMDresult PMDSetPhaseInitializeMode(PMDAxisInterface axis_intf,
                                    PMDuint16 mode)
PMDresult PMDGetPhaseInitializeMode(PMDAxisInterface axis_intf,
                                    PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.PhaseInitializeMode = mode
mode = MagellanAxis.PhaseInitializeMode
```

see

InitializePhase (p. 64), **Set/GetPhaseInitializeTime** (p. 167)

SetPhaseInitializeTime 72h
GetPhaseInitializeTime 7Ch

Syntax

SetPhaseInitializeTime *axis time*
GetPhaseInitializeTime *axis*

Motor Types

▨	**Brushless DC**	▨	▨

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
time	unsigned 16 bits	0 *to* 2^{15}–1	unity	cycles

Packet Structure

SetPhaseInitializeTime

0	*axis*	**72**h
15 12	11 8	7 0

Data

	15 0
write	*time*

GetPhaseInitializeTime

0	*axis*	**7C**h
15 12	11 8	7 0

Data

	15 0
read	*time*

Description

SetPhaseInitializeTime sets the time value (in cycles) to be used during the algorithmic phase initialization procedure. This value determines the duration of each of the four segments in the phase initialization algorithm.

GetPhaseInitializeTime returns the value of the phase initialization time.

Restrictions

C-Motion API

```
PMDresult PMDSetPhaseInitializeTime(PMDAxisInterface axis_intf,
                                    PMDuint16 time)
PMDresult PMDGetPhaseInitializeTime(PMDAxisInterface axis_intf,
                                    PMDuint16* time)
```

VB-Motion API

```
Dim time as Short
MagellanAxis.PhaseInitializeTime = time
time = MagellanAxis.PhaseInitializeTime
```

see

InitializePhase (p. 64), **Set/GetPhaseInitializeMode** (p. 166)

SetPhaseOffset 76h
GetPhaseOffset 7Bh

Syntax

SetPhaseOffset *axis offset*
GetPhaseOffset *axis*

Motor Types

▨	**Brushless DC**	▨	▨

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	**Range**	**Scaling**	**Units**
offset	unsigned 16 bits	0 *to* 2^{15}–1	unity	counts

Packet Structure

SetPhaseOffset

15–12	11–8	7–0
0	*axis*	**76**h

Data

	15–0
write	*offset*

GetPhaseOffset

15–12	11–8	7–0
0	*axis*	**7B**h

Data

	15–0
read	*offset*

Description

Before the first index capture has occurred, **GetPhaseOffset** will return –1. **SetPhaseOffset** sets the offset from the index mark of the specified ***axis*** to the internal zero phase angle. This command will have no immediate effect on the commutation angle but will have an effect once the index pulse is encountered. The settable range of phase offset is 0 to 32,767.

GetPhaseOffset returns the value of the phase offset.

To convert counts to a phase angle in degrees, divide by the number of encoder counts per electrical cycle and multiply by 360. For example, if a value of 500 is specified using **SetPhaseOffset** and the counts per electrical cycle value has been set to 2,000 (**SetPhaseCounts** command) this corresponds to an angle of (500/2,000)*360 = 90 degrees phase angle at the index mark.

Restrictions

Some Magellan products support a 32-bit commutation parameter interface using the commands **Set/GetCommutationParameter**. It is possible to set parameters through the 32-bit interface that cannot be represented using the 16-bit interface. If an attempt is made to read a non-representable value then a value representation error (37) will be raised.

C-Motion API

```
PMDresult PMDSetPhaseOffset(PMDAxisInterface axis_intf,
                            PMDint16 offset)
PMDresult PMDGetPhaseOffset(PMDAxisInterface axis_intf,
                            PMDint16* offset)
```

VB-Motion API

```
Dim offset as Short
MagellanAxis.PhaseOffset = offset
offset = MagellanAxis.PhaseOffset
```

see

Set/GetCommutationParameter (p. 110)

SetPhaseParameter 85h
GetPhaseParameter 86h

Syntax

SetPhaseParameter *axis parameter value*
GetPhaseParameter *axis parameter value*

Motor Types

▨	**Brushless DC**	▨

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
parameter	ramp time	0
	positive pulse time	1
	negative pulse time	2
	pulse command	3
	— (Reserved)	4
	ramp command	5

	Type	**Range**	**Scaling/Units**
value	unsigned 16bits	0 *to* 2^{15}–1	counts

Packet Structure

SetPhaseParameter

	▨ 0	*axis*	**85**h
	15 12	11 8	7 0
write	*parameter*		
	15 0		
write	*value*		
	15 0		

GetPhaseParameter

	▨ 0	*axis*	**85**h
	15 12	11 8	7 0
write	*parameter*		
	15 0		
read	*value*		
	15 0		

Description

SetPhaseParameter is used to set parameters required for brushless DC motor pulse phase initialization. Phase initialization is required for commutation using an incremental encoder; the method used is set by **SetPhaseInitializeMode**.

The positive pulse time is a non-negative count of sample periods giving the duration of the first, positive pulse. The default sample period is 102 µs, but it can be changed by **SetSampleTime**.

The negative pulse time is a non-negative count of sample periods giving the duration of the second, negative pulse. Each negative pulse follows immediately after a positive pulse. The time between successive pulse pairs is given by three times the positive pulse time.

The pulse command is a non-negative value that is used as the motor command during both the positive and negative pulses.

The ramp time is a non-negative count of sample periods giving the duration of the pull-in ramp part of pulse phase initialization. It is possible, though not recommended, to set this to zero.

SetPhaseParameter (cont.) 85h
GetPhaseParameter 86h

Description (cont.)

The ramp command is a non-negative value that is used as the motor command during the pull-in ramp.

By default all phase parameters are zero, however phase initialization cannot possibly work in that state.

The process of pulse phase initialization and how to set the various parameters is discussed in the *Juno Velocity and Torque IC User Guide*. **[Radey: Change cross ref?]**

GetPhaseParameter is used to read the values set by SetPhaseParameter.

Errors

Unrecognized parameter code, or value out of range.

C-Motion API

```
PMDresult PMDGetPhaseParameter (PMDAxisInterface axis_intf,
                                PMDuint16 parameter, PMDint16* value);
PMDresult PMDSetPhaseParameter (PMDAxisInterface axis_intf,
                                PMDuint16 parameter, PMDint16 value);
```

Script API

```
GetPhaseParameter parameter
SetPhaseParameter parameter value
```

C# API

```
Int32 value = PMDAxis.GetPhaseParameter(PMDPhaseParameter parameter);
PMDAxis.SetPhaseParameter(PMDPhaseParameter parameter, Int32 value);
```

Visual Basic API

```
Int32 value = PMDAxis.GetPhaseParameter(ByVal parameter
                                        As PMDPhaseParameter)
PMDAxis.SetPhaseParameter(ByVal parameter As PMDPhaseParameter,
                          ByVal value As Int32)
```

see

InitializePhase (p. 64), **SetPhaseInitializeMode** (p. 166)

SetPhasePrescale E6h
GetPhasePrescale E7h

Syntax

SetPhasePrescale *axis scale*
GetPhasePrescale *axis*

Motor Types

	Brushless DC		

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
scale	*Off*	0
	1/64	1
	1/128	2
	1/256	3

Packet Structure

SetPhasePrescale

15 12	11 8	7 0
0	*axis*	**E6**h

Data

	15 2	1 0
write	0	*scale*

GetPhasePrescale

15 12	11 8	7 0
0	*axis*	**E7**h

Data

	15 2	1 0
read	0	*scale*

Description

SetPhasePrescale controls scaling of the encoder counts before they are used to calculate a commutation angle for the specified ***axis***. When operated in the pre-scale mode, the motion control IC can commutate motors with a high number of counts per electrical cycle, such as motors with very high accuracy encoders.

SetPhasePrescale Off removes the scale factor.

GetPhasePrescale returns the value of the scaling mode.

Restrictions

Some Magellan products do not include this command because they support a full 32-bit commutation interface using **Set/GetCommutationParameter**.

C-Motion API

```
PMDresult PMDSetPhasePrescale(PMDAxisInterface axis_intf,
                              PMDuint16 scale);
PMDresult PMDGetPhasePrescale(PMDAxisInterface axis_intf,
                              PMDuint16* scale)
```

VB-Motion API

```
Dim scale as Short
MagellanAxis.PhasePrescale = scale
scale = MagellanAxis.PhasePrescale
```

see

Get/SetCommutationParameter (p. 110)

SetPosition buffered 10h
GetPosition 4Ah

Syntax

SetPosition *axis position*
GetPosition *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding		
axis	*Axis1*	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	**Range**	**Scaling**	**Units**
position	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts microsteps

Packet Structure

SetPosition

	0	*axis*	**10h**
	15 12	11 8	7 0

First data word

write	*position* (high-order part)
	31 16

Second data word

write	*position* (low-order part)
	15 0

GetPosition

	0	*axis*	**4Ah**
	15 12	11 8	7 0

First data word

read	*position* (high-order part)
	31 16

Second data word

read	*position* (low-order part)
	15 0

Description

SetPosition specifies the trajectory destination of the specified *axis*. It is used in the Trapezoidal and S-curve profile modes.

GetPosition reads the contents of the buffered position register.

Restrictions

SetPosition is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetPosition(PMDAxisInterface axis_intf,
                         PMDint32 position);
PMDresult PMDGetPosition(PMDAxisInterface axis_intf,
                         PMDint32* position)
```

VB-Motion API

```
Dim position as Long
MagellanAxis.Position = position
position = MagellanAxis.Position
```

see

Set/GetAcceleration (p. 83), **Set/GetDeceleration** (p. 122), **Set/GetJerk** (p. 148), **Set/GetVelocity** (p. 213), **MultiUpdate** (p. 65), **Update** (p. 215)

SetPositionErrorLimit 97h
GetPositionErrorLimit 98h

Syntax

SetPositionErrorLimit *axis limit*
GetPositionErrorLimit *axis*

Motor Types

DC Brush	**Brushless DC**	**Microstepping**	**Pulse & Direction**

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	**Range**	**Scaling**	**Units**
limit	unsigned 32 bits	0 *to* 2^{31}–1	unity	counts microsteps

Packet Structure

SetPositionErrorLimit

	0	*axis*	**97**h
	15 12	11 8	7 0

First data word

write	*limit* (high-order part)
	31 16

Second data word

write	*limit* (low-order part)
	15 0

GetPositionErrorLimit

	0	*axis*	**98**h
	15 12	11 8	7 0

First data word

read	*limit* (high-order part)
	31 16

Second data word

read	*limit* (low-order part)
	15 0

Description

SetPositionErrorLimit sets the absolute value of the maximum position error allowable by the motion control IC for the specified ***axis***. If the position error exceeds this ***limit***, a motion error occurs. Such a motion error can cause a choice of actions, or no action, configurable using the **SetEventAction** (Motion Error) command.

When the motor type is microstepping or pulse & direction, this value is set in microsteps or steps, respectively.

GetPositionErrorLimit returns the value of the position error limit.

Restrictions

C-Motion API

```
PMDresult PMDSetPositionErrorLimit(PMDAxisInterface axis_intf,
                                   PMDuint32 limit)
PMDresult PMDGetPositionErrorLimit(PMDAxisInterface axis_intf,
                                   PMDuint32* limit)
```

VB-Motion API

```
Dim limit as Long
MagellanAxis.PositionErrorLimit = limit
limit = MagellanAxis.PositionErrorLimit
```

see

GetPositionError (p. 51), **GetActualPosition** (p. 87), **Set/GetPosition** (p. 172), **Set/GetEventAction** (p. 135)

SetPositionLoop buffered 67h
GetPositionLoop 68h

Syntax

SetPositionLoop *axis parameter value*
GetPositionLoop *axis parameter*

Motor Types

DC Brush	Brushless DC		

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
parameter	*PID Proportional Gain (Kp)*	0
	PID Integrator Gain (Ki)	1
	PID Integrator Limit (Ilimit)	2
	PID Derivative Gain (Kd)	3
	PID Derivative Time	4
	PID Output Gain (Kout)	5
	Velocity Feedforward Gain (Kvff)	6
	Acceleration Feedforward Gain (Kaff)	7
	Biquad1, Enable Filter	8
	Biquad1, CoefficientB0	9
	Biquad1, CoefficientB1	10
	Biquad1, CoefficientB2	11
	Biquad1, CoefficientA1	12
	Biquad1, CoefficientA2	13
	Biquad1, CoefficientK	14
	Biquad2, Enable filter	15
	Biquad2, CoefficientB0	16
	Biquad2, CoefficientB1	17
	Biquad2, CoefficientB2	18
	Biquad2, CoefficientA1	19
	Biquad2, CoefficientA2	20
	Biquad2, CoefficientK	21

	Type	Range/Scaling
value	signed 32 bits	see below

Packet Structure

SetPositionLoop

0	*axis*	**67**h
15 12	11 8	7 0

First data word

	parameter
write	
	15 0

Second data word

	value (high-order part)
write	
	31 16

Third data word

	value (low-order part)
write	
	15 0

SetPositionLoop (cont.) buffered 67h
GetPositionLoop 68h

Packet Structure (cont.)

GetPositionLoop

0	*axis*	**68**h
15 12	11 8	7 0

First data word

write	*parameter*
	15 0

Second data word

read	*value* (high-order part)
	31 16

Third data word

read	*value* (low-order part)
	15 0

Description

Set/GetPositionLoop is used to configure the operating parameters of the PID position loop. See the product user guide for more information on how each ***parameter*** is used in the position loop processing. Though these commands always use 32-bit data, the range and format vary depending on the ***parameter***, as follows:

Parameter	Range	Scaling	Units
Velocity Feedforward Gain (Kvff)	0 to 2^{15}–1	unity	gain/cycles
Acceleration Feedforward Gain (Kaff)	0 to 2^{15}–1	unity	gain/cycles^2
PID Proportional Gain (Kp)	0 to 2^{15}–1	unity	gain
PID Integrator Gain (Ki)	0 to 2^{15}–1	1/256	gain/cycles
PID Derivative Gain (Kd)	0 to 2^{15}–1	unity	gain*cycles
PID Integrator Limit (Ilimit)	0 to 2^{31}–1	unity	count*cycles
PID Derivative Time	1 to 2^{15}–1	unity	cycles
PID Output Gain (Kout)	0 to 2^{16}–1	100/2^{16}	% output
Biquad1, Enable Filter	0 to 1	0=disable, 1=enable	
Biquad1, CoefficientB0	–2^{15} to 2^{15}–1	unity	
Biquad1, CoefficientB1	–2^{15} to 2^{15}–1	unity	
Biquad1, CoefficientB2	–2^{15} to 2^{15}–1	unity	
Biquad1, CoefficientA1	–2^{15} to 2^{15}–1	unity	
Biquad1, CoefficientA2	–2^{15} to 2^{15}–1	unity	
Biquad1, CoefficientK	0 to 2^{15}–1	unity	
Biquad2, Enable Filter	0 to 1	0=disable, 1=enable	
Biquad2, CoefficientB0	–2^{15} to 2^{15}–1	unity	
Biquad2, CoefficientB1	–2^{15} to 2^{15}–1	unity	
Biquad2, CoefficientB2	–2^{15} to 2^{15}–1	unity	
Biquad2, CoefficientA1	–2^{15} to 2^{15}–1	unity	
Biquad2, CoefficientA2	–2^{15} to 2^{15}–1	unity	
Biquad2, CoefficientK	0 to 2^{15}–1	unity	

Many of these parameters are self-descriptive. However, below are some additional comments on the use of specific parameters.

- *PID Derivative Time* has units of cycles. This is the sample time of the ***axis***, as configured by **SetSampleTime**. For example, if set to 10, the derivative term will be computed every 10 cycles of the axis position loop. *PID Integrator Limit* has units of count*cycles, and scaling of unity. This matches the units and scaling of the position loop integrator sum. For example, a constant position error of 100 counts which is present for 256 cycles will result an an integrator sum of 100*256 = 25,600.
- *PID Integrator Gain* has scaling of 1/256. Thus, a setting of 256 corresponds to “unity” integrator gain. From the above example, this would make the integrator sum of 25,600 create a contribution to the PID output of 25,600.
- *PID Output Gain* is a scaling factor applied to the output of the digital servo filter, with units of % output. Its default value is 65,535, or approximately 100% output. To set the scaling to, for example, 50% of output, *PID Output Gain* would be set to 32,767.
- The biquad coefficients configure the two biquad output filters. If both filters are enabled, their outputs are chained (filter1 followed by filter2). If filter1 is disabled for an axis, filter2 is also disabled for that axis, regardless of user setting of *Biquad2 Enable Filter*. The signed coefficients and unsigned scalar K combine to implement the following equation, for each filter:

$$Y_n = K \times (B_0 \times X_n + B_1 \times X_{n-1} + B_2 \times X_{n-2} + A_1 \times Y_{n-1} \times A_2 \times Y_{n-2})$$

Where Y_n is the filter output at cycle n, and X_n is the filter input at cycle n.

Restrictions

Set/GetPositionLoop are buffered commands. All parameters set are buffered, and will not take effect until an update is done on the position loop (through **Update** command, **MultiUpdate** command, or update action on breakpoint). The values read by **GetPositionLoop** are the buffered settings.

C-Motion API

```
PMDresult PMDSetPositionLoop(PMDAxisInterface axis_intf,
                             PMDuint16 parameter,
                             PMDint32 value)
PMDresult PMDGetPositionLoop(PMDAxisInterface axis_intf,
                             PMDuint16 parameter,
                             PMDint32* value)
```

VB-Motion API

```
MagellanAxis.PositionLoopSet( [in] parameter, [in] value )
MagellanAxis.PositionLoopGet( [in] parameter, [out] value )
```

see

Update (p. 215), **Set/GetUpdateMask** (p. 211), **MultiUpdate** (p. 65), **Set/GetBreakpointUpdateMask** (p. 97), **GetPositionLoopValue** (p. 52)

SetProfileMode buffered A0h
GetProfileMode A1h

Syntax

SetProfileMode *axis mode*
GetProfileMode *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Trapezoidal*	0
	Velocity Contouring	1
	S-curve	2
	Electronic Gear	3

Packet Structure

SetProfileMode

0	*axis*	**A0**h
15 12	11 8	7 0

Data

	0	*mode*
write	15 3	2 0

GetProfileMode

0	*axis*	**A1**h
15 12	11 8	7 0

Data

	0	*mode*
read	15 3	2 0

Description

SetProfileMode sets the profile mode for the specified *axis*.

GetProfileMode returns the contents of the profile-mode register for the specified *axis*.

Restrictions

SetProfileMode is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetProfileMode(PMDAxisInterface axis_intf,
                            PMDuint16 mode)
PMDresult PMDGetProfileMode(PMDAxisInterface axis_intf,
                            PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.ProfileMode = mode
mode = MagellanAxis.ProfileMode
```

see

MultiUpdate (p. 65), **Update** (p. 215)

SetPWMFrequency 0Ch
GetPWMFrequency 0Dh

Syntax

SetPWMFrequency *axis frequency*
GetPWMFrequency *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding		
axis	*Axis1*	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	**Range**	**Scaling**	**Units**
frequency	unsigned 16 bits	0 *to* $2^{16}-1$	$1/2^8$	kHz

Packet Structure

SetPWMFrequency

	0	*axis*	**0C**h
	15 12	11 8	7 0

Data

write	*frequency*
	15 0

GetPWMFrequency

	0	*axis*	**0D**h
	15 12	11 8	7 0

Data

read	*frequency*
	15 0

Description

SetPWMFrequency sets the PWM output frequency (in kHz) for the specified *axis*. To select one of the supported frequencies, pass the value listed in the SetPWMFrequency Value column as the *frequency* argument to this command.

Approximate Frequency	PWM bit Resolution	Actual Frequency	SetPWMFrequency Value
20 kHz	10	19.531 kHz	5,000
40 kHz	9	39.062 kHz	10,000
80 kHz	8	78.124 kHz	20,000

Atlas

These commands are relayed to an attached Atlas amplifier. Atlas supports 20 kHz, 40 kHz, and 80 kHz PWM frequencies.

Restrictions

Only 20 kHz and 80 kHz are currently supported by the Magellan motion control IC. Only 20 kHz and 40 kHz are supported in the ION products.

The PWM frequency can be changed only when motor output is disabled (e.g., immediately after power-up or reset).

C-Motion API

```
PMDresult PMDSetPWMFrequency(PMDAxisInterface axis_intf,
                             PMDuint16 frequency)
PMDresult PMDGetPWMFrequency(PMDAxisInterface axis_intf,
                             PMDuint16* frequency)
```

VB-Motion API

```
Dim frequency as Short
MagellanAxis.PWMFrequency = frequency
frequency = MagellanAxis.PWMFrequency
```

see

SetOutputMode (p. 158)

SetSampleTime 3Bh
GetSampleTime 3Ch

Syntax

SetSampleTime *time*
GetSampleTime

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range	Units
time	unsigned 32 bits	51 *to* 2^{20}	microseconds

Packet Structure

SetSampleTime

	15 … 8	7 … 0
	0	**3B**h

First data word

	31 … 16
write	*time* (high-order part)

Second data word

	15 … 0
write	*time (low-order part)*

GetSampleTime

	15 … 8	7 … 0
	0	**3C**h

First data word

	31 … 16
read	*time* (high-order part)

Second data word

	15 … 0
read	*time* (low-order part)

Description

SetSampleTime sets the time basis for the motion control IC. This time basis determines the trajectory update rate for all motor types as well as the servo loop calculation rate for DC brush and brushless DC motors. It does not, however, determine the commutation rate of the brushless DC motor types, nor the PWM or current loop rates for any motor type.

The ***time*** value is expressed in microseconds. The motion control IC hardware can adjust the cycle time only in increments of 51.2 microseconds; the ***time*** value passed to this command will be rounded to the nearest multiple of this base value.

Minimum cycle time depends on the product and number of enabled axes as follows:

# Enabled Axes	Minimum Cycle Time	Cycle Time w/ Trace Capture	Time per Axis	Maximum Cycle Frequency
1 (ION)	102.4 µs	102.4 µs	102.4 µs	9.76 kHz
1 MC58113	51.2 µs	51.2 µs	51.2 µs	19.53 kHz
1 (Magellan Single-axis)	51.2 µs	102.4 µs	51.2 µs	19.53 kHz (9.76 w/ trace capture)
1 (Magellan Multi-axis)	102.4 µs	102.4 µs	102.4 µs	9.76 kHz
2 (Magellan)	153.6 µs	153.6 µs	76.8 µs	6.51 kHz
3 (Magellan)	204.8 µs	204.8 µs	68.3 µs	4.88 kHz
4 (Magellan)	256 µs	256 µs	64 µs	3.91 kHz

Description (cont.)

Using the trace feature on single axis Magellan products with the sample time set to 51.2 µs will result in unexpected behavior.

GetSampleTime returns the value of the sample time.

Restrictions

This command affects the cycle time for all axes on multi-axis configurations.

This command cannot be used to set a sample time lower than the required minimum cycle time for the current configuration. Attempting to do so will set the sample time to the required minimum cycle time as specified in the previous table.

C-Motion API

```
PMDresult PMDSetSampleTime(PMDAxisInterface axis_intf,
                           PMDuint32 time)
PMDresult PMDGetSampleTime(PMDAxisInterface axis_intf,
                           PMDuint32* time)
```

VB-Motion API

```
Dim time as Long
MagellanAxis.SampleTime = time
time = MagellanAxis.SampleTime
```

see

SetSerialPortMode 8Bh
GetSerialPortMode 8Ch

Syntax

SetSerialPortMode *mode*
GetSerialPortMode

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Encoding
mode	unsigned 16 bits	see below

Packet Structure

SetSerialPortMode

0	*axis*	**8B**h
15	8	7 0

Data

	multi-drop address	0	protocol	stop bits	parity	transmission rate
write	15 11	10 9	8 7	6	5 4	3 0

GetSerialPortMode

0	*axis*	**8C**h
15	8	7 0

Data

	multi-drop address	0	protocol	stop bits	parity	transmission rate
read	15 11	10 9	8 7	6	5 4	3 0

Description

SetSerialPortMode sets the configuration for the asynchronous serial port. It configures the timing and framing of the serial port on the unit, regardless of whether RS-232 or RS-485 voltage levels are being used. The response to this command will use the serial port settings in effect before the command is executed, for example, transmission rate and parity. The new serial port settings must be used for the next command.

GetSerialPortMode returns the configuration for the asynchronous serial port, regardless of whether RS-232 or RS-485 voltage levels are being used.

The following table shows the encoding of the data used by this command.

Bit Number	Name	Instance	Encoding
0–3	Transmission Rate	1200 baud	0
		2400 baud	1
		9600 baud	2
		19200 baud	3
		57600 baud	4
		115200 baud	5
		230400 baud	6
		460800 baud	7
4–5	Parity	none	0
		odd	1
		even	2
6	Stop Bits	1	0
		2	1
7–8	Protocol	Point-to-point	0
		Multi-drop using idle-line detection	1
		— (Reserved)	2
		— (Reserved)	3
11–15	Multi-Drop Address	Address 0	0
		Address 1	1
			...
		Address 31	31

Restrictions

C-Motion API

```
PMDresult PMDSetSerialPortMode(PMDAxisInterface axis_intf,
                               PMDuint8 baud,
                               PMDuint8 parity,
                               PMDuint8 stopBits,
                               PMDuint8 protocol,
                               PMDuint8 multiDropID)
PMDresult PMDGetSerialPortMode(PMDAxisInterface axis_intf,
                               PMDuint8* baud,
                               PMDuint8* parity,
                               PMDuint8* stopBits,
                               PMDuint8* protocol,
                               PMDuint8* multiDropID)
```

VB-Motion API

```
CommunicationSerial.SerialPortModeSet( [in] baud,
                                       [in] parity,
                                       [in] stopBits,
                                       [in] protocol,
                                       [in] multidropID )
```

see

SetSettleTime AAh
GetSettleTime ABh

Syntax

SetSettleTime *axis time*
GetSettleTime *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
time	unsigned 16 bits	0 *to* 2^{16}–1	unity	cycles

Packet Structure

SetSettleTime

0	*axis*	**AA**h
15 12	11 8	7 0

Data

write	*time*
	15 0

GetSettleTime

0	*axis*	**AB**h
15 12	11 8	7 0

Data

read	*time*
	15 0

Description

SetSettleTime sets the time, in number of cycles, that the specified ***axis*** must remain within the settle window before the Axis Settled indicator in the Activity Status register is set.

GetSettleTime returns the value of the settle time for the specified ***axis***.

Restrictions

C-Motion API

```
PMDresult PMDSetSettleTime(PMDAxisInterface axis_intf,
                           PMDuint16 time)
PMDresult PMDGetSettleTime(PMDAxisInterface axis_intf,
                           PMDuint16* time)
```

VB-Motion API

```
Dim time as Short
MagellanAxis.SettleTime = time
time = MagellanAxis.SettleTime
```

see

Set/GetMotionCompleteMode (p. 149), **Set/GetSettleWindow** (p. 185), **GetActivityStatus** (p. 25)

SetSettleWindow BCh
GetSettleWindow BDh

Syntax

SetSettleWindow *axis window*
GetSettleWindow *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
window	unsigned 16 bits	0 *to* 2^{16}–1	unity	counts

Packet Structure

SetSettleWindow

0	*axis*	**BC**h
15 12	11 8	7 0

Data

	window
write	15 0

GetSettleWindow

0	*axis*	**BD**h
15 12	11 8	7 0

Data

	window
read	15 0

Description

SetSettleWindow sets the position range within which the specified ***axis*** must remain for the duration specified by **SetSettleTime** before the Axis Settled indicator in the Activity Status register is set.

GetSettleWindow returns the value of the settle window.

Restrictions

C-Motion API

```
PMDresult PMDSetSettleWindow (PMDAxisInterface axis_intf,
                              PMDuint16 window)
PMDresult PMDGetSettleWindow (PMDAxisInterface axis_intf,
                              PMDuint16* window)
```

VB-Motion API

```
Dim window as Short
MagellanAxis.SettleWindow = window
window = MagellanAxis.SettleWindow
```

see

Set/GetMotionCompleteMode (p. 149), **Set/GetSettleTime** (p. 184), **GetActivityStatus** (p. 25)

SetSignalSense A2h
GetSignalSense A3h

Syntax

SetSignalSense *axis sense*
GetSignalSense *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Indicator	Encoding	Bit Number
sense	*EncoderA*	0001h	0
	EncoderB	0002h	1
	Encoder Index	0004h	2
	Capture Input	0008h	3
	Positive Limit	0010h	4
	Negative Limit	0020h	5
	AxisIn	0040h	6
	HallA	0080h	7
	HallB	0100h	8
	HallC	0200h	9
	AxisOut	0400h	10
	Step Output/SPI Enable	0800h	11
	Motor Direction	1000h	12
	— (Reserved)		13–15

Packet Structure

SetSignalSense

15–12	11–8	7–0
0	*axis*	**A2**h

Data

	15–13	12–0
write	0	*sense*

GetSignalSense

15–12	11–8	7–0
0	*axis*	**A3**h

Data

	15–13	12–0
read	0	*sense*

Description

SetSignalSense establishes the sense of the corresponding bits of the Signal Status register, with the addition of ***Step Output*** and ***Motor Direction***, for the specified ***axis***.

For ***Encoder Index***, if the sense bit is 1, an index will be recognized for use in index-based phase correction if the index is high. For MC58113, if the sense bit is 1, an index capture will happen on a rising edge of the index signal, otherwise on a falling edge. This is true for index phase correction as well.

For the ***Capture Input***, if the sense bit is 1, a capture will occur on a low-to-high signal transition. Otherwise, a capture will occur on a high-to-low transition. For MC58113 this bit applies only to capture on the home signal, bit 2 applies to capture on the index signal.

SetSignalSense (cont.) A2h
GetSignalSense A3h

Description (cont.)

For *Positive Limit* and *Negative Limit*: if the sense bit is 1, an overtravel condition will occur if the signal is high. Otherwise, an overtravel condition will occur when the signal is low.

The AxisOut signal is inverted if the sense bit is set to one; otherwise it is not inverted.

When the Step Output/SPI Enable bit is set to 1, a step will be generated by the motion control IC with a low-to-high transition on the Pulse signal. Otherwise, a step will be generated by the motion control IC with a high-to-low transition on the Pulse signal.

For non-MC58113 motion control ICs, the same bit is used to control the sense of the ***SPI Enable*** signal, either in SPI DAC or in Atlas SPI output mode. When the bit is set the ***Enable*** signal will be held low when addressing the SPI output device, otherwise it will be held high. When driving an Atlas amplifier this bit must be set. Setting the Motor Direction bit has the effect of swapping the sense of positive and negative motor movement.

For MC58113, the signal sense for SPI Enabler is active low for Atlas and active high for SPI DAC.

GetSignalSense returns the value of the Signal Sense mask.

Atlas

No additional Atlas communication is performed for these commands. Atlas communication will fail if bit 11 is not properly set.

Restrictions

In ION products, FaultOut and /Enable exist in the Signal Status register, but their sense is not controllable.

In ION products, when the Capture Source is set to Encoder Index, only the Encoder Index bit of signal sense should be used to configure its polarity. The Capture Input bit of Signal Sense should always be cleared to zero (0) in this case.

Not all bits are implemented for all products. See the product user guide.

For Atlas these signals are not included in the Magellan signal status register.

C-Motion API

```
PMDresult PMDSetSignalSense(PMDAxisInterface axis_intf,
                            PMDuint16 sense)
PMDresult PMDGetSignalSense(PMDAxisInterface axis_intf,
                            PMDuint16* sense)
```

VB-Motion API

```
Dim sense as Short
MagellanAxis.SignalSense = sense
sense = MagellanAxis.SignalSense
```

see

GetSignalStatus (p. 55)

SetSPIMode 0Ah
GetSPIMode 0Bh

Syntax

SetSPIMode *mode*
GetSPIMode

Motor Types

DC Brush			

Arguments

Name	Instance	Encoding
mode	*RisingEdge*	0
	RisingEdgeDelay	1
	FallingEdge	2
	FallingEdgeDelay	3

Packet Structure

SetSPIMode

	0	**0A**h
	15 8	7 0

Data

write	0	*mode*
	15 2	1 0

GetSPIMode

	0	**0B**h
	15 8	7 0

Data

read	0	*mode*
	15 2	1 0

Description

SetSPIMode configures the communication settings for the motion control IC's SPI (Serial Peripheral Interface) DAC output port. Data is output as a series of 16-bit data words transmitted at 10 Mbps. The ***mode*** parameter controls the data clocking scheme as shown in the following table.

Mode	Encoding	Description
RisingEdge	0	Rising edge without phase delay: The SPIClock signal is inactive low. The SPIXmt pin transmits data on the rising edge of the SPIClock signal.
RisingEdgeDelay	1	Rising edge with phase delay: The SPIClock signal is inactive low. The SPIXmt pin transmits data one half-cycle ahead of the rising edge of the SPIClock signal.
FallingEdge	2	Falling edge without phase delay: The SPIClock signal is inactive high. The SPIXmt pin transmits data on the falling edge of the SPIClock signal.
FallingEdgeDelay	3	Falling edge with phase delay: The SPIClock signal is inactive high. The SPIXmt pin transmits data one half-cycle ahead of the falling edge of the SPIClock signal.

Atlas

No additional Atlas communication is performed for these commands. When using Atlas output the SPI mode must be zero.

Restrictions

SPI output is only available when the motor type is **DC brush**, and only in some products. See the product user guide.

C-Motion API

```
PMDresult PMDSetSPIMode(PMDAxisInterface axis_intf, PMDuint16 mode)
PMDresult PMDGetSPIMode(PMDAxisInterface axis_intf, PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanObject.SPIMode = mode
mode = MagellanObject.SPIMode
```

see

SetOutputMode (p. 158)

SetStartVelocity 6Ah
GetStartVelocity 6Bh

Syntax

SetStartVelocity *axis velocity*
GetStartVelocity *axis*

Motor Types

▨	▨	**Microstepping**	**Pulse & Direction**

Arguments

Name	Instance	Encoding		
axis	*Axis1*	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	**Range**	**Scaling**	**Units**
velocity	unsigned 32 bits	0 *to* 2^{31}–1	$1/2^{16}$	steps/cycle microsteps/cycle

Packet Structure

SetStartVelocity

	0	*axis*	**6A**h
	15 – 12	11 – 8	7 – 0

First data word

write	*velocity* (high-order part)
	31 – 16

Second data word

write	*velocity* (low-order part)
	15 – 0

GetStartVelocity

	0	*axis*	**6B**h
	15 – 12	11 – 8	7 – 0

First data word

read	*velocity* (high-order part)
	31 – 16

Second data word

read	*velocity* (low-order part)
	15 – 0

Description

SetStartVelocity loads the starting velocity register for the specified ***axis***. The start velocity is the instantaneous velocity at the start and at the end of the profile.

GetStartVelocity reads the value of the starting velocity register.

Scaling example: To load a starting velocity value of 1.750 steps/cycle multiply by 65,536 (giving 114,688) and load the resultant number as a 32-bit number, giving 0001 in the high word and C000h in the low word. Values returned by **GetStartVelocity** must correspondingly be divided by 65,536 to convert them to units of counts/cycle.

Restrictions

SetStartVelocity is only used in the Velocity Contouring and Trapezoidal profile modes.

C-Motion API

```
PMDresult PMDSetStartVelocity(PMDAxisInterface axis_intf,
                              PMDuint32 velocity)
PMDresult PMDGetStartVelocity(PMDAxisInterface axis_intf,
                              PMDuint32* velocity)
```

VB-Motion API

```
Dim velocity as Long
MagellanAxis.StartVelocity = velocity
velocity = MagellanAxis.StartVelocity
```

see

Set/GetVelocity (p. 213), **Set/GetAcceleration** (p. 83), **Set/GetDeceleration** (p. 122), **Set/GetPosition** (p. 172)

SetStepRange CFh
GetStepRange CEh

Syntax

SetStepRange *axis range*
GetStepRange *axis*

Motor Types

			Pulse & Direction

Arguments

Name	**Instance**	**Encoding**
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
range	*0–4.98 Msteps/sec*	1
	0–622.5 ksteps/sec	4
	0–155.6 ksteps/sec	6
	0–38906 steps/sec	8

Packet Structure

SetStepRange

0	*axis*	**CF**h
15 12	11 8	7 0

Data

write	0	*range*
	15 4	3 0

GetStepRange

0	*axis*	**CE**h
15 12	11 8	7 0

Data

read	0	*range*
	15 4	3 0

Description

SetStepRange sets the maximum pulse rate frequency for the specified *axis*. For example, if the desired maximum pulse rate is 200,000 pulses/second, the **SetStepRange 6** command should be issued.

GetStepRange returns the maximum pulse rate frequency for the specified *axis*.

Restrictions

The MC55110 and the MC58110 have a maximum step range of 100 Kstep/s, which cannot be changed. The MC58113 has a maximum step range of 1 Mstep/s which cannot be changed.

SetStepRange must be called before any moves are made, and must not be called after any moves have been made.

C-Motion API

```
PMDresult PMDSetStepRange(PMDAxisInterface axis_intf,
                          PMDuint16 range)
PMDresult PMDGetStepRange(PMDAxisInterface axis_intf,
                          PMDuint16* range)
```

VB-Motion API

```
Dim range as Short
MagellanAxis.StepRange = range
range = MagellanAxis.StepRange
```

see

SetStopMode buffered D0h
GetStopMode D1h

Syntax

SetStopMode *axis mode*
GetStopMode *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*No Stop*	0
	Abrupt Stop	1
	Smooth Stop	2

Packet Structure

SetStopMode

0	*axis*	**D0**h
15 12	11 8	7 0

Data

write	0	*mode*
	15 2	1 0

GetStopMode

0	*axis*	**D1**h
15 12	11 8	7 0

Data

read	0	*mode*
	15 2	1 0

Description

SetStopMode stops the specified *axis*. The available stop modes are *Abrupt Stop*, which instantly (without any deceleration phase) stops the axis; *Smooth Stop,* which uses the programmed deceleration value and profile shape for the current profile mode to stop the axis; or *No Stop*, which is generally used to turn off a previously issued set stop command.

Note: After an **Update**, a buffered stop command (**SetStopMode** command) will reset to the *No Stop* condition. In other words, if the **SetStopMode** command is followed by an **Update** command and then by a **GetStopMode** command, the retrieved stop mode will be *No Stop*.

GetStopMode returns the value of the stop mode.

Restrictions

Smooth Stop mode is not available in the Electronic Gear profile mode.

SetStopMode is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetStopMode(PMDAxisInterface axis_intf, PMDuint16 mode)
PMDresult PMDGetStopMode(PMDAxisInterface axis_intf, PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.StopMode = mode
mode = MagellanAxis.StopMode
```

see

MultiUpdate (p. 65), **Update** (p. 215)

SetSynchronizationMode F2h
GetSynchronizationMode F3h

Syntax

SetSynchronizationMode *mode*
GetSynchronizationMode

Motor Types

DC Brush	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
mode	*Disabled*	0
	Master	1
	Slave	2

Packet Structure

SetSynchronizationMode

	0	**F2h**
	15 8	7 0

Data

write	0	*mode*
	15 2	1 0

GetSynchronizationMode

	0	**F3h**
	15 8	7 0

Data

read	0	*mode*
	15 2	1 0

Description

SetSynchronizationMode sets the mode of the pin used for the synchronization of the internal timer across multiple motion ICs. In the *Disabled* mode, the pin is configured as an input and is not used. In the *Master* mode, the pin outputs a synchronization pulse that can be used by slave nodes or other devices to synchronize with the internal chip cycle of the master node. In the *Slave* mode, the pin is configured as an input and a pulse on the pin synchronizes the internal chip cycle.

When the synchronization mode is set to either Master or Slave, the internal time counter will be set to zero. This feature is intended to allow synchronization of updates across processors by using time breakpoints.

GetSynchronizationMode returns the value of the synchronization mode.

Restrictions

If the motion control IC is configured as a slave, and any axis is configured for pulse & direction output, multi-chip synchronization cannot be used.

Multichip synchronization is not supported in all products. See the product user guide.

C-Motion API

```
PMDresult PMDSetSynchronizationMode(PMDAxisInterface axis_intf,
                                    PMDuint16 mode)
PMDresult PMDGetSynchronizationMode(PMDAxisInterface axis_intf,
                                    PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanObject.SynchronizationMode = mode
mode = MagellanObject.SynchronizationMode
```

see

GetTime (p. 58), **SetBreakPoint** (p. 94)

SetTraceMode B0h
GetTraceMode B1h

Syntax

SetTraceMode *mode*
GetTraceMode

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
mode	*16-bit unsigned*	see below

Packet Structure

SetTraceMode

15 … 8	7 … 0
0	**B0**h

Data

	15 … 0
write	*mode*

GetTraceMode

15 … 8	7 … 0
0	**B1**h

Data

	15 … 0
read	*mode*

Description

SetTraceMode sets the behavior for the next trace. Mode is a bitmask, as shown below:

Name	Bit
Wrap Mode	0
- (Reserved)	1-7
Trigger Mode	8
- (Reserved)	9-15

Wrap mode may be either One Time (zero), or Rolling Buffer (one). In One Time mode, the trace continues until the trace buffer is filled, then stops. In Rolling Buffer mode, the trace continues from the beginning of the trace buffer after the end is reached. When in rolling mode, values stored at the beginning of the trace buffer are lost if they are not read before being overwritten by the wrapped data.

Trigger mode may be either Internal (zero), or External (one). This mode is used to control tracing on attached Atlas amplifiers. In Internal trigger mode the trace bit in all Atlas torque commands will be set whenever Magellan trace is active. In this mode Atlas should be configured to use its own internal trace period to time trace samples. In External mode the trace bit in all Atlas torque commands will be set exactly once each time Magellan stores a trace sample, and clear at other times. In this mode Atlas should be configured to use its external trigger mode to synchronize sampling with Magellan.

GetTraceMode returns the value for the trace mode.

Atlas

No additional Atlas communication is performed for these commands, but the Atlas trace mode, and other trace parameters may have to be set by addressing an Atlas amplifier directly. See *Atlas Digital Amplifier Complete Technical Reference* for more detail.

Restrictions

C-Motion API

```
PMDresult PMDSetTraceMode(PMDAxisInterface axis_intf, PMDuint16 mode)
PMDresult PMDGetTraceMode(PMDAxisInterface axis_intf, PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanObject.TraceMode = mode
mode = MagellanObject.TraceMode
```

see

GetTraceStatus (p. 60)

SetTracePeriod B8h
GetTracePeriod B9h

Syntax

SetTracePeriod *period*
GetTracePeriod

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range	Scaling	Units
period	unsigned 16 bits	1 *to* 2^{16}–1	unity	cycles

Packet Structure

SetTracePeriod

15 … 8	7 … 0
0	**B8h**

Data

	15 … 0
write	*period*

GetTracePeriod

15 … 8	7 … 0
0	**B9h**

Data

	15 … 0
read	*period*

Description

SetTracePeriod sets the interval between contiguous trace captures. For example, if the trace period is set to one, trace data will be captured at the end of every chip cycle. If the trace period is set to two, trace data will be captured at the end of every second chip cycle, and so on.

GetTracePeriod returns the value for the trace period.

Atlas

No additional Atlas communication is performed for these commands, but Atlas trace parameters may have to be set by addressing an Atlas amplifier directly. Atlas trace may be synchronized to Magellan trace by using the "external trigger" trace mode, which is done using the trace bit in each Atlas torque command. See *Atlas Digital Amplifier Complete Technical Reference* for more detail.

Restrictions

C-Motion API

```
PMDresult PMDSetTracePeriod(PMDAxisInterface axis_intf,
                            PMDuint16 period);
PMDresult PMDGetTracePeriod(PMDAxisInterface axis_intf,
                            PMDuint16* period)
```

VB-Motion API

```
Dim period as Short
MagellanObject.TracePeriod = period
period = MagellanObject.TracePeriod
```

see

Set/GetSampleTime (p. 180), **Set/GetTraceStart** (p. 196), **Set/GetTraceStop** (p. 199)

SetTraceStart B2h
GetTraceStart B3h

Syntax

SetTraceStart *triggerAxis_condition_triggerBit_triggerState*
GetTraceStart

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
triggerAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
condition	*Immediate*	0
	Next Update	1
	Event Status	2
	Activity Status	3
	Signal Status	4
	Drive Status	5
triggerBit	*Status Register Bit*	0 *to* 15
triggerState (tS)	*Triggering State of the Bit*	0 (value = 0) 1 (value = 1)

Packet Structure

SetTraceStart

0	**B2**h
15 8	7 0

Data

write	0	*tS*	*triggerBit*	*condition*	*triggerAxis*
	15 13	12	11 8	7 4	3 0

GetTraceStart

0	**B3**h
15 8	7 0

Data

read	0	*tS*	*triggerBit*	*condition*	*triggerAxis*
	15 13	12	11 8	7 4	3 0

Description

SetTraceStart sets the condition for starting the trace. The *Immediate* condition requires no axis to be specified and the trace will begin upon execution of this instruction. The other four conditions require an axis to be specified, and when the condition for that axis is attained, the trace will begin.

When a status register bit is the trigger, the bit number and state must be included in the argument. The trace is started when the indicated bit reaches the specified state (0 or 1).

GetTraceStart returns the value of the trace-start trigger.

Once a trace has started, the trace-start trigger is reset to zero (0).

Description (cont.)

The following table shows the corresponding value for combinations of *triggerBit* and *register*0.

TriggerBit	Event Status Register	Activity Status Register	Signal Status Register	Drive Status Register
0	Motion Complete	Phasing Initialized	Encoder A	
1	Wrap-around	At Maximum Velocity	Encoder B	In Foldback
2	Breakpoint 1	Tracking	Encoder Index	Overtemperature
3	Position Capture		Capture Input	Shunt Active
4	Motion Error		Positive Limit	In Holding
5	Positive Limit		Negative Limit	Overvoltage
6	Negative Limit		AxisIn	Undervoltage
7	Instruction Error	Axis Settled	Hall Sensor A	Atlas Disabled
8	Disable	Motor mode	Hall Sensor B	
9	Overtemperature Fault	Position Capture	Hall Sensor C	
0Ah	Drive Exception	In Motion		
0Bh	Commutation Error	In Positive Limit		
0Ch	Current Foldback	In Negative Limit		Clipping
0Dh			/Enable Input	
0Eh	Breakpoint 2		FaultOut	
0Fh				Atlas not connected

Examples:

If it is desired that the trace begin on the next **Update** for axis 3, then a 2 is set for the axis number, a 1 is set for the condition, and bit number and state can be loaded with zeroes since they are not used. The actual data word sent to the motor processor in this case is 0012h.

If it is desired that the trace begin when bit 7 of the Activity Status register for axis 2 goes to 0, then the trace start is loaded as follows: A 1 is loaded for axis number, a 3 is loaded for condition, a 7 is loaded for bit number, and a 0 is loaded for state. The actual data word sent to the motor processor is 0731h.

Atlas

No additional Atlas communication is performed for these commands, but Atlas trace parameters may have to be set by addressing an Atlas amplifier directly. Magellan trace start is signaled to Atlas by using the trace bit in each Atlas torque command, See *Atlas Digital Amplifier Complete Technical Reference* for more detail.

Restrictions

Not all trace start conditions are available in all products. See the product user guide.

C-Motion API

```
PMDresult PMDSetTraceStart(PMDAxisInterface axis_intf,
                           PMDAxis traceAxis,
                           PMDuint8 condition,
                           PMDuint8 triggerBit,
                           PMDuint8 triggerState)
PMDresult PMDGetTraceStart(PMDAxisInterface axis_intf,
                           PMDAxis* traceAxis,
                           PMDuint8* condition,
                           PMDuint8* triggerBit,
                           PMDuint8* triggerState)
```

VB-Motion API

```
MagellanObject.TraceStartSet( [in] triggerAxis,
                              [in] condition,
                              [in] triggerBit,
                              [in] triggerState )
MagellanObject.TraceStartGet( [out] triggerAxis,
                              [out] condition,
                              [out] triggerBit,
                              [out] triggerState )
```

see

Set/GetBufferLength (p. 101), **GetTraceCount** (p. 59), **Set/GetTraceMode** (p. 193), **Set/GetTracePeriod** (p. 195), **Set/GetTraceStop** (p. 199)

SetTraceStop **B4**h
GetTraceStop **B5**h

Syntax

SetTraceStop *triggerAxis_condition_triggerBit_triggerState*
GetTraceStop

Motor Types

DC Brush	**Brushless DC**	**Microstepping**	**Pulse & Direction**

Arguments

Name	**Instance**	**Encoding**
triggerAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
condition	*Immediate*	0
	Next Update	1
	Event Status	2
	Activity Status	3
	Signal Status	4
	Drive Status	5
triggerBit	*Status Register Bit*	0 *to* 15
triggerState (tS)	*Triggering State of the Bit*	0 (value = 0) 1 (value = 1)

Packet Structure

SetTraceStop

0	**B4**h
15 8	7 0

Data

write	0	*tS*	*triggerBit*	*condition*	*triggerAxis*
	15 13	12	11 8	7 4	3 0

GetTraceStop

0	**B5**h
15 8	7 0

Data

read	0	*tS*	*triggerBit*	*condition*	*triggerAxis*
	15 13	12	11 8	7 4	3 0

Description

SetTraceStop sets the condition for stopping the trace. The *Immediate* condition requires no axis to be specified and the trace will stop upon execution of this instruction. The other four conditions require an axis to be specified, and when the condition for that axis is attained, the trace will stop.

When a status register bit is the trigger, the bit number and state must be included in the argument. The trace is stopped when the indicated bit reaches the specified state (0 or 1).

GetTraceStop returns the value of the trace-stop trigger.

Once a trace has stopped, the trace-stop trigger is reset to zero (0).

SetTraceStop (cont.) B4h
GetTraceStop B5h

Description (cont.)

The following table shows the corresponding value for combinations of *triggerBit* and *register*.

TriggerBit	Event Status Register	Activity Status Register	Signal Status Register	Drive Status Register
0	Motion Complete	Phasing Initialized	Encoder A	
1	Wrap-around	At Maximum Velocity	Encoder B	In Foldback
2	Breakpoint 1	Tracking	Encoder Index	Overtemperature
3	Position Capture		Capture Input	Shunt Active
4	Motion Error		Positive Limit	In Holding
5	Positive Limit		Negative Limit	Overvoltage
6	Negative Limit		AxisIn	Undervoltage
7	Instruction Error	Axis Settled	Hall Sensor A	Atlas Disabled
8	Disable	Motor mode	Hall Sensor B	
9	Overtemperature Fault	Position Capture	Hall Sensor C	
0Ah	Drive Exception	In Motion		
0Bh	Commutation Error	In Positive Limit		
0Ch	Current Foldback	In Negative Limit		Clipping
0Dh			/Enable Input	
0Eh	Breakpoint 2		FaultOut	
0Fh				Atlas not connected

Examples:

If it is desired that the trace ends on the next **Update** for axis 3, then a 2 is set for the axis number, a 1 is set for the condition, and bit number and state can be loaded with zeroes since they are not used. The actual data word sent to the motor processor in this case is 0012h.

If it is desired that the trace ends when bit 7 of the Activity Status register for axis 2 goes to 0, then the trace stop is loaded as follows: A 1 is loaded for axis number, a 3 is loaded for condition, a 7 is loaded for bit number, and a 0 is loaded for state. The actual data word sent to the motor processor in this case is 0731h.

Atlas

No additional Atlas communication is performed for these commands, but Atlas trace parameters may have to be set by addressing an Atlas amplifier directly. Magellan trace stop is signaled to Atlas by using the trace bit in each Atlas torque command, See *Atlas Digital Amplifier Complete Technical Reference* for more detail.

Restrictions

Not all trace stop conditions are available in all products. See the product user guide.

C-Motion API

```
PMDresult PMDSetTraceStop(PMDAxisInterface axis_intf,
                          PMDAxis traceAxis,
                          PMDuint8 condition,
                          PMDuint8 triggerBit,
                          PMDuint8 triggerState)
PMDresult PMDGetTraceStop(PMDAxisInterface axis_intf,
                          PMDAxis* traceAxis,
                          PMDuint8* condition,
                          PMDuint8* triggerBit,
                          PMDuint8* triggerState)
```

VB-Motion API

```
MagellanObject.TraceStopSet( [in] triggerAxis,
                             [in] condition,
                             [in] triggerBit,
                             [in] triggerState )
MagellanObject.TraceStopGet( [out] triggerAxis,
                             [out] condition,
                             [out] triggerBit,
                             [out] triggerState )
```

see **GetTraceCount** (p. 59), **Set/GetTraceStart** (p. 196), **GetTraceStatus** (p. 60)

SetTraceVariable B6h
GetTraceVariable B7h

Syntax

SetTraceVariable *variableNumber traceAxis_variableID*
GetTraceVariable *variableNumber*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

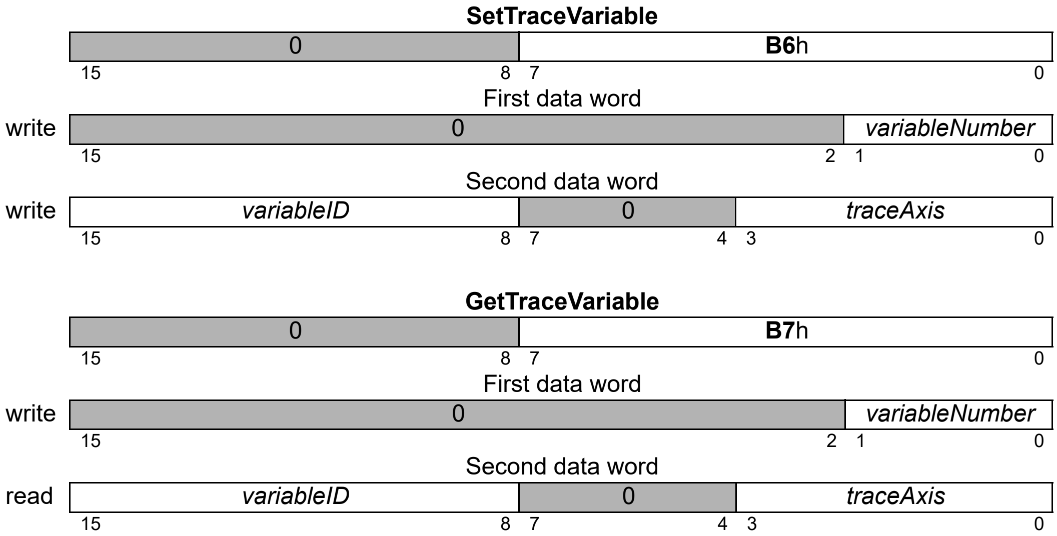
Name		Instance	Encoding
variableNumber		*Variable1*	0
		Variable2	1
		Variable3	2
		Variable4	3
traceAxis		*Axis1*	0
		Axis2	1
		Axis3	2
		Axis4	3
variableID			
	Trajectory Generator	*Commanded Position*	2
		Commanded Velocity	3
		Commanded Acceleration	4
	Encoder	*Actual Position*	5
		Actual Velocity	6
		Position Capture Register	9
		Phase Angle	15
		Phase Offset	16
		Raw Encoder Reading	84
		Encoder sin raw reading	111
		Encoder cos raw reading	112
		Encoder sin corrected reading	113
		Encoder cos corrected reading	114
		Encoder sin/cos angle	115
		Encoder sin/cos digital count	116
	Position Loop	*Position Error*	1
		Position Loop Integrator Sum	10
		Position Loop Integrator Contribution	57
		Position Loop Derivative	11
		Biquad1 Input	64
		Biquad2 Input	65
	Status Registers	*Event Status Register*	12
		Activity Status Register	13
		Signal Status Register	14
		Drive Status Register	56
		Drive Fault Status Register	79
	Commutation/Phasing	*Active Motor Command*	7
		Phase A Command	17
		Phase B Command	18
		Phase C Command	19
		Phase Angle Scaled	29

SetTraceVariable (cont.) **B6**h
GetTraceVariable **B7**h

Arguments (cont.)	*Current Loops*	*Phase A Reference*	66
		Phase A Error	30
		Phase A Actual Current	31
		Phase A Integrator Sum	32
		Phase A Integrator Contribution	33
		Current Loop A Output	34
		Phase B Reference	67
		Phase B Error	35
		Phase B Actual Current	36
		Phase B Integrator Sum	37
		Phase B Integrator Contribution	38
		Current Loop B Output	39
		D Feedback	40
		Q Feedback	48
		Leg A Current	69
		Leg B Current	70
		Leg C Current	71
		Leg D Current	72
	Field Oriented Control	*D Reference*	40
		D Error	41
		D Feedback	42
		D Integrator Sum	43
		D Integrator Contribution	44
		D Output	45
		Q Reference	46
		Q Error	47
		Q Feedback	48
		Q Integrator Sum	49
		Q Integrator Contribution	50
		Q Output	51
		FOC Alpha Output	52
		FOC Beta Output	53
		Phase Alpha Actual Current	73
		Phase Beta Actual Current	74
	Motor Output	*Bus Voltage*	54
		Temperature	55
		Foldback Energy	68
		Bus Current Supply	86
		Bus Current Return	87
		PWM Output A	75
		PWM Output B	76
		PWM Output C	77
	Analog Inputs	*Analog Input0*	20
		Analog Input1	21
		Analog Input2	22
		Analog Input3	23
		Analog Input4	24
		Analog Input5	25
		Analog Input6	26
		Analog Input7	27
	Miscellaneous	*None* (disable variable)	0
		Motion Control IC Time	8

SetTraceVariable (cont.) B6h
GetTraceVariable B7h

Packet Structure



Description

SetTraceVariable assigns the given variable to the specified *variableNumber* location in the trace buffer. Up to four variables may be traced at one time.

All variable assignments must be contiguous starting with *variableNumber* = 0.

GetTraceVariable returns the variable and axis of the specified *variableNumber*.

Example: To set up a three variable trace capturing the commanded acceleration for axis 1, the actual position for axis 1, and the event status word for axis 3, the following sequence of commands would be used. First, a **SetTraceVariable** command with *variableNumber* of 0, *axis* of 0, and *variableID* of 4 would be sent. Then, a **SetTraceVariable** command with *variableNumber* of 1, *axis* of 0, and *variableID* of 5 would be sent. Finally, a **SetTraceVariable** command with a *variableNumber* of 3, *axis* of 2 and *variableID* of 0h would be sent.

The table below summarizes the data type and scaling factor for the trace variables supported by Magellan. Note that all values are actually stored in the trace buffer or returned by **GetTraceValue** as 32 bit quantities. If the data type is "16 bit signed" then the data will be sign-extended to 32 bits. If the data type is "16 bit unsigned" then the high word will be zero.

Variable	Encoding	Type	Scaling	Units/Notes
Command Source				
Commanded Position	2	signed 32 bit	unity	counts or microsteps
Commanded Velocity	3	signed 32 bit	$1/2^{16}$	counts/cycle or microsteps/cycle
Commanded Acceleration	4	signed 32 bit	$1/2^{16}$	counts/cycle^2 or microsteps/cycle^2

Description (cont.)

Variable	Encoding	Type	Scaling	Units/Notes
Encoder				
Actual Position	5	signed 32 bit	unity	counts or microsteps
Capture Value	9	signed 32 bit	unity	counts or microsteps
Actual Velocity (not smoothed)	83	signed 32 bit	unity	counts/cycle or microsteps/cycle
Raw Encoder Reading	84	signed 32 bit	unity	counts
Encoder sin raw reading	111	unsigned 16 bit	$100/2^{16}$	% full scale
Encoder cos raw reading	112	unsigned 16 bit	$100/2^{16}$	% full scale
Encoder sin corrected reading	113	signed 16 bit	$100/2^{15}$	% full scale
Encoder cos corrected reading	114	signed 16 bit	$100/2^{15}$	% full scale
Encoder sin/cos angle	115	unsigned 16 bit	360/16384	degrees
Encoder sin/cos digital count	116	unsigned 32 bit	unity	counts
Position Loop				
Position Error	1	signed 32 bit	unity	counts or microsteps
Position Loop Integrator Sum	10	signed 32 bit	$100K_{out}/2^{39}$	% output
Position Loop Derivative	11	signed 32 bit	$100K_{out}/2^{16}$	% output
Position Loop Integration Contribution	57	signed 32 bit	$100K_{out}/2^{31}$	% output (eg scaled velocity)
Biquad1 Input	64	signed 32 bit	$100/2^{15}$	% output
Biquad2 Input	65	signed 32 bit	$100/2^{15}$	% output
Status Registers				
Event Status	12	unsigned 16 bit	-	see **GetEventStatus**
Activity Status	13	unsigned 16 bit	-	see **GetActivityStatus**
Signal Status	14	unsigned 16 bit	-	see **GetSignalStatus**
Drive Status	56	unsigned 16 bit	-	see **GetDriveStatus**
Drive Fault Status	79	unsigned 16 bit	-	see **GetDriveFaultStatus**
Active Operating Mode	110	unsigned 16 bit	-	see **GetActiveOperating Mode**

Description (cont.)

Variable	Encoding	Type	Scaling	Units/Notes
Commutation/Phasing				
Active Motor Command	7	signed 16 bit	$100/2^{15}$	% output
Phase Angle	15	unsigned 32 bit	unity	counts or microsteps
Phase Offset	16	signed 32 bit	unity	counts
Phase A Command	17	signed 16 bit	$100/2^{15}$	% output
Phase B Command	18	signed 16 bit	$100/2^{15}$	% output
Phase C Command	19	signed 16 bit	$100/2^{15}$	% output
Phase Angle Scaled	29	unsigned 16 bit	$360/2^{15}$	degrees
Commutation Error	89	signed 32 bit	unity	counts (set during phase initialization or correction)
Commutation Error Cause	119	unsigned 16 bit		enumerated value, explanation below
Current Loops				
Phase A Reference	66	signed 16 bit	$100/2^{15}$	% full scale
Phase A Error	30	signed 16 bit	$100/2^{15}$	% full scale
Phase A Actual Current	31	signed 16 bit	$100/2^{15}$	% full scale
Phase A Integrator Sum	32	signed 16 bit	$100/2^{15}$	% full scale
Phase A Integrator Contribution	33	signed 16 bit	$100/2^{14}$	% full scale
Current Loop A Output	34	signed 16 bit	$100/2^{15}$	% output
Phase B Reference	67	signed 16 bit	$100/2^{15}$	% full scale
Phase B Error	30	signed 16 bit	$100/2^{15}$	% full scale
Phase B Actual Current	35	signed 16 bit	$100/2^{15}$	% full scale
Phase B Integrator Sum	36	signed 16 bit	$100/2^{15}$	% full scale
Phase B Integrator Contribution	37	signed 16 bit	$100/2^{14}$	% full scale
Current Loop B Output	39	signed 16 bit	$100/2^{15}$	% output
D Feedback	40	signed 16 bit	$100/2^{15}$	% full scale
Q Feedback	48	signed 16 bit	$100/2^{15}$	% full scale

Description (cont.)

Variable	Encoding	Type	Scaling	Units/Notes
Current Loops (cont.)				
Leg A Current	69	signed 16 bit	$100/2^{15}$	% full scale
Leg B Current	70	signed 16 bit	$100/2^{15}$	% full scale
Leg C Current	71	signed 16 bit	$100/2^{15}$	% full scale
Leg D Current	72	signed 16 bit	$100/2^{15}$	% full scale
Field Oriented Control				
D Reference	40	signed 16 bit	$100/2^{15}$	% full scale
D Error	41	signed 16 bit	$100/2^{15}$	% full scale
D Feedback	42	signed 16 bit	$100/2^{15}$	% full scale
D Integrator Sum	43	signed 16 bit	$100/2^{15}$	% full scale
D Integrator Contribution	44	signed 16 bit	$100/2^{14}$	% full scale
D Output	45	signed 16 bit	$100/2^{15}$	% output
Q Reference	46	signed 16 bit	$100/2^{15}$	% full scale
Q Error	47	signed 16 bit	$100/2^{15}$	% full scale
Q Feedback	48	signed 16 bit	$100/2^{15}$	% full scale
Q Integrator Sum	49	signed 16 bit	$100/2^{15}$	% full scale
Q Integrator Contribution	50	signed 16 bit	$100/2^{14}$	% full scale
Q Output	51	signed 16 bit	$100/2^{15}$	% output
FOC Alpha Output	52	signed 16 bit	$100/2^{15}$	% output
FOC Beta Output	53	signed 16 bit	$100/2^{15}$	% output
Phase A Actual Current	31	signed 16 bit	$100/2^{15}$	% full scale
Phase B Actual Current	35	signed 16 bit	$100/2^{15}$	% full scale

Description (cont.)

Variable	Encoding	Type	Scaling	Units/Notes
Motor Output				
Bus Voltage	54	unsigned 16 bit	$100/2^{16}$	% bus voltage analog input
Temperature	55	unsigned 16 bit	$100/2^{15}$	% temperature analog input
Foldback Energy	68	unsigned 32 bit	see note below	A^2s
PWM A Output	75	signed 16 bit	$100/2^{15}$	% max output
PWM B Output	76	signed 16 bit	$100/2^{15}$	% max output
PWM C Output	77	signed 16 bit	$100/2^{15}$	% max output
Bus Current Supply	86	signed 16 bit	$100/2^{15}$	% max bus current analog input
Bus Current Return	87	signed 16 bit	$100/2^{15}$	% max leg current analog input
Analog Inputs				
Analog Raw Channel 0	20	unsigned 16 bit	$100/2^{16}$	% input
Analog Raw Channel 1	21	unsigned 16 bit	$100/2^{16}$	% input
Analog Raw Channel 2	22	unsigned 16 bit	$100/2^{16}$	% input
Analog Raw Channel 3	23	unsigned 16 bit	$100/2^{16}$	% input
Analog Raw Channel 4	24	unsigned 16 bit	$100/2^{16}$	% input
Analog Raw Channel 5	25	unsigned 16 bit	$100/2^{16}$	% input
Analog Raw Channel 6	26	unsigned 16 bit	$100/2^{16}$	% input
Analog Raw Channel 7	27	unsigned 16 bit	$100/2^{16}$	% input
None	0	-	-	Terminates variable list
Motion Processor Time	8	unsigned 32 bit	unity	cycles

The foldback energy scaling factor is $t_c(i_{fs}/20480)^2 2^{15}$, where t_c is the current loop period of 51.2×10^{-6}s and i_{fs} is the actual current when a leg current sensor is at full scale. The full scale current depends on the product and, for ICs, on the current sense circuit. In all cases it is greater than the maximum current that is actually readable. Consult your product user guide for more information on scaling.

Description (cont.)

The Commutation Error Cause trace value indicates the reason for the first commutation error since the value was cleared. Reading the value, either with trace or by using **GetTraceValue**, clears it to zero. The error codes are:

Error Code	Encoding
No error	0
Phase correction too large	1
Invalid Hall state	2
— (Reserved)	3
Pulse phase initialization, signal/noise too low, or no movement	4
Pulse phase initialization, too much movement during ramp	5

The script inteface combines the traceAxis with the variableID in a single code argument as shown below. For example, to set the second trace variable to Active Motor Command (7) for axis 1 (0), code = 7*256 + 0 = 1792, so the command should be:

SetTraceVariable 1 1792

Atlas

No additional Atlas communication is performed for these commands, but Atlas trace parameters may have to be set by addressing an Atlas amplifier directly. See *Atlas Digital Amplifier Complete Technical Reference* for more detail.

Restrictions

When selecting *ActualVelocity* as a trace variable, the reported value is the change in position between trace captures.

In FOC (Field Oriented Control) mode, A/B current values are not meaningful. Select D/Q current values instead.

Not all trace variables are available in all products. See the product user guide.

C-Motion API

```
PMDresult PMDSetTraceVariable(PMDAxisInterface axis_intf,
                              PMDuint16 variableNumber,
                              PMDAxis traceAxis,
                              PMDuint8 variableID)
PMDresult PMDGetTraceVariable(PMDAxisInterface axis_intf,
                              PMDuint16 variableNumber,
                              PMDAxis* traceAxis,
                              PMDuint8* variableID)
```

VB-Motion API

```
MagellanObject.TraceVariableSet( [in] variableNumber,
                                 [in] traceAxis,
                                 [in] variableID )
MagellanObject.TraceVariableGet( [in] variableNumber,
                                 [out] traceAxis,
                                 [out] variableID )
```

see

SetTracePeriod (p. 195), **SetTraceStart** (p. 196), **SetTraceStop** (p. 199)

SetTrackingWindow A8h
GetTrackingWindow A9h

Syntax

SetTrackingWindow *axis window*
GetTrackingWindow *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
window	unsigned 16 bits	0 *to* 2^{16}–1	unity	counts

Packet Structure

SetTrackingWindow

0	*axis*	**A8**h
15 12	11 8	7 0

Data

	window
write	15 0

GetTrackingWindow

0	*axis*	**A9**h
15 12	11 8	7 0

Data

	window
read	15 0

Description

SetTrackingWindow sets boundaries for the position error of the specified *axis*. If the absolute value of the position error exceeds the tracking window, the tracking indicator (bit 2 of the Activity Status register) is set to 0. When the position error returns to within the window, the tracking indicator is set to 1.

GetTrackingWindow returns the value of the tracking window.

Restrictions

C-Motion API

```
PMDresult PMDSetTrackingWindow(PMDAxisInterface axis_intf,
                               PMDuint16 window)
PMDresult PMDGetTrackingWindow(PMDAxisInterface axis_intf,
                               PMDuint16* window)
```

VB-Motion API

```
Dim window as Short
MagellanAxis.TrackingWindow = window
window = MagellanAxis.TrackingWindow
```

see

GetActivityStatus (p. 25), **GetActualPosition** (p. 85)

SetUpdateMask F9h
GetUpdateMask FAh

Syntax

SetUpdateMask *axis mask*
GetUpdateMask *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
	Type	**Scaling**
mask	unsigned 16 bit	bitmask

Packet Structure

SetUpdateMask

15–12	11–8	7–0
0	*axis*	**F9**h

First data word

	15–0
write	*mask*

GetUpdateMask

15–12	11–8	7–0
0	*axis*	**FA**h

First data word

	15–0
read	*mask*

Description

SetUpdateMask configures what loops in the *axis* are updated when an update is executed on the given *axis*. If the bitmask for a given loop is set in the *mask*, the operating parameters for that loop will be updated from the buffered values when an **Update** or **MultiUpdate** command is received. The bitmask encoding is given below.

Name	Bit(s)	Description
Trajectory	0	Set to 1 to update trajectory from buffered parameters.
Position Loop	1	Set to 1 to update position loop from buffered parameters.
—	2	Reserved
Current Loop	3	Set to 1 to update current loop from buffered parameters.
—	4–15	Reserved

For example, if the update mask for a given *axis* is set to hexadecimal 0003h, the trajectory and position loop parameters will be updated from their buffered values when an **Update** or **MultiUpdate** command is received for that *axis*.

The Current Loop bit applies regardless of the active current control mode. When it is set, an **Update** or **MultiUpdate** command will update either the active FOC parameters, or the active digital current loop parameters, depending on which Current Control mode is active.

GetUpdateMask gets the update mask for the indicated *axis*.

SetUpdateMask (cont.) F9h
GetUpdateMask FAh

Restrictions

The current loop bit is only valid for products that include a current loop.

C-Motion API

```
PMDresult PMDSetUpdateMask (PMDAxisInterface axis_intf,
                            PMDuint16 mask)
PMDresult PMDGetUpdateMask (PMDAxisInterface axis_intf,
                            PMDuint16* mask)
```

VB-Motion API

```
Dim mask as Short
MagellanAxis.UpdateMask = mask
mask = MagellanAxis.UpdateMask
```

see

Set/GetBreakpointUpdateMask (p. 211), **Update** (p. 215), **MultiUpdate** (p. 65)

SetVelocity buffered 11h
GetVelocity 4Bh

Syntax

SetVelocity *axis velocity*
GetVelocity *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding		
axis	*Axis1*	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	**Range**	**Scaling**	**Units**
velocity	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	counts/cycle microsteps/cycle

Packet Structure

SetVelocity

	0	*axis*	**11**h
	15 12	11 8	7 0

First data word

write	*velocity* (high-order part)	
	31	16

Second data word

write	*velocity* (low-order part)	
	15	0

GetVelocity

	0	*axis*	**4B**h
	15 12	11 8	7 0

First data word

read	*velocity* (high-order part)	
	31	16

Second data word

read	*velocity* (low-order part)	
	15	0

Description

SetVelocity loads the maximum velocity buffer register for the specified *axis*.

GetVelocity returns the contents of the maximum velocity buffer register.

Scaling example: To load a velocity value of 1.750 counts/cycle, multiply by 65,536 (giving 114,688) and load the resultant number as a 32-bit number; giving 0001 in the high word and C000h in the low word. Numbers returned by **GetVelocity** must correspondingly be divided by 65,536 to convert to units of counts/cycle.

Restrictions

SetVelocity may not be issued while an axis is in motion with the S-curve profile.

SetVelocity is not valid in Electronic Gear profile mode.

The velocity cannot be negative, except in the Velocity Contouring profile mode.

SetVelocity is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory Update bit set in the update mask.

SetVelocity (cont.) buffered 11h
GetVelocity 4Bh

C-Motion API

```
PMDresult PMDSetVelocity(PMDAxisInterface axis_intf,
                         PMDint32 velocity)
PMDresult PMDGetVelocity(PMDAxisInterface axis_intf,
                         PMDint32* velocity)
```

VB-Motion API

```
Dim velocity as Long
MagellanAxis.Velocity = velocity
velocity = MagellanAxis.Velocity
```

see

Set/GetAcceleration (p. 83), **Set/GetDeceleration** (p. 122), **Set/GetJerk** (p. 148), **Set/GetPosition** (p. 172), **MultiUpdate** (p. 65), **Update** (p. 215)

Update 1Ah

Syntax

Update *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet Structure

Update

15–12	11–8	7–0
0	*axis*	**1A**h

Description

Update causes all buffered data parameters to be copied into the corresponding run-time registers on the specified ***axis***. When the **Update** command is executed, the update mask is used to determine which groups of parameters are actually updated.

The following table shows the buffered commands and variables that are activated by the **Update** command.

Group	Command/Parameter
Trajectory	**Acceleration** **Deceleration** **Gear Ratio** **Jerk** **Position** **Profile Mode** **Stop Mode** **Velocity** **Clear Position Error**
Position Servo	**Derivative Time** **Integrator Sum Limit** **Kaff** **Kd** **Ki** **Kp** **Kvff** **Kout** **Motor Command**
Current Loops	**Integrator Sum Limit** **Ki** **Kp**

Atlas

No additional Atlas communication need be performed for this command, because the update bit in the Atlas torque command is used to cause an Atlas amplifier update. See *Atlas Digital Amplifier Complete Technical Reference* for more detail.

Restrictions

C-Motion API

```
PMDresult PMDUpdate(PMDAxisInterface axis_intf)
```

VB-Motion API

```
MagellanAxis.Update()
```

see

MultiUpdate (p. 65), **Set/GetUpdateMask** (p. 211)

WriteBuffer C8h

Syntax

WriteBuffer *bufferID value*

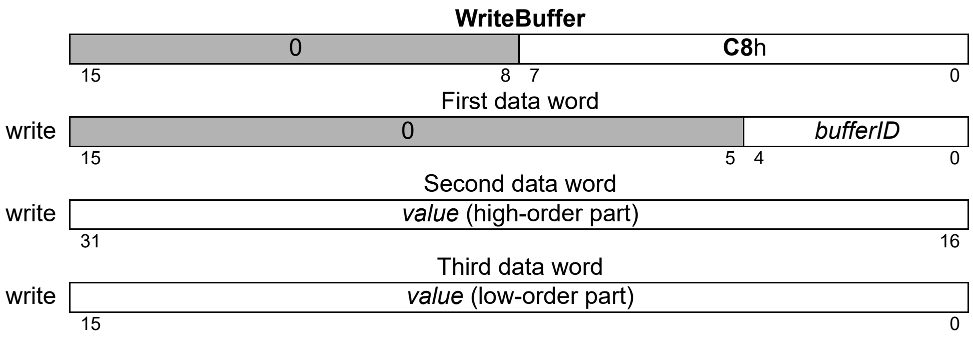
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range
bufferID	unsigned 16 bits	0 *to* 31
value	signed 32 bits	-2^{31} *to* $2^{31}-1$

Packet Structure



Description

WriteBuffer writes the 32-bit *value* into the location pointed to by the write buffer index in the specified buffer. After the contents have been written, the write index is incremented by 1. If the result is equal to the buffer length (set by **SetBufferLength**), the index is reset to zero (0).

Restrictions

The command is not available on all products. See the product user guide.

C-Motion API

```
PMDresult PMDWriteBuffer(PMDAxisInterface axis_intf,
                         PMDuint16 bufferID,
                         PMDint32 data)
```

VB-Motion API

```
Dim data as Long
MagellanObject.WriteBuffer( bufferID ) = data
```

see

ReadBuffer (p. 72), **Set/GetBufferWriteIndex** (p. 106)

WriteIO 82h

Syntax

WriteIO *address data*

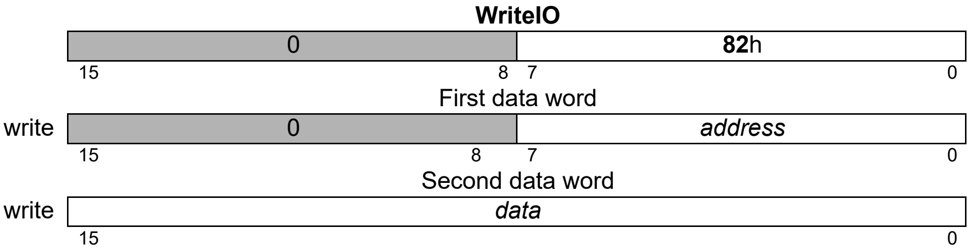
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range
address	unsigned 16 bits	0 *to* 255
data	unsigned 16 bits	0 *to* 2^{16}–1

Packet Structure



Description

WriteIO writes one 16-bit word of *data* to *address*. The *address* is an offset from location 1000h of the motion control IC's peripheral device address space.

The format and interpretation of the 16-bit data word are dependent on the user-defined device being addressed. User-defined I/O can be used to implement a variety of features such as additional parallel I/O, flash memory for non-volatile configuration information storage, or display devices such as LED arrays.

Restrictions

This command is only available in products with general purpose parallel port interfaces. See the product user guide.

C-Motion API

```
PMDresult PMDWriteIO(PMDAxisInterface axis_intf,
                     PMDuint16 address,
                     PMDuint16 data)
```

VB-Motion API

```
Dim data as Short
MagellanObject.IO( address ) = data
```

see

ReadIO (p. 74)

This page intentionally left blank.

3. Instruction Summary Tables

3.1 Descriptions by Functional Category

Breakpoints and Interrupts		Page
Set/GetBreakpointUpdateMask	Set/Get mask for what is updated by breakpoint action “update.”	97
ClearInterrupt	Reset interrupt.	19
Set/GetBreakpoint	Set/Get breakpoint type.	94
Set/GetBreakpointValue	Set/Get breakpoint comparison value.	99
GetInterruptAxis	Get the axes with pending interrupts.	49
Set/GetInterruptMask	Set/Get interrupt mask.	146

Motor Phase and Commutation		
Set/GetCommutationMode	Set/Get the commutation phasing mode.	109
Set/GetCommutationParameter	Set/Get the commutation parameter.	110
Set/GetPhaseAngle	Set/Get current commutation phase angle.	161
GetPhaseCommand	Get the motor output command for a given phase A, B, or C.	50
Set/GetPhaseCorrectionMode	Set/Get phase correction mode.	163
Set/GetPhaseCounts	Set/Get number of encoder counts per commutation cycle.	164
Set/GetPhaseInitializeMode	Set/Get phase initialization method.	166
Set/GetPhaseInitializeTime	Set/Get the time parameters for algorithmic phase initialization.	167
Set/GetPhaseOffset	Set/Get phase offset value.	168
Set/GetPhaseParameter	Set/Get phase parameter.	169
Set/GetPhasePrescale	Set/Get commutation prescaler mode.	171
InitializePhase	Perform phase initialization procedure.	64

Current Loops		
CalibrateAnalog	Set analog offsets to zero output.	17
Set/GetAnalogCalibration	Set analog measurement offsets.	88
Set/GetCurrentControlMode	Set/Get current loop mode (PhaseA/B or FOC).	115
Set/GetCurrentLoop	Set/Get a parameter for the PhaseA/B current loops.	120
GetCurrentLoopValue	Get the instantaneous value of a node in the PhaseA/B current loops.	34
Set/GetFOC	Set/Get a parameter for the FOC current control.	141
GetFOCValue	Get the instantaneous value of a node in the FOC current control.	44

Digital Servo Filter		
ClearPositionError	Set position error to 0.	20
GetPositionError	Get actual position error.	51
Set/GetPositionLoop	Set/Get a parameter for the Digital Servo Loop.	174
GetPositionLoopValue	Get the current value of a node in the Digital Servo Loop.	52
Set/GetPositionErrorLimit	Set/Get the maximum position error limit.	173
Set/GetAuxiliaryEncoderSource	Controls the dual encoder loop feature.	90

Encoder		
AdjustActualPosition	Sums the specified offset with the actual encoder position.	16
Set/GetActualPosition	Set/Get the actual encoder position.	85

Encoder

Set/GetActualPositionUnits	Set/Get the unit type returned for the actual encoder position.	87
GetActualVelocity	Get the actual encoder velocity.	27
Set/GetCaptureSource	Set/Get the capture source.	108
GetCaptureValue	Get current axis position capture value and reset the capture.	29
Set/GetEncoderModulus	Set/Get the full scale range of the parallel-word encoder.	131
Set/GetEncoderSource	Set/Get the encoder type.	132
Set/GetEncoderToStepRatio	Set/Get encoder count to step ratio.	134

Motor Output

GetActiveMotorCommand	Read the active motor command value.	23
GetDriveValue	Read drive bus voltage, bus current, or temperature.	40
Set/GetMotorCommand	Set/Get direct value to motor output register, read buffered motor output command.	151
Set/GetMotorType	Set/Get motor type for axis.	154
Set/GetOutputMode	Set/Get the motor output mode.	158
Set/GetPWMFrequency	Set/Get the PWM output frequency	178
Set/GetDrivePWM	Set/Get PWM parameters	129
Set/GetStepRange	Set/Get the allowable range (in KHz) for step output generation.	190
Set/GetCurrentFoldback	Set/Get the maximum continuous operating current for I^2t Current Foldback	117
Set/GetCurrent	Set/Get parameters for holding current.	113
Set/GetMotorLimit	Set/Get motor output limit.	153
Set/GetMotorBias	Set/Get the motor bias (applied outside of position loop).	150

Operating Mode, Event, and Update Control

Set/GetOperatingMode	Set/Get static Operating Mode of the axis.	156
RestoreOperatingMode	Restore the Current Operating Mode to the static Operating Mode.	82
GetActiveOperatingMode	Get the Active Operating Mode of the axis.	24
MultiUpdate	Forces buffered command values to become active for multiple axes.	65
Update	Forces buffered command values to become active.	215
Set/GetUpdateMask	Set/Get mask for what loops are updated by update command.	211
Set/GetEventAction	Set/Get the response of the axis to an event.	135

Postion Servo Loop Control

Set/GetMotionCompleteMode	Set/Get the motion complete mode.	149
Set/GetSampleTime	Set/Get servo loop sample time.	180
Set/GetSettleTime	Set/Get the axis-settled time.	184
Set/GetSettleWindow	Set/Get the settle-window boundary value.	185
GetTime	Get current chipset time (number of servo loops).	58
Set/GetTrackingWindow	Set/Get the tracking window boundary value.	210

Profile Generation

Set/GetAcceleration	Set/Get acceleration limit.	83
GetCommandedAcceleration	Get commanded (instantaneous desired) acceleration.	31
GetCommandedPosition	Get commanded (instantaneous desired) position.	32
GetCommandedVelocity	Get commanded (instantaneous desired) velocity.	33
Set/GetDeceleration	Set/Get deceleration limit.	122
Set/GetGearMaster	Set/Get the electronic gear mode master axis and source.	143
Set/GetGearRatio	Set/Get commanded electronic gear ratio.	145

Profile Generation		
Set/GetJerk	Set/Get jerk limit.	148
Set/GetPosition	Set/Get destination position.	172
Set/GetProfileMode	Set/Get current profile mode.	177
Set/GetStartVelocity	Set/Get start velocity.	189
Set/GetStopMode	Set/Get stop command: abrupt, smooth, or none.	191
Set/GetVelocity	Set/Get velocity limit.	213

RAM Buffer		
Set/GetBufferLength	Set/Get the length of a memory buffer.	101
Set/GetBufferReadIndex	Set/Get the buffer read pointer for a particular buffer.	103
Set/GetBufferStart	Set/Get the start location of a memory buffer.	104
Set/GetBufferWriteIndex	Set/Get the buffer write pointer for a particular buffer.	106
ReadBuffer	Read a long word value from a buffer memory location.	72
WriteBuffer	Write a long word value to a buffer memory location.	216

Drive		
Set/GetDriveFaultParameter	Set/Get threshold for Overvoltage, Undervoltage, or Overcurrent fault.	28
GetBusVoltage	Get the current bus voltage reading.	28
Set/GetOvertemperatureLimit	Set/Get threshold for Overtemperature fault.	159
GetTemperature	Gets current temperature reading.	57
Set/GetFaultOutMask	Set/Get mask for FaultOut from Event Status register.	137
GetDriveFaultStatus	Gets the Drive Fault Status register.	36
ClearDriveFaultStatus	Clears the Drive Fault Status register.	18

Status Registers and AxisOut Indicator		
GetActivityStatus	Get Activity Status register.	25
GetDriveStatus	Gets the Drive Status register.	38
Set/GetAxisOutMask	Set/Get AxisOut source.	92
GetEventStatus	Get Event Status word.	42
GetSignalStatus	Get the current axis Signal Status register.	55
Set/GetSignalSense	Set/Get the interpretation of the Signal Status bits.	186
ResetEventStatus	Reset bits in Event Status word.	80

Traces		
GetTraceCount	Get the number of traced data points.	59
Set/GetTraceMode	Set/Get the trace mode (rolling or one-time).	193
Set/GetTracePeriod	Set/Get the trace period.	195
Set/GetTraceStart	Set/Get the trace start condition.	196
GetTraceStatus	Get the trace status word.	60
Set/GetTraceStop	Set/Get the trace stop condition.	199
Set/GetTraceVariable	Set/Get a trace variable setting.	202

Communications		
Set/GetCANMode	Set/Get the CAN 2.0B configuration mode.	107
GetInstructionError	Get the most recent I/O error code.	46
Set/GetSerialPortMode	Set/Get the serial-port configuration mode.	182
Set/GetSPIMode	Set/Get the SPI output mode.	188

Miscellaneous		
ExecutionControl	Delays execution during NVRAM initialization.	21
GetChecksum	Reads the internal chip checksum.	30

Miscellaneous		
GetProductInfo	Reads fixed information about the Magellan IC.	53
Set/GetSynchronizationMode	Set/Get the synchronization mode.	192
GetVersion	Get chipset software version information.	62
NoOperation	Perform no operation, used to verify communications.	67
ReadIO	Read user-defined I/O value.	74
Reset	Reset chipset.	75
NVRAM	Program non-volatile memory	68
WriteIO	Write user-defined I/O value.	217
ReadAnalog	Read a raw analog input.	71
Set/GetDefault	Set/Get a reset default setting from non-volatile memory.	123

3.2 Command Support by Product

The following table summarizes the support of each Magellan command by the different product families. The "MC58000/Atlas" column is for commands affecting a Atlas digital amplifier attached to an MC58000 Motion Control IC. In that column "pass through" means that a command is sent directly to Atlas, even if directed to Magellan; "separate" means that a command may be directed either to Atlas or Magellan, and "combined" means that a command directed to Magellan may result in a command being sent to Atlas as well.

			MC58000/		N-Series	
Command	**MC55000**	**MC58000**	**Atlas**	**ION**	**ION**	**MC58113**
Breakpoints and Interrupts						
Set/GetBreakpointUpdateMask	Y	Y		Y	Y	Y
ClearInterrupt	Y	Y		Y	Y	Y
Set/GetBreakpoint	Y	Y		Y	Y	Y
Set/GetBreakpointValue	Y	Y		Y	Y	Y
GetInterruptAxis	Y	Y		Y	Y	Y
Set/GetInterruptMask	Y	Y		Y	Y	Y
Motor Phase and Commutation						
Set/GetCommutationMode	Y	Y		Y	Y	Y
Set/GetCommutationParameter					Y	
Set/GetPhaseAngle		Y		Y	Y	Y
GetPhaseCommand		Y	pass through	Y	Y	Y
Set/GetPhaseCorrectionMode		Y		Y	Y	Y
Set/GetPhaseCounts		Y	stepper only	Y	Y	Y
Set/GetPhaseInitializeMode		Y		Y	Y	Y
Set/GetPhaseInitializeTime		Y		Y	Y	Y
Set/GetPhaseOffset		Y		Y	Y	Y
Set/GetPhaseParameter					Y	
Set/GetPhasePrescale		Y		Y	Y	Y
InitializePhase		Y		Y	Y	Y
Current Loops						
Set/GetAnalogCalibration					Y	Y
CalibrateAnalog					Y	Y
Set/GetCurrentControlMode			pass through	Y	Y	Y
Set/GetCurrentLoop			pass through	Y	Y	Y
GetCurrentLoopValue			pass through	Y	Y	Y

			MC58000/		N-Series	
Command	**MC55000**	**MC58000**	**Atlas**	**ION**	**ION**	**MC58113**
Set/GetFOC			pass through	Y	Y	Y
GetFOCValue			pass through	Y	Y	Y
Digital Servo Filter						
ClearPositionError	Y	Y		Y	Y	Y
GetPositionError	Y	Y		Y	Y	Y
Set/GetPositionLoop		Y		Y	Y	Y
GetPositionLoopValue		Y		Y	Y	Y
Set/GetPositionErrorLimit	Y	Y		Y	Y	Y
Set/GetAuxiliaryEncoderSource		Y		Y	Y	Y
Encoder						
AdjustActualPosition	Y	Y		Y	Y	Y
Set/GetActualPosition	Y	Y		Y	Y	Y
Set/GetActualPositionUnits	Y	Y		Y	Y	Y
GetActualVelocity	Y	Y		Y	Y	Y
Set/GetCaptureSource	Y	Y		Y	Y	Y
GetCaptureValue	Y	Y		Y	Y	Y
Set/GetEncoderModulus	Y	Y			Y	Y
Set/GetEncoderSource	Y	Y		Y	Y	Y
Set/GetEncoderToStepRatio	Y	Y		Y	Y	Y
Motor Output						
GetActiveMotorCommand	Y	Y		Y	Y	Y
Set/GetMotorCommand		Y		Y	Y	Y
Set/GetMotorType	read only	Y		read only	Y	Y
Set/GetOutputMode	read only	Y		read only	Y	Y
Set/GetPWMFrequency			pass through	Y	Y	Y
Set/GetDrivePWM			pass through		Y	Y
Set/GetStepRange	Y	Y			Y	Y
Set/GetCurrentFoldback		Y	pass through	Y	Y	Y
Set/GetCurrent			combined		Y	Y
Set/GetMotorLimit		Y		Y	Y	Y
Set/GetMotorBias		Y		Y	Y	Y
Operating Mode, Event, and Update Control						
Set/GetOperatingMode	Y	Y	combined	Y	Y	Y
RestoreOperatingMode	Y	Y	combined	Y	Y	Y
GetActiveOperatingMode	Y	Y		Y	Y	Y
MultiUpdate	Y	Y			Y	Y
Update	Y	Y		Y	Y	Y
Set/GetUpdateMask	Y	Y		Y	Y	Y
Set/GetEventAction	Y	Y	combined (foldback)	Y	Y	Y
Position Servo Loop Control						
Set/GetMotionCompleteMode	Y	Y		Y	Y	Y
Set/GetSampleTime	Y	Y		Y	Y	Y

Command	MC55000	MC58000	MC58000/ Atlas	ION	N-Series ION	MC58113
Set/GetSettleTime	Y	Y		Y	Y	Y
Set/GetSettleWindow	Y	Y		Y	Y	Y
Set/GetTrackingWindow	Y	Y		Y	Y	Y
GetTime	Y	Y	separate	Y	Y	Y
Profile Generation						
Set/GetAcceleration	Y	Y		Y	Y	Y
GetCommandedAcceleration	Y	Y		Y	Y	Y
GetCommandedPosition	Y	Y		Y	Y	Y
GetCommandedVelocity	Y	Y		Y	Y	Y
Set/GetDeceleration	Y	Y		Y	Y	Y
Set/GetGearMaster	Y	Y		Y	Y	Y
Set/GetGearRatio	Y	Y		Y	Y	Y
Profile Generation						
Set/GetJerk	Y	Y		Y	Y	Y
Set/GetPosition	Y	Y		Y	Y	Y
Set/GetProfileMode	Y	Y		Y	Y	Y
Set/GetStartVelocity	Y	Y		Y	Y	Y
Set/GetStopMode	Y	Y		Y	Y	Y
Set/GetVelocity	Y	Y		Y	Y	Y
RAM Buffer						
Set/GetBufferLength	Y	Y	separate	Y	Y	Y
Set/GetBufferReadIndex	Y	Y	separate	Y	Y	Y
Set/GetBufferStart	Y	Y	separate	Y	Y	Y
Set/GetBufferWriteIndex	Y	Y	separate	Y	Y	Y
ReadBuffer	Y	Y		Y	Y	Y
WriteBuffer	Y	Y		Y	Y	Y
ReadBuffer16			Atlas only		Y	Y
Drive						
Set/GetDriveFaultParameter			pass through		Y	Y
GetBusVoltage			pass through	Y		
GetDriveValue					Y	Y
Set/GetOvertemperatureLimit				Y		
GetTemperature			pass through	Y		
Set/GetFaultOutMask			Atlas only	Y	Y	Y
GetDriveFaultStatus			pass through	Y	Y	Y
ClearDriveFaultStatus			pass through	Y	Y	Y
Status Registers and AxisOut Indicator						
GetActivityStatus	Y	Y		Y	Y	Y
GetDriveStatus	Y	Y		Y	Y	Y
Set/GetAxisOutMask	Y	Y		Y	Y	Y
GetEventStatus	Y	Y		Y	Y	Y
GetSignalStatus	Y	Y	separate	Y	Y	Y
Set/GetSignalSense	Y	Y		Y	Y	Y
ResetEventStatus	Y	Y	combined	Y	Y	Y

Command	MC55000	MC58000	MC58000/ Atlas	ION	N-Series ION	MC58113
Traces						
GetTraceCount	Y	Y	separate	Y	Y	Y
Set/GetTraceMode	Y	Y	separate	Y	Y	Y
Set/GetTracePeriod	Y	Y	separate	Y	Y	Y
Set/GetTraceStart	Y	Y	separate	Y	Y	Y
GetTraceStatus	Y	Y	separate	Y	Y	Y
Set/GetTraceStop	Y	Y	separate	Y	Y	Y
Set/GetTraceVariable	Y	Y	separate	Y	Y	Y
Communications						
Set/GetCANMode	Y	Y		Y	Y	Y
GetInstructionError	Y	Y	separate	Y	Y	Y
Set/GetSerialPortMode	Y	Y		Y	Y	Y
Set/GetSPIMode		Y			Y	Y
Miscellaneous						
ExecutionControl					Y	
GetChecksum	Y	Y	separate	Y	Y	Y
GetProductInfo					Y	
Set/GetSynchronizationMode		Y			Y	Y
GetVersion	Y	Y	separate	Y	Y	Y
NoOperation	Y	Y	separate	Y	Y	Y
ReadIO	Y	Y				
Reset	Y	Y	separate	Y	Y	Y
WriteIO	Y	Y		Y		
ReadAnalog	Y	Y	separate	Y	Y	Y
Set/GetDefault				Y		
NVRAM			Y		Y	Y

3.3 Alphabetical Listing

Get/Set instructions pairs are shown together on the same line of the table.

Instruction	Code	Instruction	Code	Page
AdjustActualPosition	F5h			16
CalibrateAnalog	6Fh			17
ClearDriveFaultStatus	6Ch			18
ClearInterrupt	ACh			19
ClearPositionError	47h			20
NVRAM	30h			68
ExecutionControl	35h			21
GetAcceleration	4Ch	**SetAcceleration**	90h	83
GetActiveMotorCommand	3Ah			23
GetActiveOperatingMode	57h			24
GetActivityStatus	A6h			25
GetActualPosition	37h	**SetActualPosition**	4Dh	85

Instruction	Code	Instruction	Code	Page
GetActualPositionUnits	BFh	**SetActualPositionUnits**	BEh	87
GetActualVelocity	ADh			27
GetAnalogCalibration	2Ah	**SetAnalogCalibration**	29h	88
GetAuxiliaryEncoderSource	09h	**SetAuxiliaryEncoderSource**	08h	90
GetAxisOutMask	46h	**SetAxisOutMask**	45h	92
GetBreakpoint	D5h	**SetBreakpoint**	D4h	94
GetBreakpointUpdateMask	33h	**SetBreakpointUpdateMask**	32h	97
GetBreakpointValue	D7h	**SetBreakpointValue**	D6h	99
GetBufferLength	C3h	**SetBufferLength**	C2h	101
GetBufferReadIndex	C7h	**SetBufferReadIndex**	C6h	103
GetBufferStart	C1h	**SetBufferStart**	C0h	104
GetBufferWriteIndex	C5h	**SetBufferWriteIndex**	C4h	106
GetBusVoltage	40h			28
GetCANMode	15h	**SetCANMode**	12h	107
GetCaptureSource	D9h	**SetCaptureSource**	D8h	108
GetCaptureValue	36h			29
GetChecksum	F8h			30
GetCommandedAcceleration	A7h			31
GetCommandedPosition	1Dh			32
GetCommandedVelocity	1Eh			33
GetCommutationMode	E3h	**SetCommutationMode**	E2h	109
GetCommutationParameter	64h	**SetCommutationParameter**	63h	110
GetCurrent	5Fh	**SetCurrent**	5Eh	113
GetCurrentControlMode	44h	**SetCurrentControlMode**	43h	115
GetCurrentFoldback	42h	**SetCurrentFoldback**	41h	117
GetCurrentLoop	74h	**SetCurrentLoop**	73h	120
GetCurrentLoopValue	71h			34
GetDeceleration	92h	**SetDeceleration**	91h	122
GetDefault	8Ah	**SetDefault**	89h	123
GetDriveCommandMode	7Fh	**SetDriveCommandMode**	7Eh	125
GetDriveFaultParameter	60h	**SetDriveFaultParameter**	62h	126
GetDriveFaultStatus	6Dh			36
GetDrivePWM	24h	**SetDrivePWM**	23h	129
GetDriveStatus	0Eh			38
GetDriveValue	70h			40
GetEncoderModulus	8Eh	**SetEncoderModulus**	8Dh	131
GetEncoderSource	DBh	**SetEncoderSource**	DAh	132
GetEncoderToStepRatio	DFh	**SetEncoderToStepRatio**	DEh	134
GetEventAction	49h	**SetEventAction**	48h	135
GetEventStatus	31h			42
GetFaultOutMask	FCh	**SetFaultOutMask**	FBh	137
GetFeedbackParameter	22h	**SetFeedbackParameter**	21h	139
GetFOC	F7h	**SetFOC**	F6h	141
GetFOCValue	5Ah			44
GetGearMaster	AFh	**SetGearMaster**	AEh	143
GetGearRatio	59h	**SetGearRatio**	14h	145
GetInstructionError	A5h			46
GetInterruptAxis	E1h			49
GetInterruptMask	56h	**SetInterruptMask**	2Fh	146
GetJerk	58h	**SetJerk**	13h	148
GetMotionCompleteMode	ECh	**SetMotionCompleteMode**	EBh	149

Instruction	Code	Instruction	Code	Page
GetMotorBias	2Dh	**SetMotorBias**	0Fh	150
GetMotorCommand	69h	**SetMotorCommand**	77h	151
GetMotorLimit	07h	**SetMotorLimit**	06h	153
GetMotorType	03h	**SetMotorType**	02h	154
GetOperatingMode	66h	**SetOperatingMode**	65h	156
GetOutputMode	6Eh	**SetOutputMode**	E0h	158
GetOvertemperatureLimit	1Ch	**SetOvertemperatureLimit**	1Bh	159
GetPhaseAngle	2Ch	**SetPhaseAngle**	84h	161
GetPhaseCommand	EAh			50
GetPhaseCorrectionMode	E9h	**SetPhaseCorrectionMode**	E8h	163
GetPhaseCounts	7Dh	**SetPhaseCounts**	75h	164
GetPhaseInitializeMode	E5h	**SetPhaseInitializeMode**	E4h	166
GetPhaseInitializeTime	7Ch	**SetPhaseInitializeTime**	72h	167
GetPhaseOffset	7Bh	**SetPhaseOffset**	76h	168
GetPhaseParameter	86h	**SetPhaseParameter**	85h	169
GetPhasePrescale	E7h	**SetPhasePrescale**	E6h	171
GetPosition	4Ah	**SetPosition**	10h	172
GetPositionError	99h			51
GetPositionErrorLimit	98h	**SetPositionErrorLimit**	97h	173
GetPositionLoop	68h	**SetPositionLoop**	67h	174
GetPositionLoopValue	55h			52
GetProductInfo	01h			53
GetProfileMode	A1h	**SetProfileMode**	A0h	177
GetPWMFrequency	0Dh	**SetPWMFrequency**	0Ch	178
GetSampleTime	3Ch	**SetSampleTime**	3Bh	180
GetSerialPortMode	8Ch	**SetSerialPortMode**	8Bh	182
GetSettleTime	ABh	**SetSettleTime**	AAh	184
GetSettleWindow	BDh	**SetSettleWindow**	BCh	185
GetSignalSense	A3h	**SetSignalSense**	A2h	186
GetSignalStatus	A4h			55
GetSPIMode	0Bh	**SetSPIMode**	0Ah	188
GetStartVelocity	6Bh	**SetStartVelocity**	6Ah	189
GetStepRange	CEh	**SetStepRange**	CFh	190
GetStopMode	D1h	**SetStopMode**	D0h	191
GetSynchronizationMode	F3h	**SetSynchronizationMode**	F2h	192
GetTemperature	53h			57
GetTime	3Eh			58
GetTraceCount	BBh			59
GetTraceMode	B1h	**SetTraceMode**	B0h	193
GetTracePeriod	B9h	**SetTracePeriod**	B8h	195
GetTraceStart	B3h	**SetTraceStart**	B2h	196
GetTraceStatus	BAh			60
GetTraceStop	B5h	**SetTraceStop**	B4h	199
GetTraceValue	28h			61
GetTraceVariable	B7h	**SetTraceVariable**	B6h	202
GetTrackingWindow	A9h	**SetTrackingWindow**	A8h	210
GetUpdateMask	FAh	**SetUpdateMask**	F9h	211
GetVelocity	4Bh	**SetVelocity**	11h	213
GetVersion	8Fh			62
InitializePhase	7Ah			64
MultiUpdate	5Bh			65

Instruction	Code	Instruction	Code	Page
NoOperation	00h			67
ReadAnalog	EFh			71
ReadBuffer	C9h			72
ReadIO	83h			74
Reset	39h			75
ResetEventStatus	34h			80
RestoreOperatingMode	2Eh			82
Update	1Ah			215
WriteBuffer	C8h			216
WriteIO	82h			217

3.4 Numerical Listing

Code	Instruction	Page	Code	Instruction	Page
00h	**NoOperation**	67	3Eh	**GetTime**	58
01h	**GetProductInfo**	53	40h	**GetBusVoltage**	28
02h	**SetMotorType**	154	41h	**SetCurrentFoldback**	117
03h	**GetMotorType**	154	42h	**GetCurrentFoldback**	117
06h	**SetMotorLimit**	153	43h	**SetCurrentControlMode**	115
07h	**GetMotorLimit**	153	44h	**GetCurrentControlMode**	115
08h	**SetAuxiliaryEncoderSource**	90	45h	**SetAxisOutMask**	92
09h	**GetAuxiliaryEncoderSource**	90	46h	**GetAxisOutMask**	92
0Ah	**SetSPIMode**	188	47h	**ClearPositionError**	20
0Bh	**GetSPIMode**	188	48h	**SetEventAction**	135
0Ch	**SetPWMFrequency**	178	49h	**GetEventAction**	135
0Dh	**GetPWMFrequency**	178	4Ah	**GetPosition**	172
0Eh	**GetDriveStatus**	38	4Bh	**GetVelocity**	213
0Fh	**SetMotorBias**	150	4Ch	**GetAcceleration**	83
10h	**SetPosition**	172	4Dh	**SetActualPosition**	85
11h	**SetVelocity**	213	53h	**GetTemperature**	57
12h	**SetCANMode**	107	55h	**GetPositionLoopValue**	52
13h	**SetJerk**	148	56h	**GetInterruptMask**	146
14h	**SetGearRatio**	145	57h	**GetActiveOperatingMode**	24
15h	**GetCANMode**	107	58h	**GetJerk**	148
1Ah	**Update**	215	59h	**GetGearRatio**	145
1Bh	**SetOvertemperatureLimit**	159	5Ah	**GetFOCValue**	44
1Ch	**GetOvertemperatureLimit**	159	5Bh	**MultiUpdate**	65
1Dh	**GetCommandedPosition**	32	5Eh	**SetCurrent**	113
1Eh	**GetCommandedVelocity**	33	5Fh	**GetCurrent**	113
21h	**SetFeedbackParameter**	139	60h	**GetDriveFaultParameter**	126
22h	**GetFeedbackParameter**	139	62h	**SetDriveFaultParameter**	126
23h	**SetDrivePWM**	129	63h	**SetCommutationParameter**	110
24h	**GetDrivePWM**	129	64h	**GetCommutationParameter**	110
28h	**GetTraceValue**	61	65h	**SetOperatingMode**	156
29h	**SetAnalogCalibration**	88	66h	**GetOperatingMode**	156
2Ah	**GetAnalogCalibration**	88	67h	**SetPositionLoop**	174
2Ch	**GetPhaseAngle**	161	68h	**GetPositionLoop**	174
2Dh	**GetMotorBias**	150	69h	**GetMotorCommand**	151
2Eh	**RestoreOperatingMode**	82	6Ah	**SetStartVelocity**	189
2Fh	**SetInterruptMask**	146	6Bh	**GetStartVelocity**	189
30h	**NVRAM**	68	6Ch	**ClearDriveFaultStatus**	18
31h	**GetEventStatus**	42	6Dh	**GetDriveFaultStatus**	36
32h	**SetBreakpointUpdateMask**	97	6Eh	**GetOutputMode**	158
33h	**GetBreakpointUpdateMask**	97	6Fh	**CalibrateAnalog**	17
34h	**ResetEventStatus**	80	70h	**GetDriveValue**	40
35h	**ExecutionControl**	21	71h	**GetCurrentLoopValue**	34
36h	**GetCaptureValue**	29	72h	**SetPhaseInitializeTime**	167
37h	**GetActualPosition**	85	73h	**SetCurrentLoop**	120
39h	**Reset**	75	74h	**GetCurrentLoop**	120
3Ah	**GetActiveMotorCommand**	23	75h	**SetPhaseCounts**	164
3Bh	**SetSampleTime**	180	76h	**SetPhaseOffset**	168
3Ch	**GetSampleTime**	180	77h	**SetMotorCommand**	151

Code	Instruction	Page	Code	Instruction	Page
7Ah	**InitializePhase**	64	BBh	**GetTraceCount**	59
7Bh	**GetPhaseOffset**	168	BCh	**SetSettleWindow**	185
7Ch	**GetPhaseInitializeTime**	167	BDh	**GetSettleWindow**	185
7Dh	**GetPhaseCounts**	164	BEh	**SetActualPositionUnits**	87
7Eh	**SetDriveCommandMode**	125	BFh	**GetActualPositionUnits**	87
7Fh	**GetDriveCommandMode**	125	C0h	**SetBufferStart**	104
82h	**WriteIO**	217	C1h	**GetBufferStart**	104
83h	**ReadIO**	74	C2h	**SetBufferLength**	101
84h	**SetPhaseAngle**	161	C3h	**GetBufferLength**	101
85h	**SetPhaseParameter**	169	C4h	**SetBufferWriteIndex**	106
86h	**GetPhaseParameter**	169	C5h	**GetBufferWriteIndex**	106
89h	**SetDefault**	123	C6h	**SetBufferReadIndex**	103
8Ah	**GetDefault**	123	C7h	**GetBufferReadIndex**	103
8Bh	**SetSerialPortMode**	182	C8h	**WriteBuffer**	216
8Ch	**GetSerialPortMode**	182	C9h	**ReadBuffer**	72
8Dh	**SetEncoderModulus**	131	CEh	**GetStepRange**	190
8Eh	**GetEncoderModulus**	131	CFh	**SetStepRange**	190
8Fh	**GetVersion**	62	D0h	**SetStopMode**	191
90h	**SetAcceleration**	83	D1h	**GetStopMode**	191
91h	**SetDeceleration**	122	D4h	**SetBreakpoint**	94
92h	**GetDeceleration**	122	D5h	**GetBreakpoint**	94
97h	**SetPositionErrorLimit**	173	D6h	**SetBreakpointValue**	99
98h	**GetPositionErrorLimit**	173	D7h	**GetBreakpointValue**	99
99h	**GetPositionError**	51	D8h	**SetCaptureSource**	108
A0h	**SetProfileMode**	177	D9h	**GetCaptureSource**	108
A1h	**GetProfileMode**	177	DAh	**SetEncoderSource**	132
A2h	**SetSignalSense**	186	DBh	**GetEncoderSource**	132
A3h	**GetSignalSense**	186	DEh	**SetEncoderToStepRatio**	134
A4h	**GetSignalStatus**	55	DFh	**GetEncoderToStepRatio**	134
A5h	**GetInstructionError**	46	E0h	**SetOutputMode**	158
A6h	**GetActivityStatus**	25	E1h	**GetInterruptAxis**	49
A7h	**GetCommandedAcceleration**	31	E2h	**SetCommutationMode**	109
A8h	**SetTrackingWindow**	210	E3h	**GetCommutationMode**	109
A9h	**GetTrackingWindow**	210	E4h	**SetPhaseInitializeMode**	166
AAh	**SetSettleTime**	184	E5h	**GetPhaseInitializeMode**	166
ABh	**GetSettleTime**	184	E6h	**SetPhasePrescale**	171
ACh	**ClearInterrupt**	19	E7h	**GetPhasePrescale**	171
ADh	**GetActualVelocity**	27	E8h	**SetPhaseCorrectionMode**	163
AEh	**SetGearMaster**	143	E9h	**GetPhaseCorrectionMode**	163
AFh	**GetGearMaster**	143	EAh	**GetPhaseCommand**	50
B0h	**SetTraceMode**	193	EBh	**SetMotionCompleteMode**	149
B1h	**GetTraceMode**	193	ECh	**GetMotionCompleteMode**	149
B2h	**SetTraceStart**	196	EFh	**ReadAnalog**	71
B3h	**GetTraceStart**	196	F2h	**SetSynchronizationMode**	192
B4h	**SetTraceStop**	199	F3h	**GetSynchronizationMode**	192
B5h	**GetTraceStop**	199	F5h	**AdjustActualPosition**	16
B6h	**SetTraceVariable**	202	F6h	**SetFOC**	141
B7h	**GetTraceVariable**	202	F7h	**GetFOC**	141
B8h	**SetTracePeriod**	195	F8h	**GetChecksum**	30
B9h	**GetTracePeriod**	195	F9h	**SetUpdateMask**	211
BAh	**GetTraceStatus**	60	FAh	**GetUpdateMask**	211

Code	Instruction	Page	Code	Instruction	Page
FBh	**SetFaultOutMask**	137			
FCh	**GetFaultOutMask**	137			

3.5 Magellan Compatibility

Below are commands from Magellan v1.x that have been replaced/superseded by new commands in Magellan v2.x.

Old Command	Old Code	New Command
Set/GetBiquadCoefficient	04h/05h	**Set/GetPositionLoop**
GetDerivative	9Bh	**GetPositionLoopValue**
Set/GetDerivativeTime	9Ch/9Dh	**Set/GetPositionLoop**
GetHostIOError	A5h	**GetInstructionError (name change only)**
GetIntegral	9Ah	**GetPositionLoopValue**
Set/GetIntegrationLimit	95h/96h	**Set/GetPositionLoop**
Set/GetKaff	93h/94h	**Set/GetPositionLoop**
Set/GetKd	27h/52h	**Set/GetPositionLoop**
Set/GetKi	26h/51h	**Set/GetPositionLoop**
Set/GetKout	9Eh/9Fh	**Set/GetPositionLoop**
Set/GetKp	25h/50h	**Set/GetPositionLoop**
Set/GetKvff	2Bh/54h	**Set/GetPositionLoop**
Set/GetAutoStopMode	D2h/D3h	**Set/GetEventAction**
Set/GetMotorMode	DCh/DDh	**Set/GetOperatingMode, RestoreOperatingMode**
Set/GetAxisOutSource	EDh/EEh	**Set/GetAxisOutMask**
Set/GetAxisMode	87h/88h	**Set/GetOperatingMode**
Set/GetLimitSwitchMode	80h/81h	**Set/GetEventAction**

This page intentionally left blank.

For additional information, or for technical assistance,
please contact PMD at (978) 266-1210.

You may also e-mail your request to support@pmdcorp.com

Visit our website at http://www.pmdcorp.com

PMD

Performance Motion Devices
80 Central Street
Boxborough, MA 01719