UNIDEX 31

INTEGRATED MACHINE CONTROLLER

SOFTWARE MANUAL

P/N: EDU 120

AEROTECH, Inc.
101 Zeta Drive
Pittsburgh, PA. 15238-2897 U.S.A.
(412) 963-7470
Fax (412) 963-7459

UNIDEX 31 Integrated Machine Controller Software Manual Revision History:

Preliminary release	June 29, 1990
Revision 2.0	July 14, 1990
Revision 3.0	August 10, 1990
Revision 4.0	January 2, 1991
Revision 5.0	April 17, 1991
Revision 6.0	March 12, 1992
Revision 6.1	February 21, 1995

TABLE OF CONTENTS

∇ ∇ ∇

PREFACE

This section gives you an overview of topics covered in each of the sections of this manual as well as conventions used in this manual. The U31 Machine Controller Software Manual contains information on the following topics:

CHAPTER 1: INTRODUCTION

This chapter contains information on who should read this manual as well as minimum requirements for installation of the software.

CHAPTER 2: INSTALLATION AND SETUP

This chapter contains information regarding first time installation, device driver initialization, header files, include defines, SET and SETM calls, interrupt handling, and axis configuration.

CHAPTER 3: UTILITY FUNCTIONS

This chapter contains all the utility commands, giving a brief description of each and an example of the syntax.

CHAPTER 4: MOTION FUNCTIONS

This chapter contains all the motion commands providing a brief description of each with an example of the syntax.

CHAPTER 5: SYNCHRONIZED MOTION FUNCTIONS

This chapter contains all the synchronized motion commands providing a brief description of each with an example of the syntax.

CHAPTER 6: PROFILING MOTION FUNCTIONS

This chapter contains all the profiling motion commands, giving a brief description of each and an example of the syntax.

CHAPTER 7: PARAMETER FUNCTIONS

This chapter contains all the parameter commands, giving a brief description of each and an example of the syntax.

CHAPTER 8: AUXILIARY I/O FUNCTIONS

This chapter contains all the auxiliary I/O functions providing a brief description of each with an example of the syntax.

CHAPTER 9: VMIC DISCRETE I/O BOARD FUNCTIONS

This chapter contains all the VMIC descrete I/O board functions providing a brief description of each with an example of the syntax.

CHAPTER 10: XYCOM I/O FUNCTIONS

This chapter contains all the XYCOM I/O functions providing a brief description of each with an example of the syntax.

CHAPTER 11: ANALOG I/O FUNCTIONS

This chapter contains all the analog I/O commands providing a brief description of each with an example of the syntax.

CHAPTER 12: ALLEN-BRADLEY VME/PLC5 FUNCTIONS

This chapter contains all the Allen-Bradley VME/PLC5 functions providing a brief description of each with an example of the syntax.

CHAPTER 13: MODICON I/O FUNCTIONS

This chapter contains all the modicon I/O functions providing a brief description of each with an example of the syntax.

CHAPTER 14: SERIAL COMMUNICATION FUNCTIONS

This chapter contains all the serial communication commands providing a brief description of each with an example of the syntax.

CHAPTER 15: PROBE FUNCTIONS

This chapter contains all the probe commands providing a brief description of each with an example of the syntax.

CHAPTER 16: ERROR MAPPING FUNCTIONS

This chapter contains all the error mapping commands providing a brief description of each with an example of the syntax.

CHAPTER 17: TORQUE FUNCTIONS

This chapter contains all the torque commands providing a brief description of each with an example of the syntax.

APPENDIX A: WARRANTY and FIELD SERVICE

Appendix A contains the warranty and field service policy for Aerotech products.

Throughout this manual the following conventions are used:

- The terms UNIDEX 31 and U31 are used interchangeably throughout this manual
- *Italic font* is used to illustrate syntax and arguments for the programming commands
- Note symbols (see right) appear in the outer margin next to important notes
- This manual uses the symbol "∇ ∇ ∇" to indicate the end of a chapter.

The majority of programming commands presented in this manual appear on an individual page with an example of their syntax following it.

∇ ∇ ∇

CHAPTER 1: INTRODUCTION

In This Section:

1.1. Who Should Read This

This manual is intended for anyone who is going to write an application program for the UNIDEX 31 Integrated Machine Controller. It assumes the user has a working knowledge of the "C" programming language or the BASIC programming language and the OS/2 operating system. This manual will not attempt to explain either of these subjects.

This manual contains a description of all the UNIDEX 31 commands outlining how to call them and what parameters are required. A copy of the header files is also attached which contains prototypes for each command.

All commands in the software library begin with the capital letters "DMR" such as *DMROpen ()* or *DMRConfigureAxis ()* allowing convenient searching for any motion command. They are frequently referred to as DMR commands in this text. Care must be taken to ensure that the proper letters in the commands are capitalized to avoid errors at link time for unresolved externals.

The commands are modeled after the OS/2 commands and are cast as unsigned far Pascal so that they may be called from Presentation Manager applications.

The commands are grouped by category such as utility, parameter, synched functions, profile functions, etc.

1.2. Minimum Requirements

The minimum programming environment is a 486 machine with 4 megabytes of RAM running the OS/2 operating system.

Programs may be written and tested for functionality on any machine meeting the above requirement. The software library has a debug mode of operation that does not pass commands on to the axis card. This mode is turned off and on with *DMRCallDebug* command.

The **MOTION$** device driver must be loaded at boot time for axis communication and interrupt handling to occur. This is done by adding the following line to the end of the *CONFIG.SYS* file:

DEVICE=MOTION.SYS

where *MOTION.SYS* exists in the default directory.

∇ ∇ ∇

CHAPTER 2: INSTALLATION AND SETUP

In This Section:

2.1. First Time Installation

The Hardware Manual should be the first document read before installing or powering up the UNIDEX 31 Integrated Machine Controller. After successfully completing all the instructions found there, the machine may be turned on.

The OS/2 operating system should boot up and present the Desktop Manager and Group - Main boxes on the screen. A DOS box should appear in the lower left of the screen and the Print Manager box will appear next to it. This DOS box may not appear depending on how the system is configured when OS/2 is installed.

If a mouse is connected, the arrow may be moved around the screen and used to select the operation desired. Please refer to the OS/2 documentation for further details on using the operating system.

The UNIDEX 31 is programmed under the 'C' programming language. The Microsoft 'C' Version 6.0 compiler is the only compiler. It may be found under the subdirectory 'C:\C600'. There is a file called NEW-VARS.CMD which is a batch file that sets the include, library, and search paths for the compiler. This must be called before any compiling or linking may occur. Remember to add any other needed paths to this file.

The header files previously described and the *DMR.LIB* library file are found in the subdirectory *'C:\U31\INCLUDE'*. Be sure to set the include path and the library path for compiling and linking to look here for the DMR header files and library file.

2.2. Device Driver Initialization

Before axis communication can take place, the application program must open the **MOTION.SYS** device driver with the command *"DMROpen()"*. Any DMR command will return an error or possibly cause a communication interrupt if this command is not called first. After *"DMROpen()"* is called the library keeps track of the handle to the **MOTION.SYS** device driver for future calls to the axis card. The driver should be closed upon exiting the application program by calling *"DMRClose()"*.

All DMR commands will return zero on success and a non-zero error code on failure.

A debug mode of axis communication is available by calling *"DMRCallDebug(int flag)"* where the variable flag sets the debug mode (off, semi, or full debug). This command should be called before the *"DMROpen()"* command if full debug is desired. This causes any DMR command to return a zero code for success. Any future DMR command is not passed on to the device driver allowing the user to write and test DMR motion code on a system that does not have axis cards or the **MOTION.SYS** device driver.

Debugging can be turned off again by passing the off code in the *"DMRCallDebug(int flag)"* call.

2.3. Header Files and Library File

The header files contain prototypes of each DMR command. There are two ways that they may be used depending on individual style.

The first way is to include the file *DMRHEAD.H* with defines immediately above the include statement that cause the desired portions of the prototypes to be included. For example:

```
#define DMRALL_INCL

#include "dmrhead.h"
```

The above will cause all of the DMR prototypes to be included. Separate portions may be excluded by using only the defines that are required. The list of include defines is as follows:

INCL_DMRALL - includes all DMR header files

INCL_DMRUTIL - includes utility header file

INCL_DMRMTN - includes motion header file

INCL_DMRPROF - includes profile header file
INCL_DMRSYNC - includes sync header file

INCL_DMRPARM - includes parameter header file

INCL_DMRXFORM - includes transform header file

INCL_DMRAUX - includes auxiliary I/O header file

INCL_DMRVMIC - includes VMIC Discrete I/O header file

INCL_XYCOM - includes XYCOM I/O header file

INCL_MATRX - includes Matrix Analog header file

INCL_DMRABPLC - includes Allen Bradley VME/PLC5 header file

INCL_DMRSERIAL - includes serial communication header file

INCL_DMRPROBE - includes Probe commands header file

INCL_DMRERRORMAP - includes Error Mapping header file

The second way to use the header files is to first include the *DMRSTRCT.H* header file and then explicitly include each header file that is needed. The only exception is that the *DMRUTIL.H* file must be included before the *DMRPARM.H* header file.

The list of DMR header files is as follows:

DMRHEAD.H - contains ifdef's to include all the other header files. It also contains any structures needed by those header files.

DMRUTIL.H - contains prototypes for the utility functions such as configuration, open and close, etc.

DMRMTN.H - contains prototypes for the movement calls.

DMRPROF.H - contains prototypes for the profiling calls.

DMRSYNC.H - contains prototypes for the cam table calls.

DMRPARM.H - contains prototypes for the set and get parameter functions.

DMRAUX.H - contains the prototypes for auxiliary output table functions. The auxiliary output on/off state can be directly coupled to the position of a master for tightly coupled motion and IO control.

DMRVMIC.H - contains the prototypes for the VMIC Discrete I/O board functions.

DMRXYCOM.H - contains the prototypes for the XYCOM Discrete I/O board functions.

DMRMATRX.H - contains the prototypes for the MATRIX Analog board functions.

DMRABPLC.H - contains the prototypes for the Allen- Bradley VME/PLC5 I/O functions.

DMRSERL.H - contains the prototypes for the serial communication commands. Included are commands to allow transmitting and receiving characters via one of the four RS-232C ports residing on axis processor board, as well as Teach Pendant commands which utilize the RS-232 ports.

DMRPROB.H - contains the prototype for the Probe commands. These commands allow user to setup I/O points for triggering, enabling user to latch current axis position at the trigger.

DMRERRMP.H - contains the prototypes for error correction table functions. These commands are used to correct for position errors due to mechanical mis-alignments, gear backlash, etc..

DMRDEF.H - contains the #defines for the parameter header file. This is automatically included by the DMRPARM.H file and should not have to done by the user.

DMRSTRCT.H - contains the typedef's for all the structures used by the DMR commands. It should not be included if the first method of including is used.

TORQUE.H - contains the function prototypes for the torque control mode. These two functions allow the user to get and set the torque mode.

DMRVIRT.H - contains the typdef's for the virtual I/O data structure and the required virtual I/O function prototypes.

DMRVJOG.H - contains the defines for the axis VME jog address's and the required function prototypes.

DMRPHASE.H - contains the function prototypes for the Get/Set phase advance functions.

DMRLASER.H - contains the definitions, structures and function prototypes for the Aerotech PSO Laser Firing board.

2.3.1. Library File

The library of software commands are contained in a file called *DMR.LIB*. This file must be linked in with any application that makes DMR calls.

2.4. SET and SETM Calls

There are two types of set commands listed in the header files.

The first type looks like *DMRSetIntegralGain*(ax,parm). This allows the user to specify any number of axes (up to 16) and set them all with the same integral gain parameter number.

The second appears as *DMRSetMIntegralGain*(axis,*parm) where the M stands for multiple parameters. The user may specify several axes and set them with different integral gain numbers. The parm parameter in this case is a pointer to an array of numbers. The numerically lowest axis receives the first parameter, the next numerically largest axis receives the next parameter, and so on.

2.5. Interrupt Handling

The **MOTION.SYS** device driver handles interrupt processing between the axis card and the application program. The application creates system semaphores and passes the semaphores' handle to the driver along with an axis mask of the axes that are to be associated with that semaphore. The application should have a thread that blocks on that semaphore and then does fault processing when it is cleared. It should then set the semaphore again to enable interrupt processing again.

In this release there are (16) semaphores for interrupt handling. Any axis associated with a semaphore causes the semaphore to clear. The commands used for this are *DMRSendIntSemHandle*(), *DMRClearIntSem*() and *DMRGetSemMask*().

In order for the axis to cause interrupts to the application program, the axis fault mask has to be set for the faults that the user wants notified on. The axis interrupt mask must also be set correspondingly so that the interrupt will occur. The commands needed to accomplish this are the axis parameter calls *DMRSetFaultMask*() and *DMRSetInterruptMask*().

There are several other masks that can be used to cause the axis to either halt, disable, or cause the auxiliary output to turn off. These commands are *DMRSetDisableMask*(), *DMRSetHaltMask*(), and *DMRSetAuxMask*().

There is also a command to retrieve the last fault vector from the **MOTION.SYS** device driver. This command accepts a semaphore handle that has been previously sent to the device driver with an axis mask. The vector returned will represent which axes last faulted.

If an interrupt semaphore is not sent to the MOTION.SYS device driver before an interrupt occurs, the system will lock up on an axis interrupt.

There is also a communication semaphore that must be set with the *DMRSendCommSem* command after the *DMROpen* call. This semaphore is cleared whenever there is a communication error. If no communication semaphore has been passed to the driver and a communication error occurs, then the system will lock up.

2.6. Axis Configuration

Before any axis may be enabled, or even return a meaningful position value, it must be configured via the *DMRConfigAxis* or *DMRConfigEncoder* command. This allows the application to associate a logical axis with physical feedback and D to A channels on the IO boards. If an enable or movement command is issued before the configure command, nothing will take place. The axis card does not default to a given mapping.

One of the parameters in the *DMRConfigAxis* command is a commutation offset. If a brushless motor is attached to the axis card, the commutation offset provides a means of allowing the axis card to know where the null of the resolver is in relation to the poles of the motor. This enables the axis card to commutate the three phase signal to the motor drive. Refer to Motor Manufacturer's Manual to obtain this information.

∇ ∇ ∇

CHAPTER 3: UTILITY FUNCTIONS

3.1. DMROpen

The *DMROpen* function opens the **MOTION.SYS** device driver for axis card communication, interrupt handling, and fault handling. No axis communication may take place nor will interrupt acknowledge cycles occur until after this command is called.

DMROpen invokes a "DosOpen()" call to the device driver in the file **MOTION.SYS**. The library keeps track of the handle passed back for subsequent calls to the device driver.

The file **MOTION.SYS** must be included in the **CONFIG.SYS** file with the line:

DEVICE=MOTION.SYS

This file will be loaded during boot-up by the OS/2 operating system.

A debug mode of operation may be enabled using the *"DMRCallDebug"* command. If this mode is desired, then the *"DMRCallDebug"* command should be performed before any other DMR calls.

The *DMROpen* function returns the same values as the DosOpen call.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMROpen(void)

For related commands see:

DMRClose

DMRCallDebug

3.2. DMRClose

The *DMRClose* function closes the **MOTION.SYS** device driver. No axis communication may take place after this command is called unless "DMRCallDebug" has been invoked with a value of two (i.e. communication between the application and the device driver has been disabled - zero will be returned on any call).

The *DMRClose* function returns the same values as the DosClose call.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRClose(void)

For related commands see:

DMROpen

DMRCallDebug

3.3. DMRCallDebug

The *DMRCallDebug* function allows the user to turn axis communication on and off through this call.

A flag value of zero turns on axis communication. The device driver is initialized in this state. Full error reporting will be performed. The communication semaphore will be cleared whenever a communication error occurs and the interrupt semaphore will be cleared on an axis interrupt fault.

A flag value of one disables error returns on commands that are not implemented yet, but still does error reporting on commands that do exist.

A flag value of two turns off error reporting on all commands and does not pass the commands on to the axis card. This allows a user to test a motion program on any 80286 or 80386 system. It should be called before any other DMR commands if this option is going to be used.

The *DMRCallDebug* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRCallDebug(flag)

int flag; mode command for this call

For related commands see:

DMRSendCommSem

DMRSendIntSem DMROpen

3.4. DMRDownload

This allows axis card firmware to be downloaded to a particular axis card.

Firmware may not be downloaded to an axis card that is running. The *DMRGetCpuStatus* command will return the state of the axis card.

The *DMRDownload* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRDownload(cpu,filename)

unsigned int cpu;	cpu device number
char far *filename;	file name

For related commands see:

DMRGetCpuStatus

3.5. DMRGetCpuStatus

This function returns the status of the specified cpu. It returns a one (1) if the monitor boot ROM is running. A two (2) indicates that the axis firmware is running. A value of three (3) means that the axis card has experienced a general board failure and must be returned to DMR for servicing. Firmware may be downloaded if the return value is equal to one.

The *DMRGetCpuStatus* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetCPUStatus(cpu,stat)

unsigned int cpu; cpu device number

unsigned int far *stat; cpu status

For related commands see:

DMRDownload

3.6. DMRSetParm

The *DMRSetParm* function is a generic function call used by other DMR #defines to set parameter values into parameters specified by the opcode. It should not be necessary for the user to use this form of the call. It is included in the documentation for reference only. Issuing *DMRSetParm* calls with invalid opcodes will cause programming faults.

It allows the setting of multiple axes with different values. The variable *value can be a pointer to an array of long numbers or it can be a pointer to one number.

This command will loop on the number of axes called out in the bitwise axis mask variable 'axis'. The first axis will have its parameter set to the number to which value is pointing. The next axis will have its parameter set to the number pointed to by the pointer value after the pointer is incremented. This continues until there are no more axes left in the axis mask.

Care must be taken to have at least as many numbers in the array pointed to by value as there are axes in the axis mask. Otherwise, axes can be set with invalid numbers.

The *DMRSetParm* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetParm(axis,opcode,value)

unsigned long axis;	bit mask of axes
int opcode;	operation code of this call
long value;	value being passed

unsigned far Pascal DMRSetMParm(axis,opcode,value)

unsigned long axis;	bit mask of axes
int opcode;	operation code of this call
long far *value;	value being passed

3.7. DMRGetParm

The *DMRGetParm* function is a generic function call used by other DMR #defines to get parameter values from the axis card specified by the opcode. It should not be necessary for the user to use this form of the call. It is included in the documentation for reference only. Issuing *DMRGetParm* calls with invalid opcodes will cause programming faults.

This command is also used to return the following masks:

OPCODE	MASK	
	0x01	Status (see DMRGetAxisStat)
	0x41	Fault (see DMRGetFault)
	0x61	Motion
	0x62	Servo

The *DMRGetParm* funcuon returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetParm(axis,opcode,value)

unsigned long axis;	bit mask of the axes
int opcode;	mode command for this call
long far *value;	value to be passed back

For related commands see:

DMRSetParm

DMRSetMParm

3.8. DMRSendIntSem

This function allows the user to pass a system semaphore to the device driver. This semaphore is associated with the axes specified in the variable axis. When one of these axes goes into a fault condition, the axis card will cause a VME interrupt and this semaphore will be cleared.

This must be performed before enabling axis card interrupts or the system will lock up.

The application program should create a thread that blocks on this system semaphore. The thread will create and set the semaphore, and then send the handle to the DMR device driver. It should then block on the semaphore and perform any desired recovery functions when the semaphore is cleared.

Whenever one of the axes in the mask causes an interrupt, the device driver will clear the system semaphore signaling the application program that an interrupt occurred.

There may be up to 16 different interrupt semaphores sent to the driver each with its own axis mask. Axes' may be associated with more than one semaphore.

The application program must clean up these semaphores upon closing with a *DMRClearIntSem* command. If these semaphores are still registered and an interrupt condition occurs on the axis card after the application program has terminated the system will lock up.

The *DMRSendIntSem* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSendIntSem(axis,handle)

unsigned long axis;	bit mask of axes
unsigned long handle;	handle of semaphore

For related commands see:

DMRGetSemMask

DMRSendCommSem

3.9. DMRClearIntSem

This function allows the user to remove an interrupt system semaphore from the list maintained by the device driver.

Interrupt semaphores must be removed with this command or the *DMRClearAllSem* command before the application program terminates. If this is not done and an interrupt condition occurs, the system will lock up.

There may be up to 16 different interrupt semaphores sent to the driver each with its own axis mask. Axes may be associated with more than one semaphore.

The *DMRClearIntSem* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRClearIntSem(handle)

unsigned long handle; handle of semaphore

For related commands see:

DMRGetSemMask

DMRSendCommSem

3.10. DMRClearAllSem

This function allows the user to remove all interrupt system semaphores from the list maintained by the device driver.

Interrupt semaphores must be removed with this command or individually with the *DMRClearIntSem* command before the application program terminates. If this is not done and an interrupt condition occurs the system will lock up.

There may be up to 16 different interrupt semaphores sent to the driver each with its own axis mask. Axes may be associated with more than one semaphore.

The *DMRClearAllSem* function returns a zero if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRClearAllSem(void)

For related commands see:

DMRGetSemMask

DMRSendCommSem

3.11. DMRSendSysFailSem

This function allows the user to pass a system semaphore to the device driver. This semaphore is associated with the SysFail error of the VME Bus.

This command must be performed after the *DMROpen* and before any other calls. If not, communication errors will lock up the system.

The application program should create a thread that blocks on this system semaphore. The thread will create a system semaphore, set it, send the handle to the DMR device driver. It will then block on it until a SysFail error occurs.

Whenever a SysFail error occurs, the device driver will clear the system semaphore signaling the application program of this event.

The *DMRSendSysFailSem* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSendSysFailSem(handle)

unsigned long handle; handle of semaphore

For related commands see:

DMRSendIntSem

3.12. DMRClearSysFailSem

This function allows the user to remove the SysFail interrupt system semaphore from the list maintained by the device driver.

The *DMRClearSysFailSem* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRClearSysFailSem(void)

For related commands see:

DMRGetSemMask

DMRSendCommSem

3.13. DMRSendCommSem

This function allows the user to pass a system semaphore to the device driver. This semaphore is associated with axis communication errors.

This command must be performed after the *DMROpen* and before any other calls. If not, communication errors will lock up the system.

The application program should create a thread that blocks on this system semaphore. The thread will create a system semaphore, set it, and send the handle to the DMR device driver. It will then block on it until a communication error occurs.

Whenever a communication error occurs, the device driver will clear the system semaphore signaling the application program of this event.

The *DMRSendCommSem* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSendCommSem(handle)

unsigned long handle; handle of semaphore

For related commands see:

DMRSendIntSem

3.14. DMRClearCommSem

This function allows the user to remove the communication interrupt system semaphore from the list maintained by the device driver.

The *DMRClearCommSem* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRClearCommSem(handle)

unsigned long far handle; handle of semaphore

For related commands see:

DMRGetSemMask

DMRSendCommSem

3.15. DMRGetSemMask

This function passes a system semaphore to the device driver and the driver returns an axis mask of the axes associated with that semaphore.

There may be up to 16 different interrupt semaphores sent to the driver each with its own axis mask. Axes may be associated with more than one semaphore.

The *DMRGetSemMask* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetSemMask(handle,axis)

unsigned long far *axis; bit mask of axes

unsigned long handle; handle of semaphore

For related commands see:

DMRSendIntSem

DMRSendCommSem

3.16. DMRGetFaultVector

This function returns the last fault vector sent to the DMR device driver on an interrupt acknowledge cycle.

When the device driver gets more than one interrupt before being asked for this vector, it OR's the vectors together.

The vector returned to the driver specifies which axis causes the interrupt. If the vector returned to the application has more than one axis indicated, then multiple axes have faulted since the last occurrence of this call. The device driver clears its vector holding variable after this call is issued.

Each interrupt semaphore has an axis mask associated with it and a vector variable for holding the last interrupt vector that occurred. When an interrupt occurs, the driver takes that vector (really an axis mask) and OR's it into the vector variable associated with the semaphore that has that axis mask. If an axis appears in more than one semaphore axis mask, then all of those vector variables associated with that semaphore will be updated.

There may be up to 16 different interrupt semaphores with corresponding vector variables.

The *DMRGetFaultVector* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetFaultVector(handle,vector)

unsigned long handle; handle for the semaphore

unsigned long far *vector; last interrupt vector

For related commands see:

DMRSendIntSem

DMRGetSemMask

3.17. DMRSendMouseSem

This function allows the user to pass a system semaphore to the device driver. This semaphore is associated with the functionality of a mouse connected to one of the RS-232 channel residing on the Axis Processor board.

The application program should create a thread that blocks on this system semaphore. The thread will create a system semaphore, set it, send the handle to the DMR device driver. It will then block on it until mouse button is pressed.

Whenever the mouse button is pressed, the device driver will clear the system semaphore signaling the application program of this event.

The *DMRSendMouseSem* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSendMouseSem(handle)

unsigned long handle; handle of semaphore

For related commands see:

DMRSendIntSem

3.18. DMRClearMouseSem

This function allows the user to remove the mouse interrupt system semaphore from the list maintained by the device driver.

The *DMRClearMouseSem* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRClearMouseSem(handle)

unsigned long far handle; handle of semaphore

For related commands see:

DMRGetSemMask

DMRSendMouseSem

3.19. DMRGetFreeMemory

The *DMRGetFreeMemory* returns the number of bytes available on the axis card and the largest contiguous section of free memory.

The *DMRGetFreeMemory* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetFreeMemory(cpu,bytes,largest)

unsigned int cpu; axis processor number

unsigned long far *bytes; number of free bytes remaining

unsigned long far *largest; largest contiguous block

3.20. DMRConfigAxis

This command must be executed on power-up to activate the axis. No motion will be allowed until this command is executed. This command allows the user to set several parameters about each axis which generally do not change during normal operation.

The channel variables allow the user to configure any R/D and D/A section with any axis. The allowable numbers are between one and sixteen.

The resolution of the resolver R/D section in bits and the number of AC motor poles (zero for DC) are also set here. The resolution can be 10, 12, 14, or 16. If the Poles variable is set to zero, then no commutation will be performed (i.e. for a DC motor).

CommOffset is the commutation offset value used for Brushless Motors only. This value is set to zero for Brush Motors. Refer to Motor Manufacturer's Manual to obtain this value if necessary.

The bound variable tells the axis whether it is going to run in a continuous mode or whether there will be end of travel limits to monitor. When bounding is off, the axis card will not fault on position rollover, and will not recognize software end of travel settings. However hardware end of travel limits are still active.

The *DMRConfigAxis* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

```
unsigned far Pascal DMRConfigMAxis(axis_mask,table)
unsigned long axis_mask;          bitwise axis mask
struct ConfigAxisStruct {
   int R2DChannel;                channel # for this axis
   int D2AChannel;                channel # for this axis
   int Resolution;          bits used on R/D
   int Poles;                     motor poles
   unsigned int CommOffset;       offset for commutation
   int Bound;                     1-on, 0-off
   } far *table;
```

or

unsigned far Pascal DMRConfigAxis(axis_mask,R2DChannel,D2AChannel,Resolution,Poles,CommOffset,Bound)

unsigned long axis_mask; bitwise axis mask

int R2DChannel; channel # for this axis

int D2AChannel; channel # for this axis

int Resolution; bits used on R/D

int Poles; motor poles

unsigned int CommOffset; offset for commutation

int Bound; 1-on, 0-off

For related commands see:

DMRGetConfigAxis

3.21. DMRGetConfigAxis

This command returns the configuration of the specified axis. A description of each parameter may be found under the *DMRConfigAxis* command.

The *DMRGetConfigAxis* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetConfigMAxis(axis_mask,table)

unsigned long axis_mask; bitwise axis mask

struct config_axis_struct {

int R2D_channel; channel # for this axis

int D2A_channel; channel # for this axis

int Resolution; bits used on R/D

int Poles; motor poles

unsigned int CommOffset; offset for commutation

int Bound; 1-on, 0-off

} *table;

or

unsigned far Pascal DMRGetConfigAxis(axis_mask,R2D_channel,D2A_channel,resolution,poles,comm_offset,bound)

unsigned long axis_mask; bitwise axis mask

int far *R2D_channel; channel # for this axis

int far *D2A_channel; channel # for this axis

int far *resolution; bits used on R/D

int far *poles; motor poles

unsigned int far *comm_offset;	offset for commutation
int far *bound;	1-on, 0-off

For related commands see:

DMRConfigAxis

3.22. DMRConfigEncoder

This command must be executed on power-up to activate the axis. No motion will be allowed until this command is executed. This command allows the user to set several parameters about each axis which generally do not change during normal operation.

The channel variables allow the user to configure any Encoder and D/A section with any axis. The allowable numbers are between one and sixteen. The Lines variable specifies the encoder lines per motor revolution.

The bound variable tells the axis whether it is going to run in a continuous mode or whether there will be end of travel limits to monitor. When bounding is off, the axis card will not fault on position rollover, and it will not recognize software end of travel settings. However hardware end of travel limits are still active.

The *DMRConfigEncoder* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal
DMRConfigEncoder(axis_mask,EncoderChannel,D2AChannel,Lines,Bound)

unsigned long axis_mask;	bitwise axis mask
int EncoderChannel;	channel # of encoder
int D2AChannel;	channel # for this axis
unsigned long far Lines;	lines on encoder
int Bound;	1-on, 0-off

For related commands see:

DMRGet*ConfigEncoder*

3.23. DMRGetConfigEncoder

This command returns the configuration of the specified axis. A description of each parameter may be found under the *DMRConfigEncoder* command.

The *DMRGetConfigEncoder* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal
DMRGetConfigEncoder(axis_mask,EncoderChannel,D2AChannel,Lines,Bound)

unsigned long axis_mask;	bitwise axis mask
int *EncoderChannel;	channel # of encoder
int *D2AChannel;	channel # for this axis
unsigned long far *Lines;	lines on encoder
int *Bound;	1-on, 0-off

For related commands see:

DMR*ConfigEncoder*

3.24. DMRConfig

The *DMRConfig* axis command is used to configure the axis specified in the axis_mask. This is a more generic version of *DMRConfigaxis* and *DMRConfigEncoder*. Feedback type of device is specified in FBType, where current options are:

Bit#	Symbol	Bit Definition
0	FB_NULL	no feedback
1	FB_RESOLVER	resolver feedback
2	FB_ENCODER	encoder feedback

If Resolver feedback is selected, then Channel, Resolution, Poles, CommOffset, and Bounded are specified in Resolver substructure. See *DMRConfigAxis* for meaning of these arguments.

If Encoder feedback is selected, then Channel, Number of Lines, and Bounded is specified in Encoder substructure. See *DMRConfigEncoder* for meaning of these arguments.

I/O type of device used to command the Drives is specified in IOType, where current options are:

Bit#	Symbol	Bit Definition
0	IO_NULL	no command
1	IO_D2A	Digital to Analog

If Digital to Analog Converter is used to send the command signal to drives, Channel number is selected.

The *DMRConfig* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRConfig(axis_mask,config)

unsigned long axis_mask;

struct ConfigPacket *config;

{refer to DMRHEAD.H header file for definition of ConfigPacket structure}

For related commands see:

DMRGetConfig

3.25. DMRGetConfig

This command returns the configuration of the specified axis. A description of each parameter may be found under the *DMRConfig* command.

The *DMRGetConfig* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal unsigned far Pascal DMRGetConfig(axis_mask,config)

unsigned long axis_mask;

struct ConfigPacket *config;

{refer to DMRHEAD.H header file for definition of ConfigPacket structure}

For related commands *see:*

DMRConfig

3.26. DMRGetInfo

The *DMRGetInfo* function returns a null terminated ASCII string of various device information. This information includes software generation time, board ID#, copyright notice, etc. The info string is 256 bytes long.

The *DMRGetInfo* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetInfo(cpu,info)

unsigned int cpu; axis processor number

char far *info; general information

For related commands see:

DMRGetVersion

DMRGetPartNumber

DMRGetDate DMRGetVersion

3.27. DMRGetPartNumber

The *DMRGetPartNumber* function returns the part number of the firmware.

The *DMRGetPartNumber* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetPartNumber(cpu,info)

unsigned int cpu;	axis processor number
unsigned long far *info;	general information

For related commands see:

DMRGetVersion

DMRGetInfo

DMRGetDate

DMRGetVersion

3.28. DMRGetVersion

The *DMRGetVersion* function returns the software version number of the specified cpu module.

The *DMRGetVersion* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetVersion(cpu,version)

unsigned int cpu;	axis processor number
unsigned long far *version;	software version

For related commands see:

DMRGetInfo

DMRGetDate

DMRGetPartNumber

3.29. DMRGetDate

The *DMRGetDate* function returns the creation date of the axis firmware.

The *DMRGetDate* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetDate(cpu,info)

unsigned int cpu;	axis processor number
unsigned long far *info;	general information

For related commands see:

DMRGetVersion

DMRGetInfo DMRGetPartNumber

DMRGetVersion

3.30. DMRGetParmCount

This command returns the number of parameters in the axis card.

The *DMRGetParmCount* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetParmCount(axis_mask,count)

unsigned long axis_mask;	bitwise axis mask
unsigned int far *count;	number of parameters

For related commands see:

DMRGetParmDef

3.31. DMRGetParmDef

This command passes the parameter number to the axis card and receives the name of the parameter, a status word, and the minimum and maximum allowed values.

The status word contains a bit mask of the status of that variable. The following bit masks are defined in the file *DMRDEF.H*:

Bit#	Symbol	Bit Definition
0x01	PARM_VALID	valid parameter or not
0x02	PARM_READ	parameter may be read
0x04	PARM_WRITE	parameter may be written
0x08	PARM_UPDATE	axis card will update periodically
0x10	PARM_NOLIMIT	if set then no limit checks done
0x20	PARM_TEST	this is a DMR test parameter

The *DMRGetParmDef* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

```
unsigned far Pascal DMRGetParmDef(axis_mask,parmnum,name)
unsigned long axis_mask;              bitwise axis mask
int parmnum;                          parameter number
struct PDef{
  char Name[14];              name of the parameter
  int Status;                         valid, read, write, etc.
  long Min;                           minimum allowed value
  long Max;                           maximum allowed value
 } DMRParmDef far *name;
```

For related commands see:

DMRGetParmCount

3.32. DMRGetProgFault

The *DMRGetProgFault* function returns information about the command that caused the last programming fault on the specified axis. Each axis keeps track of the command that caused the last programming error.

ErrorCode contains an error code number. The ErrorMessage contains a text string explaining the cause of the fault. ErrorLength contains the length of the data in ErrorData. ErrorData contains debug information relating to the fault, i.e. the last invalid command received in hex format.

The *DMRGetProgFault* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

```
unsigned far Pascal DMRGetProgFault(axis,table)
unsigned long axis;                    axis processor number
ProgFaultStruct {
 unsigned int ErrorCode;               error code number
 char ErrorMessage[80];                ASCII error message
 unsigned int ErrorLength;             message length in bytes
 unsigned char ErrorData[80];          hex error data
} DMRProgFaultStruct far *table;
```

For related commands see:

DMRFlushProgFault

3.33. DMRFlushProgFault

The *DMRFlushProgFault* function clears the buffer in the axis card that holds the last command that caused a programming fault on the specified axis. There is a separate buffer for each axis.

The *DMRFlushProgFault* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRFlushProgFault(axis)

unsigned long axis; axis processor number

For related commands see:

DMRGetProgFault

3.34. DMRGetMessage

The *DMRGetMessage* command is used to retrieve ASCII strings which define the function of bits for system faults, axis status, servo status, and motion status.

Type of message is specified in type_mess where current values are:

type_mess	type of message
0	fault message
1	status message
2	servo message
3	motion

Message number can range from 0 to 31 for each type of message and are specified in mess_num. These numbers correspond to the bit numbers of the corresponding mask type.

Mess_ptr is a pointer to the character string of the message. Maximum message size is 32 characters.

The *DMRGetMessage* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetMessage(type_mess,mess_num,mess_str)

unsigned short type_mess; type of message

unsigned short mess_num; message number

char far *mess_str; message string pointer

For related commands see:

DMRGetFaultMesssage
DMRGetStatusMessage
DMRGetServoMessage
DMRGetMotionMessage

3.35. DMRGetFaultMessage

The *DMRGetFaultMessage* function retrieves ASCII strings which define the function of bits for system faults. The messages are stored in axis processor memory. Message number is specified in mess_num. Refer to *DMRGetFault* for message number and fault message association. Mess_ptr is a pointer to the character string of the message. Maximum message size is 32 characters.

Refer to the *DMRGetFault* command for fault mask bit definition.

The DMRGetFaultMessage function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetFaultMessage(mess_num,mess_str)

unsigned short mess_num; ;message number

char far *mess_str; ;message string pointer

For related commands see:

DMRGetMessage
DMRGetStatusMessage
DMRGetServoMessage
DMRGetMotionMessage

3.36. DMRGetStatusMessage

The *DMRGetStatusMessage* function retrieves the ASCII strings which define the function of bits for axis status. The messages are stored in axis processor memory. Message number is specified in mess_num. Mess_ptr is a pointer to the character string of the message.

Refer to the DMRGetAxisStat command for status mask bit definition.

The *DMRGetStatusMessage* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetStatusMessage(mess_num,mess_str)

unsigned short mess_num; ;message number

char far *mess_str; ;message string pointer

For related commands see:

DMRGetMessage
DMRGetFaultMessage
DMRGetServoMessage
DMRGetMotionMessage

3.37. DMRGetServoMessage

The *DMRGetServoMessage* function retrieves the ASCII strings which define the function of bits for servo status. The messages are stored in axis processor memory. Message number is specified in mess_num. Mess_ptr is a pointer to the character string of the message.

The *DMRGetParm* command is used to return the servo status mask. Following is the bit definition of the servo mask:

Bit#	Servo Definition
0x00000001	drive enabled
0x00000002	aux output enable
0x00000004	CW input active
0x00000008	CCW input active
0x00000010	home input active
0x00000020	drive fault active
0x00000040	at home
0x00000080	done status
0x00000100	inposition status
0x00000200	faulted status
0x00000400	probe input
0x0000080	marker input
0x00001000	hall input 1
0x00002000	hall input 2
0x00004000	hall input 3
0x00008000	hall input 4

The *DMRGetServoMessage* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetServoMessage(mess_num,mess_str)

unsigned short mess_num; ;message number

char far *mess_str; ;message string pointer

For related commands see:

DMRGetMessage,
DMRGetFaultMessage
DMRGetStatusMessage,
DMRGetMotionMessage

3.38. DMRGetMotionMessage

The *DMRGetMotionMessage* function retrieves the ASCII strings which define the function of bits for motion status. The messages are stored in axis processor memory. Message number is specified in mess_num. Mess_ptr is a pointer to the character string of the message.

The *DMRGetParm* command is used to return the motion status mask. Following is the bit definition of the servo mask:

Bit#	Motion Definition
0x00000001	direction of motion
0x00000002	moving
0x00000004	in accel phase
0x00000008	in decel phase
0x00000010	homing
0x00000020	feedrate override
0x00000040	in profile mode
0x00000080	in sync mode
0x00000100	cam table enabled
0x00000200	homing direction
0x00000400	continuous move
0x00000800	queued command
0x00001000	hold on
0x00002000	aux table mode
0x00004000	block motion commands
0x00008000	hold queue

The *DMRGetMotionMessage* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetMotionMessage(mess_num,mess_str)

unsigned short mess_num; ;message number

char far *mess_str; ;message string pointer

For related commands see:

DMRGetMessage
DMRGetFaultMessage
DMRGetStatusMessage
DMRGetServoMessage

3.39. DMRWriteNVRam

The *DMRWriteNVRam* function writes 'count' number of bytes to the non-volatile ram area. The variable 'offset' is used to index into the table before beginning the write.

The table can hold up to 1024 bytes. If a write will exceed the upper table boundary, then an error code will be returned.

The *DMRReadNVRam* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRWriteNVRam(offset,count,ptr)

unsigned int offset;	offset into NV Ram table
unsigned int count;	number of bytes to write
void far *ptr;	pointer to info

For related commands see:

DMRReadNVRam

3.40. DMRReadNVRam

The *DMRReadNVRam* function reads 'count' number of bytes from the non-volatile ram area. The variable 'offset' is used to index into the table before beginning the read. The table can hold up to 1024 bytes. If a read will exceed the upper table boundary, then an error code will be returned.

The *DMRReadNVRam* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRReadNVRam(offset,count,ptr)

unsigned int offset;	offset into NV Ram table
unsigned int count;	number of bytes to write
void far *ptr;	pointer to info

For related commands see:

DMRWriteNVRam

3.41. DMRAllocateStrip

The *DMRAllocateStrip* allocates an area of ram in the axis card for storing a 'strip chart recording' of instantaneous position, position command, and torque output.

The DMRAllocateStrip function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRAllocateStrip(axis,size)

unsigned long axis; axis mask

unsigned int size; size of strip table

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.42. DMRFreeStrip

The *DMRFreeStrip* frees the space allocated by a previous DMRAllocateStrip command.

The *DMRFreeStrip* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRFreeStrip(axis)

unsigned long axis; axis mask

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.43. DMRHaltStrip

The *DMRHaltStrip* command is used to terminate data capture command. Once this command is executed, DMRSetStripTrigger is used to initialize data capture.

The *DMRHaltStrip* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRHaltStrip(axis)

unsigned long axis; axis mask

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.44. DMRTriggerStrip

The *DMRTriggerStrip* command performs a one shot trigger of the strip chart function at the time the axis card receives this command.

The *DMRTriggerStrip* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRTriggerStrip(axis)

unsigned long axis; axis mask

3.45. DMRSetStripTrigger

The *DMRSetStripTrigger* sets the strip chart operation to begin depending on the mode set. The variable timebase will determine how often to take a sample.

Mode values are:

0:enables immediate trigger.
1:time base trigger

Parm1 and *parm2* are undefined at this time.

The *DMRSetStripTrigger* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetStripTrigger(axis,mode,timebase,parm1,parm2)

unsigned long axis;	axis mask
unsigned int mode;	strip chart mode
unsigned int timebase;	sample every x ms
long parm1;	undefined
long parm2;	undefined

For related commands see:

DMRAllocateStrip

DMRGetStripSample

3.46. DMRGetStripStatus

The *DMRGetStripStatus* returns the bit mapped status of the strip chart recorder on the axis card. It also returns the number of samples allocated.

The following is the bit map description of the status word:

bit	mask	definition
0x0001	STRIP_ALLOCATED	allocated or not 1,0
0x0002	STRIP_ARMED	armed or not 1,0
0x0004	STRIP_TRIGGERED	triggered or not 1,0
0x0008	STRIP_DONE	done or not 1,0

The *DMRGetStripStatus* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetStripStatus(axis,size,status,parm1, parm2)

unsigned long axis;	axis mask
unsigned int far *size;	size of strip table
unsigned int far *status;	status of strip chart
unsigned int far *parm1;	undefined
unsigned int far *parm2;	undefined

For related commands see:

DMRAllocateStrip

DMRGetStripSample

3.47. DMRGetStripSample

The *DMRGetStripSample* returns the values recorded starting at the specified position in the table. More than one sample may be read at a time. The values are put into the array pointed at by ptr.

The DMRGetStripSample function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetStripSample(axis,start_point,size,ptr)

unsigned long axis;	axis mask
unsigned int start_point;	point to start reading
unsigned int size;	number of points to read
DMRStripStruct far *ptr;	pointer to strip structure

For related commands see:

DMRAllocateStrip

3.48. DMRAllocateGlobalStrip

The *DMRAllocateGlobalStrip* allocates an area of ram in the axis card for storing a 'strip chart recording' of instantaneous position, position command, position with no error correction, master position, velocity, velocity command, acceleration, and torque of the axis specified in axis_mask. Number of samples for each axis is specified in size.

More than one axis can be specified.

The *DMRAllocateGlobalStrip* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRAllocateGlobalStrip(axis,size)

unsigned long axis;	axis mask
unsigned int size;	size of strip table

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.49. DMRAllocateGlobalStripTrigger

The *DMRAllocateGlobalStripTrigger* allocates an area of ram in the axis card for storing axis positions. These are axis positions which act as a trigger for axis data to be logged. This command is used for position based trigger mode. Refer to *DMRAllocateGlobalStrip* for axis data information. Size specifies the number of positions to be used for triggering.

The *DMRAllocateGlobalStripTrigger* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRAllocateGlobalStripTrigger(size)

unsigned int size; size of strip table

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.50. DMRSetGlobalStripTriggerPoint

The *DMRSetGlobalStripTriggerPoint* command is used to enter position data in the table allocated using *DMRAllocateGlobalStripTrigger*. Point specifies entry number into table. Data specifies the axis trigger position.

The *DMRSetGlobalStripTriggerPoint* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetGlobalStripTriggerPoint(point,data)

unsigned int point; table entry number

unsigned int size; axis trigger position

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.51. DMRGetGlobalStripTriggerPoint

The *DMRGetGlobalStripTriggerPoint* is used to obtain status information about an entry in the position trigger table. This table is allocated using the *DMRAllocateGlobalStripTrigger* command. Point specifies an entry number to table. Data specifies axis trigger position.

The following is a bit map description of the status word:

Bit#	Symbol	Definition
0x0001	STRIP_ALLOCATED	allocated or not 1,0
0x0002	STRIP_ARMED	armed or not 1,0
0x0004	STRIP_TRIGGERED	triggered or not 1,0
0x0008	STRIP_DONE	done or not 1,0

The *DMRGetGlobalStripTriggerPoint* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetGlobalStripTriggerPoint(point,data,status)

unsigned int point;	entry into trigger table
long far *data;	trigger position
unsigned int far *status;	trigger point status

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.52. DMRGetGlobalStripTriggerStatus

The *DMRGetGlobalStripTriggerStatus* command is used to obtain overall trigger table status.

The following is a bit map description of the status word:

Bit#	Symbol	Definition
0x0001	STRIP_ALLOCATED	allocated or not 1,0
0x0002	STRIP_ARMED	armed or not 1,0
0x0004	STRIP_TRIGGERED	triggered or not 1,0
0x0008	STRIP_DONE	done or not 1,0

Size specifies the number of entries in trigger table.

The *DMRGetGlobalStripTriggerStatus* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetGlobalStripTriggerStatus(status,size)

unsigned int *status; status of trigger points

unsigned int *size; # of entries in trigger table

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.53. DMRFreeGlobalStrip

The *DMRFreeGlobalStrip* command is used to deallocate memory used by the strip table. The table was previously allocated using the *DMRAllocateGlobalStrip*.

The *DMRFreeGlobalStrip* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRFreeGlobalStrip()

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.54. DMRFreeGlobalStripTrigger

The *DMRFreeGlobalStripTrigger* command is used to free memory used by the trigger table. The table was previously allocated using *DMRAllocateGlobalStripTrigger*.

The *DMRFreeGlobalStripTrigger* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRFreeGlobalStripTrigger()

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.55. DMRHoldGlobalStrip

The *DMRHoldGlobalStrip* command is used to suspend data capture mode until a *DMRReleaseGlobalStrip* command is executed.

The *DMRHoldGlobalStrip* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRHoldGlobalStrip()

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.56. DMRReleaseGlobalStrip

The *DMRReleaseGlobalStrip* command is used to resume data capture mode previously suspended using the *DMRHoldGlobalStrip*.

The *DMRReleaseGlobalStrip* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRReleaseGlobalStrip()

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.57. DMRHaltGlobalStrip

The *DMRHaltGlobalStrip* command is used to terminate data capture command. Once this command is executed, *DMRSetGlobalStripTrigger* is used to reinitialize data capture.

The *DMRHaltGlobalStrip* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRHaltGlobalStrip()

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.58. DMRSetGlobalStripTrigger

The *DMRSetGlobalStripTrigger* command is used to start data capture mode.

Data Capture type is specified in mode, where:

0 - time based
1 - time based with speed override
2 - position based

Time specifies interval between samples. Used only in time based modes.

Size specifies number of samples to capture.

Parm1 is used to select master axis if position based capture is used.

Parm2 is not used.

The *DMRSetGlobalStripTrigger* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetGlobalStripTrigger(mode,time,size,parm1,parm2)
unsigned int mode; data capture mode
unsigned int time; time between samples
unsigned int size; number of samples
long parm1; master axis mask
long parm2; not used

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.59. DMRGetGlobalStripStatus

The *DMRGetGlobalStripStatus* command is used to obtain data capture process information.

The following is a bit map description of the status word:

Bit#	Symbol	Definition
0x0001	STRIP_ALLOCATED	allocated or not 1,0
0x0002	STRIP_ARMED	armed or not 1,0
0x0004	STRIP_TRIGGERED	triggered or not 1,0
0x0008	STRIP_DONE	done or not 1,0

Allocated specifies the size of table.

Size specifies number of samples to capture.

Collected specifies number of samples currently stored in table.

Axis_mask specifies axis whose values will be captured.

The *DMRGetGlobalStripTrigger* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal
DMRGetGlobalStripStatus(status,table_size,sample_size,collected,axis_mask)

unsigned int far *status;	status of trigger points
unsigned int far *table_size;	size of strip table
unsigned int far *sample_size;	number of samples
unsigned int far *collected;	number of samples stored
unsigned int far *axis_mask;	bitwise axis mask

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.60. DMRGetGlobalStripSample

The *DMRGetGlobalStripSample* command is used to get access to captured axis data. This command limits data size retrieved to 64k. The *DMRGetHugeGlobalStripSample* is used for larger data segments. First specifies sample number to begin reading. Count specifies number of samples to read.

Axis_mask specifies axis.

Ptr is a pointer to Data capture array. Refer to *DMRHEAD.H* file for description of *DMRGlobalPtrList* data structure.

The *DMRGetGlobalStripSample* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

```
unsigned far Pascal DMRGetGlobalStripSample(first,count,axis_mask,ptr)
unsigned int first;                 sample number to start
unsigned int count;                       number of samples to get
unsigned long axis_mask;                  bitwise axis mask
struct DMRGlobalPtrList far *ptr;   data captured
```

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.61. DMRGetHugeGlobalStripSample

The *DMRGetHugeGlobalStripSample* command is used to get captured axis data when size goes beyond one segment or 64k memory. First specifies sample number to begin reading. Count specifies number of samples to read.

Axis_mask specifies axis.

Ptr is a pointer to Data capture array. Refer to *DMRHEAD.H* file for description of *DMRGlobalPtrList* data structure.

The *DMRGetHugeGlobalStripSample* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

```
unsigned far Pascal DMRGetHugeGlobalStripSample(first,count,axis_mask,ptr)
unsigned int first;                 sample number to start
unsigned int count;                         number of samples to get
unsigned long axis_mask;                    bitwise axis mask
struct DMRGlobalPtrList far *ptr;   data captured
```

For related commands see:

DMRSetStripTrigger

DMRGetStripSample

3.62. DMRSetUserFault

The *DMRSetUserFault* allows an application program to cause a VME bus interrupt. This command causes the axis card to initiate a VME interrupt cycle. The fault and interrupt masks for this axis must be enabled for the FLT_USER bit in the mask.

The *DMRSetUserFault* function returns a (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetUserFault(axis)

unsigned long axis; axis mask

3.63. DMRReset

This function resets the axis processor by forcing it to restart from the boot EPROM. This allows the application to download the axis firmware again.

Firmware may not be downloaded to an axis card that is running. The *DMRGetCpuStatus* command will return the state of the axis card.

The *DMRReset* function returns a (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRReset(cpu)

unsigned int cpu; cpu device number

For related commands see:

DMRGetCpuStatus

DMRDownLoad

∇ ∇ ∇

CHAPTER 4: MOTION FUNCTIONS

4.1. DMRMoveIncremental

The *DMRMoveIncremental* function causes the axes specified in the axis_mask to index a distance and rate specified in variables length and speed or in the array structure length_speed. The length and speed used are in units of machine steps and machine steps/second of the axis.

The variable axis_mask is a bit mask of which axes are to be commanded. If multiple axes are specified in the mask, those axes start moving at the same time. This move is not coordinated. The axes will follow the currently selected acceleration and deceleration mode (linear or sinusoidal).

This command is cumulative. The distance will be added onto any currently executing move. A currently executing move will immediately take on the new rate defined in the variable speed. If the axis is not being commanded to move, the distance is added onto the commanded position. The length variable sign determines the direction of the move. The absolute value of the speed parameter will always be taken. An error value will be returned if the axes specified in the axis mask are not enabled.

The *DMRMoveIncremental* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRMoveIncremental(axis_mask,length,speed);

unsigned long axis_mask; bitwise axis mask

long length; index length

unsigned long speed; index speed

or

unsigned far Pascal DMRMoveMIncremental(axis_mask,length_speed);

unsigned long axis_mask; bitwise axis mask

struct MoveStruct {

long length; index length

```
    unsigned long speed;          index speed
    } DMRMoveStruct *length_speed;
```

For related commands see:

DMRMoveAbsolute

DMRQueueMoveIncremental

DMRHaltMotion

DMRAbortMotion, DMRHoldMotion

DMRSetAccelTime

DMRSetDecelTime

4.2. DMRMoveAbsolute

The *DMRMoveAbsolute* function causes the axes specified in the axis_mask to move to a target position with a given speed. The position and rate are specified in the variables target and speed or in the array structure target_speed. These variable are in user units of machine steps and machine steps/second.

The variable axis_mask is a bit mask of which axes are to be commanded. If multiple axes are specified in the mask, those axes start moving at the same time. This is not a coordinated move. The axes will follow whatever acceleration and deceleration mode is currently selected (linear or sinusoidal). Any currently executing motion command is abandoned. The axes immediately move towards the target.

The length variable sign determines the direction of the move. The absolute value of the speed parameter will always be taken. An error code will be returned if the specified axes are not enabled.

The *DMRMoveAbsolute* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRMoveAbsolute(axis_mask,target,speed);

unsigned long axis_mask; bitwise axis mask

long target; target position

unsigned long speed; target speed

or

unsigned far Pascal DMRMoveMAbsolute(axis_mask,target_speed);

unsigned long axis_mask; bitwise axis mask

struct MoveStruct {

long target; target position

unsigned long speed; target speed

} *target_speed;

For related commands see:

DMRMoveIncremental

DMRQueueMoveAbsolute

DMRHaltMotion

DMRAbortMotion

DMRHoldMotion

DMRSetAccelTime

DMRSetDecelTime

4.3. DMRStartMotion

The *DMRStartMotion* function causes the axes specified in the axis_mask to start moving in the direction specified at the given speed. The speed is in units of machine steps/second. This command is generally used in an application where continuous motion is needed and there are no end of travel settings. The axes will follow whatever acceleration and deceleration mode is currently selected (linear or sinusoidal). If the axis is intended to be operated in a continuous mode, then the bounded flag should be cleared in the *DMRConfigAxis* command.

The *DMRStartMotion* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRStartMotion(axis_mask,direction,speed);
unsigned long axis_mask; bitwise axis mask
long direction; cw, ccw
unsigned long speed; speed

or

unsigned far Pascal DMRStartMMotion(axis_mask,dir_speed);
unsigned long axis_mask; bitwise axis mask
struct MoveStruct {
long direction; cw, ccw
unsigned long speed; speed
} *dir_speed;

For related commands see:

DMRMoveIncremental

DMRMoveAbsolute

DMRMoveHome

DMRHaltMotion

DMRAbortMotion

DMRHoldMotion

DMRSetAccelTime

DMRSetAccelMode

DMRGetAxisStat

DMRConfigAxis

4.4. DMRMoveHome

The *DMRMoveHome* function causes the axes specified in the axis_mask to seek their home position. Each axis will begin moving to the home position in the direction and speed specified in the variables direction and speed or in the array structure direction_speed. The speed used is in user units of machine steps/second.

The variable axis_mask is a bit mask of which axes are to be commanded. If multiple axes are specified in the mask, those axes start moving at the same time. The axes will follow whatever acceleration and deceleration mode is currently selected (linear or sinusoidal). Any currently executing motion command is abandoned. The axes immediately move towards their home positions. A direction that is positive or zero will start the homing in the clockwise motor direction while a negative value causes a counter-clockwise homing move.

The homing procedure is as follows: the axis starts moving in the homing direction specified. If it crosses the home switch it will stop on the next resolver null it sees and set the position of that axis to zero. If it hits an end of travel limit, it will reverse. If it then hits a home switch, it will go past it, reverse and come at it from the specified direction. If it encounters another end of travel, a homing fault will occur.

The *DMRMoveHome* function returns 0 if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRMoveHome(axis_mask,direction,speed);

unsigned long axis_mask; bitwise axis mask

long direction; home direction

unsigned long speed; home speed

or

unsigned far Pascal DMRMoveMHome(axis_mask,direction_speed);

unsigned long axis_mask; bitwise axis mask

struct MoveStruct {

long dir; home direction

unsigned long speed; home speed

} *direction_speed;

For related commands see:

DMRMoveQuickHome
DMRMoveIncremental
DMRMoveAbsolute
DMRHaltMotion
DMRAbortMotion
DMRHoldMotion
DMRSetAccelTime
DMRSetAccelMode

4.5. DMRMoveHomeRev

The *DMRMoveHomeRev* instruction replaces the *DMRMoveHome* function in the case where the limit switch is used for calibration.

The *DMRMoveHomeRev* function causes the axes specified in the axis_mask to seek their home position. Each axis will begin moving to the home position in the direction and speed specified in the variables direction and speed or in the array structure direction_speed. The speed used is in user units of machine steps/second.

The variable axis_mask is a bit mask of which axes are to be commanded. If multiple axes are specified in the mask, those axes start moving at the same time. The axes will follow whatever acceleration and deceleration mode is currently selected (linear or sinusoidal). Any currently executing motion command is abandoned. The axes immediately move towards their home positions. A direction that is positive or zero will start the homing in the clockwise motor direction while a negative value causes a counter-clockwise homing move.

The homing procedure is as follows: the axis starts moving in the homing direction specified. If it crosses the home switch it will reverse direction and stop on the next resolver null or encoder marker pulse it sees and set the position of that axis to zero. If it hits an end of travel limit, it will reverse. If it then hits a home switch, it will go past it and stop on the next resolver null or encoder marker pulse it sees. If it encounters another end of travel, a homing fault will occur.

The *DMRMoveHome* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRMoveHomeRev(axis_mask,direction,speed);

unsigned long axis_mask;	bitwise axis mask
long direction;	home direction
unsigned long speed;	home speed

or

```
unsigned far Pascal DMRMoveMHomeRev(axis_mask,direction_speed);
unsigned long axis_mask;          bitwise axis mask
struct MoveStruct {
    long dir;                     home direction
    unsigned long speed;          home speed
    } *direction_speed;
```

For related commands see:

DMRMoveQuickHome
DMRMoveIncremental
DMRMoveAbsolute
DMRHaltMotion
DMRAbortMotion
DMRHoldMotion
DMRSetAccelTime
DMRSetAccelMode

4.6. DMRMoveQuickHome

The *DMRMoveQuickHome* function causes the axes specified in the axis_mask to seek their home position. Each axis will begin moving to the home position in the direction and speed specified in the variables direction and speed or in the array structure direction_speed. The speed used is in machine steps/second.

The variable axis_mask is a bit mask of which axes are to be commanded. If multiple axes are specified in the mask, those axes start moving at the same time. The axes will follow whatever acceleration and deceleration mode is currently selected (linear or sinusoidal). Any currently executing motion command is abandoned. The axes immediately move towards their home positions. A direction that is positive or zero will start the homing in the clockwise motor direction while a negative value causes a counter-clockwise homing move.

The normal homing process is not completed but the home switch can be found quickly using this function. Afterwards, a *DMRMoveHome* command can be issued at a slower speed to complete the process.

The *DMRMoveQuickHome* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

```
unsigned far Pascal DMRMoveQuickHome(axis_mask,direction,speed);
unsigned long axis_mask;          bitwise axis mask
long direction;                   home direction
unsigned long speed;              home speed
                or
unsigned far Pascal DMRMoveMQuickHome(axis_mask,direction_speed);
unsigned long axis_mask;          bitwise axis mask
struct MoveStruct {
   long dir;                      home direction
   unsigned long speed;           home speed
   } *direction_speed;
```

For related commands see:

DMRMoveHome

4.7. DMRQueueMoveIncremental

This is a queued version of the *DMRMoveIncremental* function. The is a queue 16 deep for each axis. If a move is in progress, the queued move will not begin to execute until the current move reaches its decel phase. If there is no pending move, then the queued move will begin immediately.

The *DMRQueueMoveIncremental* function causes the axes specified in the axis_mask to index a distance and rate specified in variables length and speed. The length and speed used are in units of machine steps and machine steps/second. The variable axis_mask is a bit mask of which axes are to be commanded. The axes will follow the currently selected acceleration and deceleration mode (linear or sinusoidal).

This command is cumulative. The distance will be added onto any currently executing move. A currently executing move will immediately take on the new rate defined in the variable speed. If the axis is not being commanded to move, the distance is added onto the commanded position. The length variable sign determines the direction of the move. The absolute value of the speed parameter will always be taken. An error value will be returned if the axes specified in the axis mask are not enabled.

The *DMRQueueMoveIncremental* function returns 0 if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRQueueMoveIncremental(axis_mask,length,speed);

unsigned long axis_mask;	bitwise axis mask
long length;	index length
unsigned long speed;	index speed

For related commands see:

DMRQueueMoveIncremental
DMRFlushMoveQueue
DMRHoldQueue
DMRReleaseQueue

4.8. DMRQueueMoveAbsolute

This is a queued version of the *DMRMoveAbsolute* function. The is a queue 16 deep for each axis. If a move is in progress, the queued move will not begin to execute until the current move reaches its decel phase. If there is no pending move, then the queued move will begin immediately.

The *DMRMoveAbsolute* function causes the axes specified in the axis_mask to move to a target position with a given speed. The position and rate are specified in the variables target and speed. or in. These variable are in user units of machine steps and machine steps/second.

The variable axis_mask is a bit mask of which axes are to be commanded. If multiple axes are specified in the mask, those axes start moving at the same time. This is not a coordinated move. The axes will follow whatever acceleration and deceleration mode is currently selected (linear or sinusoidal). Any currently executing motion command is abandoned. The axes immediately move towards the target.

The length variable sign determines the direction of the move. The absolute value of the speed parameter will always be taken. An error code will be returned if the specified axes are not enabled.

The *DMRQueueMoveAbsolute* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRQueueMoveAbsolute(axis_mask,target,speed);

unsigned long axis_mask;	bitwise axis mask
long target;	target position
unsigned long speed;	target speed

For related commands see:

DMRQueueMoveIncremental
DMRFlushMoveQueue
DMRHoldQueue
DMRReleaseQueue

4.9. DMRMoveLinear

This command causes the specified axes to move along each axis' specified distance at vector rate defined by the speed variable. It is a coordinated move meaning that all axes will start and stop at the same time using the acceleration and deceleration times specified in the command. Previously defined individual axis times will not be used during this move.

Each vector distance specified in the variable vector will correspond to the next axis specified in the bit mask. Vector and speed are in units of machine steps and machine steps/second. This command is valid for any number of active axes specified in axis_mask. If more than one axis is called out, there will be the same number of occurrences of the vector variable. Any currently executing motion command is abandoned. The axes immediately move towards the targets. The vector variable always determines the direction of the move. The absolute value of the speed parameter will always be taken.

The axes will follow whatever acceleration and deceleration mode is currently selected (linear or sinusoidal). The absolute value of the vector speed variable will be taken. If the number of vector variables does not agree with the number of axes specified, a programming error will be returned.

The *DMRMoveLinear* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRMoveLinear(axis_mask,vector_speed);

unsigned long axis_mask;	bitwise axis mask
struct MoveLinStruct {	
long vector;	linear vector
unsigned long speed;	vector speed
unsigned long acc_time;	acceleration time
unsigned long dec_time;	deceleration time
} *vector_speed;	

For related commands see:
DMRMoveIncremental
DMRMoveAbsolute
DMRHaltMotion, DMRHoldMotion
DMRAbortMotion, DMRSetAccelTime, DMRSetAccelMode

4.10. DMRHaltMotion

The *DMRHaltMotion* function causes the axes specified in the axis_mask to stop their motion in a controlled fashion. The axes will follow whatever deceleration mode is currently selected (linear or sinusoidal). It is not coordinated with any other axis command.

The *DMRHaltMotion* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRHaltMotion(axis_mask);

unsigned long axis_mask; bitwise axis mask

For related commands see:

DMRMoveIncremental

DMRMoveAbsolute

DMRMoveHome

DMRAbortMotion

DMRHoldMotion

DMRSetDecelTime

DMRSetDecelMode

4.11. DMRAbortMotion

The *DMRAbortMotion* function causes the axes specified in the axis_mask to stop their motion immediately. When this function is used the axes will used the current observed position as the final command position. Deceleration time, if used, does not affect this function.

The *DMRAbortMotion* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRAbortMotion(axis_mask);

unsigned long axis_mask; bitwise axis mask

For related commands see:

DMRHaltMotion

DMRHoldMotion

4.12. DMRHoldMotion

The *DMRHoldMotion* function causes the axes specified in the axis_mask to suspend their motion in a controlled fashion. The axes will follow whatever acceleration and deceleration mode is currently selected (linear or sinusoidal).

A *DMRHoldMotion* function issued to an axis not in motion will have an effect. Any motion commands issued to that axis before a *DMRReleaseMotion* command will be put into a queue that is one deep. Once a *DMRReleaseMotion* command is issued, the last commanded move will be executed and any others ignored.

The *DMRHoldMotion* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRHoldMotion(axis_mask);

unsigned long axis_mask; bitwise axis mask

For related commands see:

DMRHaltMotion

DMRAbortMotion

DMRMoveIncremental

DMRMoveAbsolute

DMRMoveHome

DMRSetAccelTime

DMRSetAccelMode

DMRGetAxisStat

DMRReleaseMotion

4.13. DMRReleaseMotion

The *DMRReleaseMotion* function causes the axes specified in the axis_mask to continue the motion that was previously suspended by a *DMRHoldMotion* function. The axes will follow whatever acceleration and deceleration mode is currently selected (linear or sinusoidal). A release function directed to an axis not in a hold mode will have no effect. The last command issued to an axis under hold will be executed.

The *DMRReleaseMotion* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

int DMRReleaseMotion(axis_mask);

unsigned long axis_mask; bitwise axis mask

For related commands see:

DMRMoveIncremental

DMRMoveAbsolute

DMRMoveHome

DMRHaltMotion

DMRAbortMotion

DMRHoldMotion

DMRSetAccelTime

DMRSetAccelMode

DMRGetAxisStat

4.14. DMRFlushMoveQueue

The *DMRFlushMoveQueue* function causes the 16 deep motion queue of the axis specified in the axis_mask to be cleared. Entries to the queue are set up using the *DMRQueueMoveIncremental* and *DMRQueueMoveAbsolute* commands.

The *DMRFlushMoveQueue* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

int DMRFlushMoveQueue(axis_mask);

unsigned long axis_mask; bitwise axis mask

For related commands see:

DMRQueueMoveIncremental

DMRQueueMoveAbsolute

DMRHoldQueue

DMRReleaseQueue

4.15. DMRHoldQueue

The *DMRHoldQueue* function prevents next queued motion of the axis specified in the axis_mask from executing until a *DMRReleaseQueue* is executed. Any currently executing motion command is completed.

The *DMRHoldQueue* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

int DMRHoldQueue(axis_mask);

unsigned long axis_mask; bitwise axis mask

For related commands see:

DMRQueueMoveIncremental

DMRQueueMoveAbsolute
DMRFlushMoveQueue
DMRReleaseQueue

4.16. DMRReleaseQueue

The *DMRReleaseQueue* function allows for queued moves of the axis specified in the axis_mask to resume execution that was previously suspended by a *DMRHoldQueue* function.

The *DMReleaseQueue* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

int DMRReleaseQueue(axis_mask);

unsigned long axis_mask; bitwise axis mask

For related commands see:

DMRQueueMoveIncremental

DMRQueueMoveAbsolute
DMRFlushMoveQueue
DMRHoldQueue

4.17. DMRSetJogMode

The *DMRSetJogMode* passes control to an external switching device used to move the axis specified in the axis_mask.

Modes are defined as follows:
0 - disable
1 - enable

(Refer to Hardware Manual for further information)

The *DMRSetJogMode* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetJogMode(axis_mask,mode,speed);

unsigned long axis_mask;	bitwise axis mask
unsigned short mode;	1-enable, 0-disable
unsigned long speed;	speed

For related commands see:

DMRGetJogMode

4.18. DMRGetJogMode

The *DMRGetJogMode* returns the status of an external switching device used to move the axis specified in the axis_mask.

Modes are defined as follows:
0 - disable
1 - enable

The *DMRGetJogMode* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRMTN

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetJogMode(axis_mask,mode,speed);
unsigned long axis_mask; bitwise axis mask
unsigned short *mode; 1-enable, 0-disable
unsigned long *speed; speed

For related commands see:

DMRSetJogMode

∇ ∇ ∇

CHAPTER 5: SYNCHRONIZED MOTION FUNCTIONS

5.1. DMRAllocateCamTable

This function allocates storage on the axis path processor for user cam tables. This allows storage of multiple cam table on one path processor. The number of cam tables available depends on the amount of storage on the path processor and the size of each cam.

There will be a total of 50 cam tables available for allocation and one megabyte of ram available for points. Cam tables are numbered 0 through 49. Cam tables are not allocated to any specific axis. Masters are associated with slaves and not with tables.

The cam table starts at point number zero and ends with point number 'cam_size' minus one. Entering a point at point number -1 is allowed to enable the user to specify the slope of the line at the first point. This is used when cubic spline interpolation is specified. The point at 'cam_size' is also allowed to specify the slope of the line at the last point in the table.

CPU = 0 currently only valid processor number.

The *DMRAllocateCamTable* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRSYNC

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRAllocateCamTable(cpu,cam_number,cam_size)

unsigned int cpu; axis processor number

unsigned int cam_number; cam table number

unsigned int cam_size; cam size in points

5.2. DMRFreeCamTable

This frees the memory associated with a previously allocated cam table.

The *DMRFreeCamTable* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRSYNC

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRFreeCamTable(cpu,cam_number)

unsigned int cpu; axis processor number

unsigned int cam_number; cam table number

5.3. DMRWriteCamPoint

The *DMRWriteCamPoint* function writes one point into the specified cam table. Points are in units machine steps.

Master points in the cam table must be non-repeating. The type of interpolation is specified in the type parameter where a zero designates linear interpolation and a one indicates cubic interpolation.

Points are entered starting at table position zero and ending at table position 'table size' minus one. Points may be entered at table position -1 and at 'table size' to allow specification of the slope entering and exiting the table..

The *DMRWriteCamPoint* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRSYNC

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRWriteCamPoint(cpu,cam_table,point,master,slave,type)

unsigned int cpu;	axis processor number
unsigned int cam_table;	cam table number
unsigned int point;	point number
long master_pos;	master position
long slave_pos;	slave position
unsigned int type;	point type 0 : linear, 1 : cubic

For related commands see:

DMRReadCamPoint

5.4. DMRReadCamPoint

The *DMRReadCamPoint* function reads one point from the specified cam table. Points are in units of machine steps. The coefficients are also returned.

The *DMRReadCamPoint* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRSYNC

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRReadCamPoint(cpu,cam_table,point,master_pos,slave_pos,a,b,c,type)

unsigned int cpu;	axis processor number
unsigned int cam_table;	cam table number
unsigned int point;	point number to read from
long master_pos;	master position
long slave_pos;	slave position
long a;	a coefficient
long b;	b coefficient
long c;	c coefficient

unsigned int type;point type 0: linear, 1: cubic

For related commands see:

DMRWriteCamPoint

5.5. DMRWriteMultCamPoints

The *DMRWriteMultCamPoints* function writes multiple cam points into the specified cam table. Points are in units of machine steps. Master points in the cam table must be non-repeating.

The type of interpolation is specified in the type parameter where a zero designates linear interpolation and a one indicates cubic interpolation.

Points are entered starting at table position zero and ending at table position 'table size' minus one. Points may be entered at table position -1 and at 'table size' to allow specification of the slope entering and exiting the table..

The *DMRWriteMultCamPoints* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRSYNC

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRWriteMultCamPoints(cpu,cam_table,point,num,ptr)

unsigned int cpu; axis processor number

unsigned int cam_table; cam table number

unsigned int point; starting point number

unsigned int num; number of points to write

DMRWriteCamStruct far *ptr; pointer to cam array

For related commands see:

DMRReadCamPoint

5.6. DMRReadMultCamPoints

The *DMRReadCamPoints* function reads multiple points from the specified cam table. Points are in units of machine steps. The coefficients are also returned.

The *DMRReadMultCamPoints* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRSYNC

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRReadMultCamPoints(cpu,cam_table,point,num,ptr)

unsigned int cpu; axis processor number

unsigned int cam_table; cam table number

unsigned int point; starting point number to read from

unsigned int num; number of points to read

DMRReadCamStruct far *ptr; pointer to array of read structure

For related commands see:

DMRWriteCamPoint

5.7. DMRCalcCoeff

The *DMRCalcCoeff* function instructs the axis processor to calculate coefficients for the specified cam table. Linear or cubic coefficients will be generated based on the type in each point entry.

This process is complete when the DMRGetCoeffStatus function returns a complete code. In order to calculate coefficients on multiple cam tables, the user must wait for the previous *DMRCalcCoeff* function to be complete.

The *DMRCalcCoeff* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRSYNC

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRCalcCoeff(cpu,cam_table)

int cpu; axis processor number

int cam_table; cam table number

5.8. DMRGetCamTableStatus

The *DMRGetCamTableStatus* returns the status of the coefficient generation procedure on the path processor. A status of zero indicates the path processor is not calculating coefficients and the coefficients are not valid for that cam table. When a one is returned, the path processor is calculating coefficients and can not accept another *DMRCalcCoeff* command.

A status of two indicates that the coefficients have been calculated and are valid for that table. The axis processor may now accept more *DMRCalcCoeff* commands. This function also returns the number of points allocated for this table.

The *DMRGetCamTableStatus* function returns 0 if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRSYNC

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetCamTableStatus(cpu,cam_table,points,status)

unsigned int cpu;	axis processor number
unsigned int cam_table;	cam table number
unsigned int far *points;	number of points in this table
unsigned int far *status;	status of cam table coefficients

5.9. DMRConfigMaster

This command is sent to a slave axis to associate a master R2D section with it for following as the **master axis**. This selection is utilized during sync mode operation for tracking between master and slave axes.

This command determines the master for motion sync operations as well as I/O (auxiliary output) operations. Multiple axes may associated with the same R2D section.

The *DMRConfigMaster* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRConfigMaster(axis_mask,R2DChannel,Resolution)

unsigned long axis_mask;	bitwise axis mask
int R2DChannel;	channel # for this axis
int Resolution;	bits used on R/D

For related commands see:

DMRGetConfigMaster

5.10. DMRGetConfigMaster

This command is used to retrieve the master R2D section associated with this slave axis.

The *DMRGetConfigMaster* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRUTIL

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetConfigMaster(axis_mask,R2DChannel,Resolution)

unsigned long axis_mask;	bitwise axis mask
int far *R2DChannel;	channel # for this axis
int far *Resolution;	bits used on R/D

For related commands see:

DMRConfigMaster

5.11. DMRSetSyncMode

The *DMRSetSyncMode* function causes the specified slave axes to enter or exit synchronized mode. Association of slave axes with cam tables is done here. Master axis for each slave axis is defined in the DMRConfigMaster command. A programming error will be returned if the associated axes are not enabled when the modes are turned on.

Synchronized modes defined as follows:

0 - off

1 - cam mode

2 - ratio mode

Cam mode causes the axes to assume that wherever they are when the *DMRSetSyncMode* command is executed should be associated with the beginning of the table.

Ratio mode causes the slave axis to assume that its present position is at the start of the cam table. It then reads the position of the master axis and moves the appropriate offset until its position corresponds with the master position in the table. It will perform this index using the speed set with the command *DMRSetSyncSpeed.*

The *DMRSetSyncMode* function returns 0 if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRSYNC

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetSyncMode(axis_mask,cam_table,mode)

unsigned long axis_mask; bitwise mask of slave axes

unsigned int cam_table; cam table number

unsigned int mode; sync mode

5.12. DMRGetSyncMode

The *DMRGetSyncMode* function returns the mode of the specified axis. The cam table number is also returned.

Synchronized modes defined as follows:

0 - off

1 - cam mode

2 - ratio mode

The *DMRGetSyncMode* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRSYNC

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetSyncMode(axis_mask,cam_table,mode)

unsigned long axis_mask;	bitwise mask of slave axes
unsigned int far *cam_table;	cam table number
unsigned int far *mode;	sync mode

5.13. DMRReferenceMaster

The *DMRReferenceMaster* function associates a position value with a specified resolver position value in order to eliminate buildup of error in a sync application.

The *DMRReferenceMaster* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRSYNC

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRReferenceMaster(axis_mask,res_pos,pos)

unsigned long axis_mask;	bitwise axis mask
long res_pos;	infeed length
long pos;	infeed speed

5.14. DMRInfeedSlave

The *DMRInfeedSlave* function, when issued in synchronized mode, causes the specified axis to infeed. A *DMRInfeedSlave* issued while not in synchronized mode will cause a fault. This allows the user to shift the cam table in the slave direction only for that slave axis. Any other slave axes associated with the cam table that this axis is assigned to are left unaltered. The sign of the length variable determines the direction in which to shift the table. The position and speed are in units of machine steps. The absolute value of the speed variable will be taken. If the axis is not enabled, a programming error will be returned.

The *DMRInfeedSlave* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRSYNC

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRInfeedSlave(axis_mask,length,speed)

unsigned long axis_mask;	bitwise axis mask
long length;	infeed length
unsigned long speed;	infeed speed

∇ ∇ ∇

CHAPTER 6: PROFILING MOTION FUNCTIONS

6.1. DMRStartProfileQueue

The *DMRStartProfileQueue* function causes the specified axis to begin motion of their respective queued profile data. Profile data is loaded using the *DMRLoadGlobalProfile* and *DMRLoadProfileQueue*.

The *DMRStartProfileQueue* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPROF

or

#define INCL_DMRALL

#include "dmrhead.h"

usigned far Pascal DMRStartProfileQueue(axis_mask);

unsigned long axis_mask; bitwise axis mask

For related commands see:

DMRStopProfileQueue

DMRFlushProfileQueue

6.2. DMRStopProfileQueue

The *DMRStopProfileQueue* function causes specified axis to suspend motion of the current profile until the *DMRStartProfileQueue* is executed.

The *DMRStopProfileQueue* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPROF

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRStopProfileQueue(axis_mask);

unsigned long axis_mask; bitwise axis mask

For related commands see:

DMRStartProfileQueue

DMRFlushProfileQueue

6.3. DMRFlushProfileQueue

The *DMRFlushProfile*Queue function causes the profile queue of the axis specified in the axis_mask to be cleared. Entries to the queue are set up using the *DMRLoadGlobalProfile* and *DMRLoadProfileQueue* commands.

The *DMRFlushProfileQueue* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPROF

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRFlushProfileQueue(axis_mask);

unsigned long axis_mask; bitwise axis mask

For related commands see:

DMRLoadGlobalProfile

DMRLoadProfileQueue

6.4. DMRLoadGlobalProfile

The *DMRLoadGlobalProfile* function causes the profile data to be queued to the respective axis. This command is similar to the *DMRLoadProfileQueue*, but allows for multiple axis profiles to be queued at the same time.

The *DMRLoadGlobalProfile* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPROF

or

#define INCL_DMRALL

#include dmrhead.h"

```
unsigned far Pascal DMRLoadGlobalProfile(axis_mask,profile)
unsigned long axis_mask;          bitwise axis mask
struct ProfileStruct{
  long Point;                     destination
  ULONG Time;           motion time
  long VelocityStart;             start velocity
  long VelocityEnd;               end velocity
  }*profile                       PTR to structure
```

For related commands see:

DMRLoadGlobalProfle

DMRFlushProfileQueue

6.5. DMRLoadProfileQueue

The *DMRLoadProfileQueue* function causes the profile data to be queued for the specified axis. This command is similar to *DMRLoadGlobalProfile*, but allows only one axis data to be queued at a time.

The *DMRLoadProfileQueue* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPROF

or

#define INCL_DMRALL

#include dmrhead.h"

```
unsigned far Pascal LoadProfileQueue(axis_mask,prof_point)
unsigned long axis_mask;          bitwise axis mask
struct ProfileStruct{
  long Point;                     destination
  ULONG Time;           motion time
  long VelocityStart;             start velocity
  long VelocityEnd;               end velocity
  }*prof_point                    PTR to structure
```

For related commands see:

DMRLoadGlobalProfile

DMRFlushProfileQueue

∇ ∇ ∇

CHAPTER 7: PARAMETER FUNCTIONS

7.1. DMRGetAxisStat

The *DMRGetAxisStat* function returns status information for the axes specified in axis_mask.

The status long word contains the present state of each of these:

Bit#	Status Definition	Bit#	Status Definition
0x00000001	actual drive on	0x00010000	move direction
0x00000002	actual aux output	0x00020000	axis in motion
0x00000004	actual cw limit	0x00040000	axis in accel phase
0x00000008	actual ccw limit	0x00080000	axis in decel phase
0x00000010	actual home	0x00100000	homing executing
0x00000020	drive fault input	0x00200000	feedrate override
0x00000040	axis at home	0x00400000	profile mode
0x00000080	axis done	0x00800000	axis in sync mode
0x00000100	axis in position	0x01000000	cam table enable
0x00000200	axis faulted	0x02000000	homing direction
0x00000400	probe input	0x04000000	continuous move
0x00000800	marker input	0x08000000	motion queue active
0x00001000	hall input #1	0x10000000	hold active
0x00002000	hall input #2	0x20000000	aux mode
0x00004000	hall input #3	0x40000000	block motion
0x00008000	hall input #4	0x80000000	hold queue active

The *DMRGetAxisStat* function returns (0) if successful and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetAxisStat(axis_mask,status)

unsigned long axis_mask; bitwise axis mask

unsigned long far *status; status for each axis

For related commands see:

DMRGetFault

DMRSetFault

7.2. DMRSetAccelTime

The *DMRSetAccelTime* function is used to control the acceleration time of any movement command. Any move will complete its acceleration in this amount of time.

Default: [0]

Range: [0 - 100000]

Units: milliseconds

The *DMRSetAccelTime* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetAccelTime(axis_mask,time)

unsigned long axis_mask; bitwise axis mask

long time; time in milliseconds

or

unsigned far Pascal DMRSetMAccelTime(axis_mask,time)

unsigned long axis_mask; bitwise axis mask

long far *time; time in milliseconds

For related commands see:

DMRGetAccelTime

DMRSetDecelTime

DMRGetDecelTime

DMRSetAccelMode

DMRGetAccelMode

DMRSetDecelMode

DMRGetDecelMode

7.3. DMRGetAccelTime

The *DMRGetAccelTime* function returns the acceleration time previously set with the *DMRSetAccelTime* function.

The *DMRGetAccelTime* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetAccelTime(axis_mask,time)

unsigned long axis_mask;	bitwise axis mask
long far *time;	time in milliseconds

For related commands see:

DMRSetAccelTime

DMRSetDecelTime

DMRGetDecelTime

DMRSetAccelMode

DMRGetAccelMode

DMRSetDecelMode

DMRGetDecelMode

7.4. DMRSetDecelTime

The *DMRSetDecelTime* function is used to control the deceleration time of any movement command. Any move will complete its deceleration in this amount of time.

Default: [0]

Range: [0 - 100000]

Units: milliseconds

The *DMRSetDecelTime* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetDecelTime(axis_mask,time)

unsigned long axis_mask; bitwise axis mask

long time; time in milliseconds

or

unsigned far Pascal DMRSetMDecelTime(axis_mask,time)

unsigned long axis_mask; bitwise axis mask

long far *time; time in milliseconds

For related commands see:

DMRGetDecelTime

DMRSetAccelTime

DMRGetAccelTime

DMRSetAccelMode

DMRGetAccelMode

DMRSetDecelMode

DMRGetDecelMode

7.5. DMRGetDecelTime

The *DMRGetDecelTime* function returns the deceleration time previously set with the *DMRSetDecelTime* function.

The *DMRGetDecelTime* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetDecelTime(axis_mask,time)

unsigned long axis_mask;	bitwise axis mask
long far *time;	time in milliseconds

For related commands see:

DMRSetDecelTime

DMRSetAccelTime

DMRGetAccelTime

DMRSetAccelMode

DMRGetAccelMode

DMRSetDecelMode

DMRGetDecelMode

7.6. DMRSetAccelMode

The *DMRSetAccelMode* function is used to control the acceleration mode on motion functions.

Default: [0]

Range: [0,1] 0 : 1-cosine, 1 : linear

Units: NA

The *DMRSetAccelMode* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetAccelMode(axis_mask,mode)

unsigned long axis_mask; bitwise axis mask

long mode; acceleration mode

or

unsigned far Pascal DMRSetMAccelMode(axis_mask,mode)

unsigned long axis_mask; bitwise axis mask

long far *mode; acceleration mode

For related commands see:

DMRGetAccelMode

DMRSetDecelMode

DMRGetDecelMode

DMRSetDecelTime

DMRGetDecelTime

DMRSetAccelMode

DMRGetAccelMode

7.7. DMRGetAccelMode

The *DMRGetAccelMode* function returns the acceleration mode previously set with the DMRSetAccelMode function.

The *DMRGetAccelMode* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetAccelMode(axis_mask,mode)

unsigned long axis_mask;	bitwise axis mask
long far *mode;	acceleration mode

For related commands see:

DMRSetAccelMode

DMRSetDecelMode

DMRGetDecelMode

DMRSetAccelTime

DMRGetAccelTime

DMRSetDecelTime

DMRGetDecelTime

7.8. DMRSetDecelMode

The *DMRSetDecelMode* function is used to control the deceleration mode on motion functions.

Default: [0]

Range: [0,1] 0 : 1-cosine, 1 : linear

Units: NA

The *DMRSetDecelMode* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetDecelMode(axis_mask,mode)

unsigned long axis_mask; bitwise axis mask

long mode; acceleration mode

or

unsigned far Pascal DMRSetMDecelMode(axis_mask,mode)

unsigned long axis_mask; bitwise axis mask

long far *mode; acceleration mode

For related commands see:

DMRGetDecelMode

DMRSetAccelMode

DMRGetAccelMode

DMRSetDecelTime

DMRGetDecelTime

DMRSetAccelMode

DMRGetAccelMode

7.9. DMRGetDecelMode

The *DMRGetDecelMode* function returns the deceleration mode previously set with the *DMRSetDecelMode* function.

The *DMRGetDecelMode* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetDecelMode(axis_mask,mode)

unsigned long axis_mask;	bitwise axis mask
long far *mode;	acceleration mode

For related commands see:

DMRSetDecelMode

DMRSetAccelMode

DMRGetAccelMode

DMRSetAccelTime

DMRGetAccelTime

DMRSetDecelTime

DMRGetDecelTime

7.10. DMRSetProportionalGain

The *DMRSetProportionalGain* function sets the proportional gain of the velocity loop for the specified axes.

Refer to the DMR 960 Hardware Manual for a description of how this parameter functions in the servo block diagram.

Default: [10000]

Range: [0 - 100,000]

Units: NA

The *DMRSetProportionalGain* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetProportionalGain(axis_mask,pgain)

unsigned long axis_mask; bitwise axis mask

long pgain; proportional gain

or

unsigned far Pascal DMRSetMProportionalGain(axis_mask,pgain)

unsigned long axis_mask; bitwise axis mask

long far *pgain; proportional gain

For related commands see:

DMRGetProportionalGain

DMRSetIntegralGain

DMRGetIntegralGain

DMRSetPositionGain

DMRGetPositionGain

7.11. DMRGetProportionalGain

The *DMRGetProportionalGain* function returns the proportional gain of the velocity loop for the specified axes.

The *DMRGetProportionalGain* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetProportionalGain(axis_mask,pgain)

unsigned long axis_mask;	bitwise axis mask
long far *pgain;	proportional gain

For related commands see:

DMRSetProportionalGain

DMRSetIntegralGain

DMRGetIntegralGain

DMRSetPositionGain

DMRGetPositionGain

7.12. DMRSetIntegralGain

The *DMRSetIntegralGain* function sets the integral gain of the velocity loop for the specified axes.

Refer to the DMR 960 Hardware Manual for a description of how this parameter functions in the servo block diagram.

Default: [2000]

Range: [0 - 10,000]

Units: NA

The *DMRSetIntegralGain* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetIntegralGain(axis_mask,igain)

unsigned long axis_mask; bitwise axis mask

long far *igain; integral gain

or

unsigned far Pascal DMRSetMIntegralGain(axis_mask,igain)

unsigned long axis_mask; bitwise axis mask

long far *igain; integral gain

For related commands see:

DMRGetIntegralGain

DMRSetProportionalGain

DMRGetProportionalGain

DMRSetPositionGain

DMRGetPositionGain

7.13. DMRGetIntegralGain

The *DMRGetIntegralGain* function returns the integral gain of the velocity loop for the specified axes.

The *DMRGetIntegralGain* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetIntegralGain(axis_mask,igain)

unsigned long axis_mask;	bitwise axis mask
long far *igain;	integral gain

For related commands see:

DMRSetIntegralGain

DMRSetProportionalGain

DMRGetProportionalGain

DMRSetPositionGain

DMRGetPositionGain

7.14. DMRSetPositionGain

The *DMRSetPositionGain* function sets the gain of the position loop for the specified axes.

Refer to the DMR 960 Hardware Manual for a description of how this parameter functions in the servo block diagram.

Default: [10]

Range: [0 - 1000]

Units: NA

The *DMRSetPositionGain* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetPositionGain(axis_mask,posgain)

unsigned long axis_mask; bitwise axis mask

long posgain; position loop gain

or

unsigned far Pascal DMRSetMPositionGain(axis_mask,posgain)

unsigned long axis_mask; bitwise axis mask

long far *posgain; position loop gain

For related commands see:

DMRGetPositionGain

DMRSetIntegralGain

DMRGetIntegralGain

DMRSetProportionalGain

DMRGetProportionalGain

7.15. DMRGetPositionGain

The *DMRGetPositionGain* function returns the gain of the position loop for the specified axes.

The *DMRGetPositionGain* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetPositionGain(axis_mask,posgain)

unsigned long axis_mask;	bitwise axis mask
long far *posgain;	position loop gain

For related commands see:

DMRSetPositionGain

DMRSetIntegralGain

DMRGetIntegralGain

DMRSetProportionalain

DMRGetProportionalGain

7.16. DMRSetVelocityFeedForward

The *DMRSetVelocityFeedForward* function enables or disables the velocity feed forward function. Enabling this function minimizes position following error.

Default: [0]

Range: [0,1] 0 : disable, 1 : enable

Units: NA

The *DMRSetVelocityFeedForward* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetVelocityFeedForward(axis_mask,switch)

unsigned long axis_mask; bitwise axis mask

long switch; enable/disable switch

or

unsigned far Pascal DMRSetMVelocityFeedForward(axis_mask,switch)

unsigned long axis_mask; bitwise axis mask

long far *switch; enable/disable switch

For related commands see:

DMRGetVelocityFeedForward

DMRSetAccelerationForward

DMRGetAccelerationFeedForward

7.17. DMRGetVelocityFeedForward

The *DMRGetVelocityFeedForward* function returns the state of the velocity feed forward function. Zero means off, non-zero means on.

The *DMRGetVelocityFeedForward* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetVelocityFeedForward(axis_mask,state)

unsigned long axis_mask;	bitwise axis mask
long far *state;	enabled/disabled

For related commands see:

DMRGetVelocityFeedforward

DMRSetAccelerationForward

DMRGetAccelerationFeedForward

7.18. DMRSetAFFGain

This function allows the user to set a gain value on the acceleration feedforward term.

Default: [1]

Range: [1-1000]

Units: none

The *DMRSetAFFGain* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetAFFGain(axis_mask,switch)

unsigned long axis_mask; bitwise axis mask

long switch; value

or

unsigned far Pascal DMRSetMAFFGain(axis_mask,switch)

unsigned long axis_mask; bitwise axis mask

long far *switch; value

7.19. DMRGetAFFGain

This function allows the user to retrieve the gain value on the acceleration feedforward term.

Default: [1]

Range: [1-1000]

Units: none

The *DMRSetAFFGain* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetAFFGain(axis_mask,switch)

unsigned long axis_mask; bitwise axis mask

long far *switch; value

7.20. DMRSetBlockMotion

Calling this function with a value of one causes the specified axis to ignore any commands that would initiate motion. The only exception would be if the axis is already under control of a sync table. Calling the function with a value of zero will allow the specified axis to again accept motion commands. While blocking is enabled, the axis will accept commands that will stop motion such as *DMRHaltMotion* so that an axis already in motion may be stopped.

The *DMRSetBlockMotion* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetBlockMotion(axis_mask,parm)

unsigned long axis_mask; bitwise axis mask

long parm; on/off

or

unsigned far Pascal DMRSetBlockMotion(axis_mask,parm)

unsigned long axis_mask; bitwise axis mask

long far *parm; on/off

7.21. DMRGetBlockMotion

This function will return the state of the blocking for the specified axis as described in the *DMRSetBlockMotion* command.

The *DMRGetBlockMotion* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetBlockMotion(axis_mask,parm)

unsigned long axis_mask;	bitwise axis mask
long far *parm;	on/off

7.22. DMRSetCurrentLimit

The *DMRSetCurrentLimit* function sets the peak current limit for the specified axes.

This sets the peak output voltage on the current command D/A's.

Default: [32767]

Range: [0 - 32767]

Units: 10 volts / 32768 counts

The *DMRSetCurrentLimit* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetCurrentLimit(axis_mask,ilimit)

unsigned long axis_mask; bitwise axis mask

long ilimit; current limit

or

unsigned far Pascal DMRSetMCurrentLimit(axis_mask,ilimit)

unsigned axis_mask; bitwise axis mask

long far *ilimit; current limit

For related commands see:

DMRGetCurrentLimit

DMRSetCurrentFaultLimit

DMRGetCurrentFaultLimit

DMRSetCurrentFaultTime

DMRGetCurrentFaultTime

7.23. DMRGetCurrentLimit

The *DMRGetCurrentLimit* function returns the current limit for the specified axes.

The *DMRGetCurrentLimit* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetCurrentLimit(axis_mask,ilimit)

unsigned long axis_mask; bitwise axis mask

long far *ilimit; current limit

For related commands see:

DMRGetCurrentLimit

DMRSetCurrentFaultLimit

DMRGetCurrentFaultLimit

DMRSetCurrentFaultTime

DMRGetCurrentFaultTime

7.24. DMRSetCurrentFaultLimit

The *DMRSetCurrentFaultLimit* function sets a parameter used to detect over current operation. The instantaneous current is averaged over a time specified by the function *DMRSetCurrentFaultTime*. If the RMS average exceeds the limit set by this command, a fault will be generated.

Default: [32767]

Range: [0 - 32767]

Units: 10 volts / 32767 counts

The *DMRSetCurrentFaultLimit* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetCurrentFaultLimit(axis_mask,ilimit)

unsigned long axis_mask; bitwise axis mask

long ilimit; current ilimit

or

unsigned far Pascal DMRSetMCurrentFaultLimit(axis_mask,ilimit)

unsigned long axis_mask; bitwise axis mask

long far *ilimit; current ilimit

For related commands see:

DMRGetCurrentFaultLimit

DMRSetCurrentLimit

DMRGetCurrentLimit

DMRSetCurrentFaultTime

DMRGetCurrentFaultTime

7.25. DMRGetCurrentFaultLimit

The *DMRGetCurrentFaultLimit* function returns the current fault limit for the specified axes.

The *DMRGetCurrentFaultLimit* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetCurrentFaultLimit(axis_mask,ilimit)

unsigned long axis_mask;	bitwise axis mask
long far *ilimit;	current ilimit

For related commands see:

DMRSetCurrentFaultLimit

DMRSetCurrentLimit

DMRGetCurrentLimit

DMRSetCurrentFaultTime

DMRGetCurrentFaultTime

7.26. DMRSetCurrentFaultTime

The *DMRSetCurrentFaultTime* function defines the time over which instantaneous current is averaged. This function along with the function DMRSetCurrentFaultLimit is used to detect over current operation.

Default: [1000]

Range: [10 - 4000] in ten millisecond increments

Units: milliseconds

The *DMRSetCurrentFaultTime* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetCurrentFaultTime(axis_mask,time)

unsigned long axis_mask; bitwise axis mask

long time; current time

or

unsigned far Pascal DMRSetMCurrentFaultTime(axis_mask,time)

unsigned long axis_mask; bitwise axis mask

long far *time; current time

For related commands see:

DMRGetCurrenFaultTime

DMRSetCurrentFaultLimit

DMRGetCurrentFaultLimit

DMRSetCurrentTime

DMRGetCurrentTime

7.27. DMRGetCurrentFaultTime

The *DMRGetCurrentFaultTime* function returns the time used to determine over current operation.

The *DMRGetCurrentFaultTime* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetCurrentFaultTime(axis_mask,time)

unsigned long axis_mask;	bitwise axis mask
long far *time;	time used in over current detect

For related commands see:

DMRSetCurrentFaultTime

DMRSetCurrentFaultLimit

DMRGetCurrentFaultLimit

DMRSetCurrentTime

DMRGetCurrentTime

7.28. DMRGetObservedCurrent

The *DMRGetObservedCurrent* function returns the instantaneous current.

Range: [0 - 32767]

Units: 10 volts / 32767 counts

The *DMRGetObservedCurrent* function returns (0) if successful and a motion error code if not.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetObservedCurrent(axis_mask,current)

unsigned long axis_mask; bitwise axis mask

long far *current; observed current in percent

For related commands see:

DMRSetCurrentFaultTime

DMRSetCurrentFaultLimit

DMRGetCurrentFaultLimit

DMRSetCurrentTime

DMRGetCurrentTime

7.29. DMRGetObservedAvgCurrent

The *DMRGetObservedAvgCurrent* function returns the average current in counts. The time over which the current is averaged is determined by the DMRSetCurrentFaultTime command.

Range: [0-32767]

Units: 10 volts / 32767 counts

The *DMRGetObservedAvgCurrent* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetObservedAvgCurrent(axis_mask,current)

unsigned long axis_mask; bitwise axis mask

long far *current; observed current in percent

For related commands see:

DMRSetCurrentFaultTime

DMRSetCurrentFaultLimit

DMRGetCurrentFaultLimit

DMRSetCurrentTime

DMRGetCurrentTime

7.30. DMRGetObservedVelocity

The *DMRGetObservedVelocity* function returns the instantaneous velocity in counts per second. Note that this value is observed velocity sampled every 250 us. It will be very granular.

The *DMRGetObservedVelocity* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetObservedVelocity(axis_mask,velocity)

unsigned long axis_mask; bitwise axis mask

long far *velocity; observed velocity

For related commands see:

DMRSetCurrentFaultTime

DMRSetCurrentFaultLimit

DMRGetCurrentFaultLimit

DMRSetCurrentTime

DMRGetCurrentTime

7.31. DMRGetAvgVelocity

The *DMRGetAvgVelocity* function returns the velocity averaged over the time period specified by the *DMRSetAvgVelocityTime* command. The units will be in counts per second.

The *DMRGetAvgVelocity* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetAvgVelocity(axis_mask,velocity)

unsigned long axis_mask;	bitwise axis mask
long far *velocity;	observed velocity

For related commands see:

DMRSetCurrentFaultTime

DMRSetCurrentFaultLimit

DMRGetCurrentFaultLimit

DMRSetCurrentTime

DMRGetCurrentTime

7.32. DMRSetAvgVelocityTime

The *DMRSetAvgVelocityTime* function sets the time over which the velocity is average for the *DMRGetAvgVelocity* command.

Default: [1000]

Range: [10-1000] in ten millisecond increments

Units: milliseconds

The *DMRSetAvgVelocityTime* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetAvgVelocityTime(axis_mask,time)

unsigned long axis_mask; bitwise axis mask

long time; averaging time

or

unsigned far Pascal DMRSetMAvgVelocityTime(axis_mask,time)

unsigned long axis_mask; bitwise axis mask

long far *time; averaging time

For related commands see:

DMRSetCurrentFaultTime

DMRSetCurrentFaultLimit

DMRGetCurrentFaultLimit

DMRSetCurrentTime

DMRGetCurrentTime

7.33. DMRGetAvgVelocityTime

The *DMRGetAvgVelocityTime* function returns the time over which the velocity is average for the *DMRGetAvgVelocity* command.

The *DMRGetAvgVelocityTime* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetAvgVelocityTime(axis_mask,time)

unsigned long axis_mask;	bitwise axis mask
long far *time;	averaging time

For related commands see:

DMRSetCurrentFaultTime

DMRSetCurrentFaultLimit

DMRGetCurrentFaultLimit

DMRSetCurrentTime

DMRGetCurrentTime

7.34. DMRSetCwEot

The *DMRSetCwEot* function defines the clockwise end of travel limit. The axis card will not generate a position that exceeds this value. A fault will be generated if the user attempts to command a position beyond this value. This fault will be suppressed if the CW_FAULT bit in the fault mask is disabled or cleared.

Default: [2^32 - 1]

Range: []

Units: Machine Steps

The *DMRSetCwEot* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetCwEot(axis_mask,eot)
unsigned long axis_mask; bitwise axis mask
long eot; end of travel limit

or

unsigned far Pascal DMRSetMCwEot(axis_mask,eot)
unsigned long axis_mask; bitwise axis mask
long far *eot; end of travel limit

For related commands see:

DMRGetCwEot

DMRSetCcwEot

DMRGetCcwEot

7.35. DMRGetCwEot

The *DMRGetCwEot* function returns the clockwise end of travel limit.

The *DMRGetCwEot* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetCwEot(axis_mask,eot)

unsigned long axis_mask; bitwise axis mask

long far *eot; end of travel limit

For related commands see:

DMRGetCwEot

DMRSetCcwEot

DMRGetCcwEot

7.36. DMRSetCcwEot

The *DMRSetCcwEot* function defines the counter clockwise end of travel limit. It operates the same as the clockwise limit.

Default: [2^32 - 1]

Range: []

Units: Machine Steps

The *DMRSetCcwEot* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetCcwEot(axis_mask,eot)
unsigned long axis_mask; bitwise axis mask
long eot; end of travel limit

or

unsigned far Pascal DMRSetMCcwEot(axis_mask,eot)
unsigned long axis_mask; bitwise axis mask
long far *eot; end of travel limit

For related commands see:

DMRGetCcwEot

DMRSetCwEot

DMRGetCwEot

7.37. DMRGetCcwEot

The *DMRGetCcwEot* function returns the counter clockwise end of travel limit.

The *DMRGetCcwEot* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetCcwLimit(axis_mask,eot)

unsigned long axis_mask;	bitwise axis mask
long far *eot;	end of travel limit

For related commands see:

DMRGetCcwEot

DMRSetCwEot

DMRGetCwEot

7.38. DMRSetPositionErrorLimit

The *DMRSetPositionLimit* function defines the position following error limit. Should this limit be exceeded, a fault will be generated.

Default: [65535]

Range: [0 - 10,000,000]

Units: Machine Steps

The *DMRSetPositionErrorLimit* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetPositionErrorLimit(axis_mask,limit)

unsigned long axis_mask; bitwise axis mask

long limit; following error limit

or

unsigned far Pascal DMRSetMPositionErrorLimit(axis_mask,limit)

unsigned long axis_mask; bitwise axis mask

long far *limit; following error limit

For related commands see:

DMRGetPositionErrorLimit

7.39. DMRGetPositionErrorLimit

The *DMRGetPositionErrorLimit* function returns the position following error fault limit.

The *DMRGetPositionErrorLimit* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetPositionErrorLimit(axis_mask,limit);

unsigned long axis_mask;	bitwise axis mask
long far *limit;	following error limit

For related commands see:

DMRSetPositionErrorLimit

7.40. DMRSetInPositionBand

The *DMRSetInPositionBand* function defines the in-position band. If the observed position plus or minus the position command is within the in_position band, and the axis is done, the axis' in-position status bit becomes active.

Default: [65535]

Range: [1 - 10,000,000]

Units: Machine Steps

The *DMRSetInPositionBand* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetInPositionBand(axis_mask,band)

unsigned long axis_mask; bitwise axis mask

long band; in-position band

or

unsigned far Pascal DMRSetMInPositionBand(axis_mask,band)

unsigned long axis_mask; bitwise axis mask

long far *band; in-position band

For related commands see:

DMRGetInPositionBand

DMRGetAxisStat

7.41. DMRGetInPositionBand

The *DMRGetInPositionBand* function returns the in-position band.

The *DMRGetInPositionBand* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetInPositionBand(axis_mask,band)

unsigned long axis_mask; bitwise axis mask

long far *band; in_position band

For related commands see:

DMRSetInPositionBand

DMRGetAxisStat

7.42. DMRGetPositionCommand

The *DMRGetPositionCommand* function returns the instantaneous position command of the specified axis.

The *DMRGetPositionCommand* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetPositionCommand(axis_mask,pcmd)

unsigned long axis_mask;	bitwise axis mask
long far *pcmd;	position command

7.43. DMRGetTarget

The *DMRGetTarget* function returns the target position of specified axis. This target position is the current destination of the axis.

The *DMRGetTarget* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetTarget(axis_mask,target)

unsigned long axis_mask;	bitwise axis mask
long far *target;	target position

7.44. DMRGetMoveRemaining

The *DMRGetMoveRemaining* function returns the amount of distance left in the position command generator for the previous motion command.

The *DMRGetMoveRemaining* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetMoveRemaining(axis_mask,remaining)

unsigned long axis_mask; bitwise axis mask

long far *remaining; amount of move remaining

7.45. DMRSetPosition

The *DMRSetPosition* function sets the desired position of the specified axis.

The *DMRSetPosition* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetPosition(axis_mask,position)
unsigned long axis_mask; bitwise axis mask
long position; desired position

or

unsigned far Pascal DMRSetMPosition(axis_mask,position)
unsigned long axis_mask; bitwise axis mask
long far *position; desired position

7.46. DMRGetPosition

The *DMRGetPosition* function returns the observed position of the specified axis.

The *DMRGetPosition* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetPosition(axis_mask,position)

unsigned long axis_mask;	bitwise axis mask
long far *position;	observed position

7.47. DMRGetResolver

The *DMRGetResolver* function returns the observed angular position of the specified axis. It will always return a value between 0 and 65535.

The *DMRGetResolver* function returns 0 if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetResolver(axis_mask,position)

unsigned long axis_mask; bitwise axis mask

long far *position; absolute position

7.48. DMRGetCamPosition

The *DMRGetCamPosition* function returns the current master position offset into the cam table.

The *DMRGetCamPosition* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetCamPosition(axis_mask,position)

unsigned long axis_mask; bitwise axis mask

int far *position; present cam table position

7.49. DMRGetCamPoint

The *DMRGetCamPoint* function returns the current master position offset into the cam table.

The *DMRGetCamPoint* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetCamPoint(axis_mask,position)

unsigned long axis_mask;	bitwise axis mask
int far *position;	present cam table position

7.50. DMRSetMasterPosition

The *DMRSetMasterPosition* function sets the observed master position of the master R/D associated with this axis. Note that an axis has both its own position and a master position when a DMRConfigMaster has been issued to the axis.

The *DMRSetMasterPosition* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetMasterPosition(axis_mask,position)

unsigned long axis_mask; bitwise axis mask

long master_position; observed master position

or

unsigned far Pascal DMRSetMMasterPosition(axis_mask,position)

unsigned long axis_mask; bitwise axis mask

long far *position; observed master position

For related commands see:

DMRGetMasterResolverPosition

7.51. DMRGetMasterPosition

The *DMRGetMasterPosition* function returns the observed master position of the master R/D associated with this axis. This command is typically used for synchronized tracking applications. Note that an axis has both its own position and a master position when a *DMRConfigMaster* has been issued to this axis.

The *DMRGetMasterPosition* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetMasterPosition(axis_mask,position)

unsigned long axis_mask; bitwise axis mask

long far *master_position; observed master position

For related commands see:

DMRGetMasterResolverPosition

7.52. DMRSetCamOffset

The *DMRSetCamOffset* function applies an offset to the master position without modifying the master position. This allows the user to advance or retard the master portion of the cam table.

The *DMRSetCamOffset* function returns 0 if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetCamOffset(axis_mask,position)

unsigned long axis_mask; bitwise axis mask

int far *position; present cam table position

or

unsigned far Pascal DMRSetMCamOffset(axis_mask,position)

unsigned long axis_mask; bitwise axis mask

long far *position; present cam table position

7.53. DMRGetCamOffset

The *DMRGetCamOffset* function returns the current master offset applied to the master position.

The *DMRGetCamOffset* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetCamOffset(axis_mask,position)

unsigned long axis_mask;	bitwise axis mask
int far *position;	present cam table position

7.54. DMRGetMasterResolver

The *DMRGetMasterResolver* function returns the observed angular master position of the specified axis.

The *DMRGetMasterResolver* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetMasterResolver(axis_mask,position)

unsigned long axis_mask;	bitwise axis mask
long far *position;	absolute position

7.55. DMRGetMasterAbsolute

The *DMRGetMasterAbsolute* function returns the current offset of the master axis in relation to the value set by the *DMRSetMasterLength* command.

The *DMGetMasterAbsolute* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetMasterAbsolute(axis_mask,position)

unsigned long axis_mask;	bitwise axis mask
long master_position;	observed master position

For related commands see:

DMRGetMasterResolverPosition

7.56. DMRSetMasterLength

The *DMRSetMasterLength* function sets an arbitrary specified length on the master axis. It does not necessarily have to match the length of the master portion of a cam table for this slave axis. It provides a means of implementing a user specified rollover of master position.

The *DMRSetMasterLength* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetMasterLength(axis_mask,position)

unsigned long axis_mask; bitwise axis mask

long master_position; observed master position

or

unsigned far Pascal DMRSetMMasterLength(axis_mask,position)

unsigned long axis_mask; bitwise axis mask

long far *position; observed master position

For related commands see:

DMRGetMasterResolverPosition

7.57. DMRGetMasterLength

The *DMRGetMasterLength* function returns the length value set by the *DMRSetMasterLength* command. If the length received here matches the actual length of the master portion of the associated cam table, and the slave axis is in sync, then the *DMRGetMasterAbsolute* and the *DMRGetMasterOffset* commands will return the same values. The *DMRGetMasterAbsolute* is intended to be used with the *DMRSetMasterLength* and *DMRGetMasterLength* commands.

The *DMGetMasterLength* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetMasterLength(axis_mask,position)

unsigned long axis_mask;	bitwise axis mask
long master_position;	observed master position

For related commands see:

DMRGetMasterResolverPosition

7.58. DMRSetSyncSpeed

This function allows the user to set the sync speed. This speed will be used whenever the axis is first put into sync mode with a mode of 2. When sync with a mode of 2 is called, the slave axis assumes that its present position is the start of the cam table. It then reads the master position and determines what offset it must move to match its position with that of the master in the table. It will then use the speed defined with this command to index the appropriate amount.

The *DMRSetSyncSpeed* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetSyncSpeed(axis_mask,speed)

unsigned long axis_mask; bitwise axis mask

long speed; value

or

unsigned far Pascal DMRSetMSyncSpeed(axis_mask,speed)

unsigned long axis_mask; bitwise axis mask

long far *speed; value

7.59. DMRGetSyncSpeed

This function returns the sync speed previously set with the *DMRSetSyncSpeed* command.

The *DMRGetSyncSpeed* function returns 0 if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetSyncSpeed(axis_mask,speed)

unsigned long axis_mask; bitwise axis mask

long far *speed; value

7.60. DMRSetBasespeed

This function sets the basespeed of the motor. Phase advance will begin when the motor speed goes above this value.

The *DMRSetBasespeed* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetBasespeed(axis_mask,basespeed)

unsigned long axis_mask; bitwise axis mask

long basespeed; motor basespeed

or

unsigned far Pascal DMRSetMBasespeed(axis_mask,basespeed)

unsigned long axis_mask; bitwise axis mask

long far *basespeed; motor basespeed

7.61. DMRGetBasespeed

This function returns the basespeed of the motor.

The *DMRSetBasespeed* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetBasespeed(axis_mask,basespeed)

unsigned long axis_mask;	bitwise axis mask
long far *basespeed;	motor basespeed

7.62. DMRSetPhaseAdvance

This function sets the maximum phase advance of the motor.

The *DMRSetPhaseAdvance* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetPhaseAdvance(axis_mask,phase)
unsigned long axis_mask; bitwise axis mask
long phase; phase advance

or

unsigned far Pascal DMRSetMPhaseAdvance(axis_mask,phase)
unsigned long axis_mask; bitwise axis mask
long far *phase; phase advance

7.63. DMRGetPhaseAdvance

This function returns the maximum phase advance of the motor.

The *DMRGetPhaseAdvance* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetPhaseAdvance(axis_mask,phase)

unsigned long axis_mask;	bitwise axis mask
long far *phase;	phase advance

7.64. DMRSetIOLevel

This function specifies the on or off state of each of the inputs and outputs into the axis card. The *DMRGetAxisStat* command description contains the defines for each bit in the status word. The first six bits of the word are used by the *DMRSetIOLevel* command. A one in the bit means the outputs and inputs are active high, a zero means they are active low. The bit mask list found in DMRGetAxisStat for the first 6 bits is valid here.

The *DMRSetIOLevel* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetIOLevel(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long fault; fault mask

or

unsigned far Pascal DMRSetMIOLevel(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long far *fault; fault mask

7.65. DMRGetIOLevel

This function returns the setting of each of the inputs and outputs into the axis card. The *DMRGetAxisStat* command description contains the defines for each bit in the status word. The first six bits of the word are used by the *DMRSetIOLevel* command. A one in the bit means the outputs and inputs are active high, a zero means they are active low. The bit mask list found in *DMRGetAxisStat* for the first 6 bits is valid here.

The *DMRGetIOLevel* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetIOLevel(axis_mask,fault)

unsigned long axis_mask;	bitwise axis mask
long fault;	fault mask

7.66. DMRSetAuxOffset

This function sets an offset which is applied to the master position of the auxiliary output table associated with this axis. The point at which the table begins and ends may be advanced or retarded by setting this value.

The *DMRSetAuxOffset* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetAuxOffset(axis_mask,offset)

unsigned long axis_mask; bitwise axis mask

long offset; amount to offset the table

or

unsigned far Pascal DMRSetAuxOffset(axis_mask,offset)

unsigned long axis_mask; bitwise axis mask

long offset; amount to offset the table

7.67. DMRGetAuxOffset

This function returns the amount of offset that is currently being applied to the master position of the auxiliary output table associated with this axis.

The *DMRGetAuxOffset* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetAuxOffset(axis_mask,offset)

unsigned long axis_mask;	bitwise axis mask
long far *offset;	amount of offset to the table

7.68. DMRSetFaultMask

This function determines which faults are active. A one in a bit sets the corresponding fault active, a zero means that a fault is ignored. Refer to the *DMRGetFault* command for fault mask bit definitions.

Once an active fault occurs, the axis will do one or more of the following:

- disable axis
- halt
- interrupt
- switch off the aux output

Other fault masks determine which of these events will occur.

The *DMRSetFaultMask* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetFaultMask(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long fault; fault mask

or

unsigned far Pascal DMRSetMFaultMask(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long far *fault; fault mask

For related commands see:

DMRSetAuxMAsk

DMRSetInterruptMask

DMRSetDisableMask

DMRSetHaltMask, DMRGetFaultMask

7.69. DMRGetFaultMask

This function returns which faults are fatal and which are not fatal.

Refer to the DMRGetFault command for fault mask bit definitions.

The *DMRGetFaultMask* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetFaultMask(axis_mask,fault)

unsigned long axis_mask;	bitwise axis mask
long far *fault;	fault mask

7.70. DMRSetFault

This function clears faults on the specified axes if the fault condition no longer exists. A one in the mask clears that specific fault while a zero does nothing.

Refer to the *DMRGetFault* command for fault mask bit definitions.

The *DMRSetFault* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetFault(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long fault; fault mask

or

unsigned far Pascal DMRSetMFault(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long far *fault; fault mask

7.71. DMRGetFault

This function returns the status of the axis card faults on the axes specified in the axis mask. This does not clear the fault. The *DMRSetFault* command must be used to clear any faults.

Axis fault bits are defined as follows:

Bit#	Fault Definition
0x00000001	position error
0x00000002	average current limit error
0x00000004	CW hard limit
0x00000008	CCW hard limit
0x00000010	CW soft limit
0x00000020	CCW soft limit
0x00000040	drive fault
0x00000080	feedback failure
0x00000100	programming error
0x00000200	master feedback failure
0x00000400	homing fault
0x00000800	user trigger by DMRSetUserFault
0x00001000	velocity trap
0x00002000	velocity command trap
0x00004000	home switch tolerance
0x00008000	probe input
(remaining bits are undefined)	

The *DMRGetFault* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetFault(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long far *fault; fault mask

7.72. DMRSetDisableMask

This function determines which faults disable the axes. A one in a bit sets the corresponding fault active, a zero means the fault does not disable the axis.

In order for this event to take place, the corresponding bit in the fault mask must be set. *DMRSetFaultMask* performs this function. Refer to the DMRGetFault command for fault mask bit definitions.

The *DMRSetDisableMask* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetDisableMask(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long fault; fault mask

or

unsigned far Pascal DMRSetMDisableMask(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long far *fault; fault mask

7.73. DMRGetDisableMask

This function returns which faults cause the axis to disable.

Refer to the *DMRGetFault* command for fault mask bit definitions.

The *DMRGetDisableMask* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetDisableMask(axis_mask,fault)

unsigned long axis_mask;	bitwise axis mask
long far *fault;	fault mask

7.74. DMRSetInterruptMask

This function determines which faults will cause a system interrupt. A one in a bit sets the corresponding fault active, a zero means the fault will not cause an interrupt. In order for this event to take place, the corresponding bit in the fault mask must be set. *DMRSetFaultMask* performs this function. Refer to the *DMRGetFault* command for fault mask bit definitions.

The *DMRSetInterruptMask* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetInterruptMask(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long fault; fault mask

or

unsigned far Pascal DMRSetMInterruptMask(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long far *fault; fault mask

7.75. DMRGetInterruptMask

This function returns which faults cause system interrupts.

Refer to the *DMRGetFault* command for fault mask bit definitions.

The *DMRGetInterruptMask* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetInterruptMask(axis_mask,fault)

unsigned long axis_mask;	bitwise axis mask
long far *fault;	fault mask

7.76. DMRSetAuxMask

This function determines which faults will turn off the auxiliary output. A one in a bit sets the corresponding fault active, a zero means that the auxiliary output will not be turned off. In order for this event to take place, the corresponding bit in the fault mask must be set. *DMRSetFaultMask* performs this function.

Refer to the *DMRGetFault* command for fault mask bit definitions.

The *DMRSetAuxMask* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetAuxMask(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long fault; fault mask

or

unsigned far Pascal DMRSetMAuxMask(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long far *fault; fault mask

7.77. DMRGetAuxMask

This function returns which faults cause the auxiliary output to come on.

Refer to the *DMRGetFault* command for fault mask bit definitions.

The *DMRGetAuxMask* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetAuxMask(axis_mask,fault)

unsigned long axis_mask;	bitwise axis mask
long far *fault;	fault mask

7.78. DMRSetHaltMask

This function determines which faults will cause the axis to halt. A one in a bit sets the corresponding fault active, a zero means that the axis will not be halted. In order for this event to take place, the corresponding bit in the fault mask must be set. *DMRSetFaultMask* performs this function.

Refer to the *DMRGetFault* command for fault mask bit definitions.

The *DMRSetHaltMask* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetHaltMask(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long fault; fault mask

or

unsigned far Pascal DMRSetMHaltMask(axis_mask,fault)

unsigned long axis_mask; bitwise axis mask

long far *fault; fault mask

7.79. DMRGetHaltMask

This function returns which faults cause the axes to halt.

Refer to the *DMRGetFault* command for fault mask bit definitions.

The *DMRGetHaltMask* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetHaltMask(axis_mask,fault)

unsigned long axis_mask;	bitwise axis mask
long far *fault;	fault mask

7.80. DMRSetEcho

This function allows the user to set a 'dummy' parameter. It has no real use other than as a holding area.

The *DMRSetEcho* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetEcho(axis_mask,value)

unsigned long axis_mask; bitwise axis mask

long value; value

or

unsigned far Pascal DMRSetMEcho(axis_mask,value)

unsigned long axis_mask; bitwise axis mask

long far *value; value

7.81. DMRGetEcho

This function allows the user to retrieve a 'dummy' parameter. It has no real use other than as a holding area.

The *DMRGetEcho* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetEcho(axis_mask,value)

unsigned long axis_mask;	bitwise axis mask
long value;	value

7.82. DMRSetClock

This function allows the user to set a one millisecond clock to any value. It will continue counting from the specified time. There is one clock for every axis.

The *DMRSetClock* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetClock(axis_mask,time)

unsigned long axis_mask; bitwise axis mask

long time; value

or

unsigned far Pascal DMRSetMclock(axis_mask,time)

unsigned long axis_mask; bitwise axis mask

long far *time; value

7.83. DMRGetClock

This function allows the user to get the clock value for a specified axis.

The *DMRGetClock* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetClock(axis_mask,time)

unsigned long axis_mask;	bitwise axis mask
long far *time;	value

7.84. DMRSetDrive

This function allows the user to enable and disable the drive for the specified axis. A value of zero disables, and any other value disables the axis.

The *DMRSetDrive* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetDrive(axis_mask,switch)
unsigned long axis_mask; bitwise axis mask
long switch; value

or

unsigned far Pascal DMRSetMDrive(axis_mask,switch)
unsigned long axis_mask; bitwise axis mask
long far *switch; value

7.85. DMRGetDrive

This function allows the user to determine whether the drive is enabled or disabled. A value of zero means that it is disabled, and any other value means that it is enabled.

The *DMRGetDrive* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetDrive(axis_mask,switch)

unsigned long axis_mask;	bitwise axis mask
long far *switch;	value

7.86. DMRSetAux

This function allows the user to enable and disable the auxiliary output of the specified axis. This may be used for things like a motor brake. It can be used in conjunction with the *DMRSetAuxMask* command to allow the output to change state on an axis fault.

The *DMRSetAux* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetAux(axis_mask,switch)

unsigned long axis_mask; bitwise axis mask

long switch; value

or

unsigned far Pascal DMRSetMAux(axis_mask,switch)

unsigned long axis_mask; bitwise axis mask

long far *switch; value

7.87. DMRGetAux

This function allows the user to find the state of the auxiliary output for the specified axis.

The *DMRGetAux* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetAux(axis_mask,switch)

unsigned long axis_mask;	bitwise axis mask
long far *switch;	value

7.88. DMRGetMoveQueueDepth

This function returns the number of queued moves present in the specified axis' queue.

The *DMRGetMoveQueueDepth* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetMoveQueueDepth(axis_mask,depth)

unsigned long axis_mask;	bitwise axis mask
long far *depth;	number of queued moves

7.89. DMRGetMoveQueueSize

This function returns the total size of the queue for each axis.

The *DMRGetMoveQueueSize* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetMoveQueueSize(axis_mask,size)

unsigned long axis_mask;	bitwise axis mask
long far *size;	size of queue

7.90. DMRGetHomeSwitchPos

This function returns the resolver position at which the axis sees the home switch during a homing procedure.

The *DMRGetHomeSwitchPos* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRPARM

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetHomeSwitchPos(axis_mask,pos)

unsigned long axis_mask; bitwise axis mask

long far *pos; position of homeswitch

∇ ∇ ∇

CHAPTER 8: AUXILIARY I/O FUNCTIONS

8.1. DMRGetAuxTables

This function returns the number of AUXILIARY output tables that are available. The total axis card RAM available is determined by calling *DMRGetFreeMemory*.

The *DMRGetAuxTables* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRAUX

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetAuxTables(cpu,tables)

int cpu;	axis processor number
int far *tables;	number of aux tables

8.2. DMRAllocateAuxTable

This function allocates storage on the axis processor for user AUXILIARY output tables. Multiple aux tables are allowed on the axis processor. The number of aux tables available depends on the amount of storage on the axis processor and the size of each aux table. The total number of aux table points is determined by the *DMRGetAuxTables* function.

Aux tables are not allocated to any specific axis. Aux tables are associated with axes by issuing the *DMRSetAuxMode* command. Note that each point in the aux table uses 6 bytes of axis memory. The aux table starts at point number zero and ends with point number 'size' minus one. These tables allow the AUXILIARY output to be turned on and off with respect to a table of positions. This provides the user with very tightly coupled motion and I/O capability. The master associated with a slave axis will be the same for motion as for AUXILIARY output functions.

The *DMRAllocateAuxTable* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRAUX

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRAllocateAuxTable(cpu,table_number,size)

int cpu;	axis processor number
int table_number;	aux table number
int size;	table size in points

8.3. DMRFreeAuxTable

This function deallocates the AUXILIARY output table from axis processor memory.

The *DMRFreeAuxTable* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRAUX

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRFreeAuxTable(cpu,table)

int cpu; axis processor number

int tables; aux table number

8.4. DMRGetAuxTableStatus

The *DMRGetAuxTableStatus* function returns the number of points in this aux table and the status of that table. The status will be zero if the table is not allocated, and one if it has been allocated.

The *DMRGetAuxTableStatus* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRAUX

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetAuxTableStatus(cpu,table,points,status)

int cpu;	axis processor number
int table;	table number
int far *points;	number of points in table
long far *status;	status of table

8.5. DMRWriteAuxPoint

The *DMRWriteAuxPoint* function writes one point into the specified AUXILIARY output table. Master position points are in units of machine steps. The output value will be either zero or one signifying an on or off state of the AUXILIARY output. The voltage level associated with the on and off state of any given AUXILIARY output is determined by the jumper settings on the D/A card and the setting of the I/O level by the command *DMRSetIOLevel*. Master points in the cam table must be non-repeating and increasing. Points are entered starting at position zero and ending at point 'size' minus one where 'size' is the number of points allocated.

The *DMRWriteAuxPoint* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRAUX

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRWriteAuxPoint(cpu,table,point,master,value)

int cpu;	axis processor number
int table;	aux table number
int point;	point number
long master_pos;	master position
int value;	AUXILIARY output value

8.6. DMRWriteMultAuxPoints

The *DMRWriteMultAuxPoints* function writes multiple points into the specified AUXILIARY output table. Master position points are in units of machine steps. The output value will be either zero or one signifying an on or off state of the AUXILIARY output. The voltage level associated with the on and off state of any given AUXILIARY output is determined by the jumper settings on the D/A card and the setting of the I/O level by the command *DMRSetIOLevel*. The variable num contains the number of points in the array. The variable ptr points to an array of type *DMRAuxStruct* where each entry contains a master position and a level. Master points in the cam table must be non-repeating and increasing. Points are entered starting at position zero and ending at point 'size' minus one where 'size' is the number of points allocated.

The *DMRAuxStruct* is as follows:

```
typedef struct auxstruct {
        long Master;
        int  Level;
        } DMRAuxStruct;
```

The *DMRWriteMultAuxPoints* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRAUX

or

#define INCL_DMRALL

#include "dmrhead.h"

```
unsigned far Pascal DMRWriteMultAuxPoint(cpu,table,point,num,ptr)
int cpu;                    axis processor number
int table;                  aux table number
int point;                  starting point number
int num;                    number of points to read
DMRAuxStruct far *ptr;      pointer to table containing points
```

8.7. DMRReadAuxPoint

The *DMRReadAuxPoint* function reads one point from the specified AUXILIARY output table. Master position points are in units of machine steps. The output value will be either zero or one signifying an on or off state of the AUXILIARY output. The voltage level associated with the on and off state of any given AUXILIARY output is determined by the jumper settings on the D/A card and the IO level set by the command *DMRSetIOLevel*. Points are entered starting at point zero and ending at point 'size' minus one where 'size' is the number of points allocated.

The *DMRReadAuxPoint* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRAUX

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRReadAuxPoint(cpu,table,point,master,value)

int cpu;	axis processor number
int table;	aux table number
int point;	point number
long far *master_pos;	master position
int far *value;	AUXILIARY output value

8.8. DMRReadMultAuxPoints

The *DMRReadMultAuxPoints* function reads multiple points from the specified AUXILIARY output table. Master position points are in units of machine steps. The output value will be either zero or one signifying an on or off state of the AUXILIARY output. The voltage level associated with the on and off state of any given AUXILIARY output is determined by the jumper settings on the D/A card and the IO level set by the command *DMRSetIOLevel*. The variable num contains the number of points to read from the table and put into the array. The variable ptr points to the array where the points should be placed. Points are entered starting at point zero and ending at point 'size' minus one where 'size' is the number of points allocated.

The *DMRAuxStruct* is as follows:

```
typedef struct auxstruct {
        long Master;
        int  Level;
        } DMRAuxStruct;
```

The *DMRReadMultAuxPoints* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRAUX

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRReadMultAuxPoints(cpu,table,point,num,ptr)

int cpu;	axis processor number
int table;	aux table number
int point;	starting point number
int num;	number of points to read
DMRAuxStruct far *ptr;	AUXILIARY output value

8.9. DMRSetAuxMode

The *DMRSetAuxMode* function associates an output table with an axis and specifies the mode to put it in.

The mode values will be :

0 - off or out of sync

1 - output mode on

A programming error will be returned if the associated axis is not enabled when sync is enabled.

The *DMRSetAuxMode* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRAUX

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRSetAuxMode(axis,table,mode)

unsigned long axis; axis number

int table; table number

int mode; mode of the table

8.10. DMRGetAuxMode

The *DMRGetAuxMode* function returns the table number associated with this axis and the mode that the table is in.

The mode values will be :

0 - off or out of sync

1 - output mode on

The *DMRGetAuxMode* function returns (0) if completed and a motion error code if not successful.

SYNTAX:

#define INCL_DMRAUX

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRGetAuxMode(axis,table,mode)

unsigned long axis;	axis number
int far *table;	table number
int far *mode;	mode of the table

∇ ∇ ∇

CHAPTER 9: VMIC DISCRETE I/O BOARD FUNCTIONS

9.1. DMRVMICWriteDiscrete

This function sets the specified I/O channel with the value passed. If value is zero, the I/O point will be cleared. If the value is non-zero, the I/O point will be set.

If there is one VMIC I/O card present, valid I/O numbers range from 0 to 127. If there are two cards present, then valid I/O point numbers are between 0 and 255. Please refer to the VMIC Instruction Manual for further information on how to designate I/O points as inputs or outputs.

The *DMRVMICWriteDiscrete* function returns (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRVMIC

or

#define INCL_DMRALL

#include "dmrhead.h"

or

#include "dmrvmic.h"

unsigned far Pascal DMRVMICWriteDiscrete(channel,value)

int channel; I/O point number to set or clear

int value; value to set the I/O point

For related commands see:

DMRVMICReadDiscrete

9.2. DMRVMICReadDiscrete

This function reads the state of one of the VMIC I/O points. The channel number specifies which I/O point to interogate. The returned value is placed at the address pointed to by value. The I/O point is cleared if the returned value is zero, and it is set if the returned state is non-zero.

If there is one VMIC I/O card present, valid I/O numbers range from 0 to 127. If there are two cards present, then valid I/O point numbers are between 0 and 255. Please refer to the VMIC Instruction Manual for further information on how to designate I/O points as inputs or outputs.

The *DMRVMICReadDiscrete* function returns (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRVMIC

or

#define INCL_DMRALL

#include "dmrhead.h"

or

#include "dmrvmic.h"

unsigned far Pascal DMRVMICReadDiscrete(channel,value)

int channel; I/O point number

int far *value; pointer to returned value

For related commands see:

DMRVMICWriteDiscrete

∇ ∇ ∇

CHAPTER 10: XYCOM I/O FUNCTIONS

10.1. DMRXycomWriteDiscrete

This function sets the specified I/O channel with the value passed. If value is zero, the output point will be cleared. If the value is non-zero, the output point will be set. Valid outputs range from 16 to 31.

The *DMRXycomWriteDiscrete* function returns (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRVMIC

or

#define INCL_DMRALL

#include "dmrhead.h"

or

#include "dmrxycom.h"

unsigned far Pascal DMRXycomWriteDiscrete(channel,value)

int channel; I/O point number to set or clear

int value; value to set the I/O point

For related commands see:

DMRXycomReadDiscrete

10.2. DMRXycomReadDiscrete

This function reads the state of one of the XYCOM I/O points. The channel number specifies which input point to interrogate. The returned value is placed at the address pointed to by value. The input point is cleared if the returned value is zero, and it is set if the returned state is non-zero. Valid input range from 0 to 15.

The *DMRXycomReadDiscrete* function returns (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRVMIC

or

#define INCL_DMRALL

#include "dmrhead.h"

or

#include "dmrxycom.h"

unsigned far Pascal DMRXycomReadDiscrete(channel,value)

int channel;	I/O point number
int far *value;	pointer to returned value

For related commands see:

DMRXycomWriteDiscrete

∇ ∇ ∇

CHAPTER 11: ANALOG I/O FUNCTIONS

11.1. DMRMatrixInitialize

This function causes the axis processor to initialize the Matrix Analog Input Board. This must be executed before any other MATRIX commands may be successfully executed. An auto calibration of the analog inputs is performed when a *DMRMatrixInitialize* command is performed.

The *DMRMatrixInitialize* function returns (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRMATRIX

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRMatrixInitialize(void)

For related commands see:

DMRMatrixReadAnalog

11.2. DMRMatrixReadAnalog

This function reads the specified analog input I/O channel on analog input card. The voltage reading is placed into the variable value. The I/O channel numbers range from zero to seven. Refer to MS-AD12-xx MSX-AD12-xx user's manual for further information on Input Voltage Range Selection.

The *DMRMatrixReadAnalog* function returns (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRMATRIX

or

#define INCL_DMRALL

#include "dmrhead.h"

unsigned far Pascal DMRMatrixReadAnalog(channel,value)

unsigned channel; I/O channel number

long far *value; address of where to put result

For related commands see:

DMRMatrixInitialize

∇ ∇ ∇

CHAPTER 12: ALLEN-BRADLEY VME/PLC5 FUNCTIONS

12.1. DMRABPLCOpen

This function opens the device driver that talks to the Allen-Bradley VME/PLC5 board (6008-LTV). Please refer to the instruction manual available from Allen-Bradley for this board for further information on its use. This command must be performed before any other commands may be sent to the PLC. The board must reside at an address in short I/O space of 4000H. If the associated calls are made and there are no I/O boards in the system, then BUS errors will occur. This will cause the system to crash.

The *DMRABPLCOpen* function returns (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRABPLC

or

#define INCL_DMRALL

#include "dmrhead.h"

or

#include "dmrabplc.h"

unsigned far Pascal DMRABPLCOpen(void)

12.2. DMRABPLCClose

This function closes the device driver associated with these commands. No PLC commands may take place after this until another *DMRABPLCOpen()* or *DMRABPLCCallDebug(2)* command is issued.

The *DMRABPLCClose* function returns (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRABPLC

or

#define INCL_DMRALL

#include "dmrhead.h"

or

#include "dmrabplc.h"

unsigned far Pascal DMRABPLCClose()

12.3. DMRABPLCCallDebug

This function enables and disables the debug mode operation of these commands. Full debug mode of operation allows the user to debug code on a system that does not have a VME/PLC5.

A flag value of zero disables debug and allows commands to be executed. A flag value of one instructs the library to return a success code for unimplemented commands. A value of 2 turns on debug completely. No commands will be passed on to the I/O card when this state has been enabled.

The *DMRABPLCCallDebug* function returns (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRABPLC

or

#define INCL_DMRALL

#include "dmrhead.h"

or

#include "dmrabplc.h"

unsigned far Pascal DMRABPLCCallDebug(flag)

int flag; debug flag

12.4. DMRABPLCGetStatus

The *DMRABPLCGetStatus* command is used to retrieve information regarding the AB PLC commands.

The *DMRABPLCGetStatus* function returns 0 if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRABPLC

or

#define INCL_DMRALL

#include "dmrhead.h"

or

#include "dmrabplc.h"

unsigned far Pascal DMRABPLCGetStatus(ptr)

char far *ptr;

12.5. DMRABPLCWriteBit

The *DMRABPLCWriteBit* command is used to modify an I/O signal of an Allen Bradley file or block. File specifies PLC program file or block name. Element is an integer ranging from 0 to 999 specifying entry into the file. Bit specifies the bit number to modify.

Value can be:

0 - off

1 - on

Status returned is :

0 - Bit set Okay

1 - Instruction failed

The *DMRABPLCWriteBit* function returns (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRABPLC

or

#define INCL_DMRALL

#include "dmrhead.h"

or

#include "dmrabplc.h"

unsigned far Pascal DMRABPLCWriteBit(file,element,bit,value,stat)

unsigned file;	Allen Bradley PLC filename
unsigned element;	Rung number
unsigned bit;	Bit number
unsigned value;	Bit value
unsigned far *stat;	status of instruction

12.6. DMRABPLCReadBit

The *DMRABPLCWriteBit* command is used to read an I/O signal of an Allen Bradley file or block. File specifies PLC program file or block name. Element is an integer ranging from 0 to 999 specifying entry into the file. Bit specifies the bit number to read.

Value can be:

0 - off

1 - on

Status returned is :

0 - Bit set Okay

1 - Instruction failed

The *DMRABPLCReadBit* function returns (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRABPLC

or

#define INCL_DMRALL

#include "dmrhead.h"

or

#include "dmrabplc.h"

unsigned far Pascal DMRABPLCReadBit(file,element,bit,data,stat)

unsigned file;	Allen Bradley PLC filename
unsigned element;	Rung number
unsigned bit;	Bit number
unsigned far *value;	Bit value
unsigned far *stat;	status of instruction

12.7. DMRABPLCecho

The *DMRABPLCecho* command is used to test the Allen Bradley board for communication failure. Len specifies the number of characters to send. Str_send specifies the character string to send. Str_return specifies the number of characters returned.

The *DMRABPLCecho* function returns (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRABPLC

or

#define INCL_DMRALL

#include "dmrhead.h"

or

#include "dmrabplc.h"

unsigned far Pascal DMRABPLCecho(len,str_send,str_return)

ushort len;	number of characters
unsigned char far *s;	pointer to string to send
unsigned char far *q;	pointer to string returned

12.8. DMRABPLCWrite

This function sets the specified I/O channel with the value passed. If value is zero, the I/O point will be cleared. If the value is non-zero, the I/O point will be set.

The *DMRABPLCWrite* function returns 0 if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRABPLC

or

#define INCL_DMRALL

#include "dmrhead.h"

or

#include "dmrabplc.h"

unsigned far Pascal DMRABPLCWrite(channel,value)

int channel;	I/O point number to set or clear
int value;	value to set the I/O point

12.9. DMRABPLCRead

This function reads the state of one of the VME/PLC5 I/O points. The channel number specifies which I/O point to interrogate. The returned value is placed at the address pointed to by value. The I/O point is cleared if the returned value is zero, and it is set if the returned state is non-zero.

The *DMRABPLC5* function returns (0) if completed and an error code if not successful.

SYNTAX:

#define INCL_DMRABPLC

or

#define INCL_DMRALL

#include "dmrhead.h"

or

#include "dmrabplc.h"

unsigned far Pascal DMRABPLCRead(channel,value)

int channel;	I/O point number
int far *value;	pointer to returned value

∇ ∇ ∇

CHAPTER 13: MODICON I/O FUNCTIONS

13.1. Modicon IO

Modicon supplies a library of commands for talking with the Modbus Plus communication card. This allows an application program to set IO registers and interrogate IO registers and status of the communication card.

DMR has modified the library so that they may be called from Presentation Manager applications. The open command has been modified, and two commands were added. The added commands allow the application to set and get a register in one command where it would take two (ncb_send, ncb_receive_wait) using the standard Modbus Plus library.

The Modbus Plus documentation should be refered to for further information on this interface.

The commands added or changed by DMR have been documented here.