ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *National Instruments PCI-Step-4OX* at our website: ***Click HERE***

Motion Control

ValueMotion™ Software Reference Manual

NATIONAL INSTRUMENTS™

May 1998 Edition
Part Number 321939A-01

Internet Support
E-mail: support@natinst.com
FTP Site: ftp.natinst.com
Web Address: http://www.natinst.com

Bulletin Board Support
BBS United States: 512 794 5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

Fax-on-Demand Support
512 418 1111

Telephone Support (USA)
Tel: 512 795 8248
Fax: 512 794 5678

International Offices
Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 288 3336,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, Israel 03 6120092, Italy 02 413091,
Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00,
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200,
United Kingdom 01635 523545

National Instruments Corporate Headquarters
6504 Bridge Point Parkway Austin, Texas 78730-5039 USA Tel: 512 794 0100

© Copyright 1998 National Instruments Corporation. All rights reserved.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

CVI™, FlexMotion™, LabVIEW™, natinst.com™, National Instruments™, and ValueMotion™ are trademarks of National Instruments Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

About This Manual

Organization of This Manual xi
Conventions Used in This Manual xii
Related Documentation xiii
Customer Communication xiii

Chapter 1
Introduction

About the ValueMotion Software 1-1
What You Need to Get Started 1-1
Installing a ValueMotion Board 1-1
Software Programming Choices 1-2
    National Instruments Application Software 1-2
ValueMotion Language Support 1-2

Chapter 2
Building Applications with the ValueMotion Software Library

The ValueMotion Windows Libraries 2-1
    Creating a 32-Bit Windows Application 2-2
        Buffer Allocation 2-2
        Parameter Passing 2-2
    Creating a Windows 3.1 Application 2-2
    Creating a LabWindows/CVI Application under Windows 95/98/NT 2-3
    Creating LabWindows/CVI Application under Windows 3.1 2-3
The ValueMotion DOS Library 2-3
    Creating a DOS Application 2-3

Chapter 3
Software Overview

Function Types 3-1
    Board-Level Functions 3-1
    Per-Axis Specific Functions 3-1
Initialization 3-1
    Recommended Initialization Procedure 3-1
    Limit and Home Switch Configuration 3-3
    Position Modes 3-3
        Absolute Position Mode 3-4

Relative Position Mode 3-4
Velocity Mode 3-5
I/O Port Configuration 3-5
Step Output Mode and Polarity 3-6
PID Filter Parameters 3-6
Find Home 3-7
Find Index 3-7
Motion Trajectory 3-8
Motion Trajectory Overview 3-8
Motion Trajectory Parameters 3-9
Acceleration and Acceleration Factor 3-10
Base Velocity 3-11
Stop Mode Selection 3-11
Decelerated Stop 3-11
Halt Stop 3-12
Kill Stop 3-12
Stop On Following Error 3-12
Closed-loop Functions 3-13
Motion Interactions with Limit and Home Switches 3-13
Advanced Features 3-14
Communications Buffers 3-14
Board-Level Communication Function Buffer 3-15
Return Data Buffer 3-15
I/O Port Trigger Buffers 3-15
Trigger Buffer Functions 3-16
Breakpoint Functions 3-17
24-Bit Digital I/O 3-18
Error Conditions 3-19

Chapter 4
Function Reference

Status Codes and Board IDs 4-1
Variable Data Types 4-2
Primary Types 4-2
Arrays 4-3
Programming Language Considerations 4-3
Visual Basic for Windows 4-3
ValueMotion Constants Include File 4-3
_rdb Functions 4-4
Special Considerations for DOS Users 4-5
acquire_samples (Servo only) 4-6
begin_prestore 4-8

communicate 4-10
enable_brk (Closed-Loop Stepper and Servo only) 4-12
enable_io_trig 4-14
enable_limits 4-16
enable_pos_trig (Closed-Loop Stepper and Servo only) 4-18
end_prestore 4-19
find_home 4-20
find_index
  find_index_rdb (Closed-Loop Stepper and Servo only) 4-22
flush_rdb 4-25
get_board_type 4-26
get_motion_board_info 4-28
get_motion_board_name 4-30
in_pos 4-32
kill_motion 4-33
load_accel 4-34
load_accel_fact (Stepper only) 4-36
load_break_mod (Closed-Loop Stepper and Servo only) 4-38
load_deriv_gain (Servo only) 4-40
load_deriv_per (Servo only) 4-41
load_fol_err (Closed-Loop Stepper and Servo only) 4-43
load_intg_gain (Servo only) 4-45
load_intg_lim (Servo only) 4-47
load_pos_brk (Closed-Loop Stepper and Servo only) 4-48
load_pos_ref (Closed-Loop Stepper and Servo only) 4-51
load_pos_scale (Closed-Loop Stepper and Servo only) 4-53
load_prop_gain (Servo only) 4-55
load_rot_counts (Closed-Loop Stepper and Servo only) 4-56
load_rpm 4-58
load_rpsps 4-60
load_steps_lines (Closed-Loop Stepper only) 4-62
load_target_pos 4-64
load_time_brk (Servo only) 4-67
load_vel 4-70
load_vel_change (Stepper only) 4-73
master_slave_cfg (Servo only) 4-75
multi_start 4-77
read_adc (Closed-Loop Stepper and Servo only) 4-79
read_axis_stat
  read_axis_stat_rdb 4-81
read_csr 4-84
read_encoder
  read_encoder_rdb (Closed-Loop Stepper only) 4-86

read_gpio
  read_gpio_rdb (Stepper only) 4-88
read_io_port
  read_io_port_rdb 4-90
read_lim_stat
  read_lim_stat_rdb 4-92
read_pos
  read_pos_rdb 4-94
read_rdb 4-96
read_rpm 4-98
read_steps_vel (Stepper only) 4-100
read_vel
  read_vel_rdb (Servo only) 4-102
reset_pos 4-104
send_command 4-105
set_base_vel (Stepper only) 4-106
set_direction (Servo only) 4-107
set_gpio (Stepper only) 4-108
set_io_output 4-109
set_io_pol 4-111
set_lim_pol 4-113
set_loop_mode (Closed-Loop Stepper only) 4-115
set_portc_dir (Stepper only) 4-116
set_pos_mode 4-117
set_rs_pulse 4-119
set_scale_seq (Closed-Loop Stepper and Servo only) 4-121
set_step_mode_pol (Stepper only) 4-123
set_stop_mode 4-125
start_motion 4-127
stop_motion 4-129
store_elc (Servo only) 4-130
store_steps_rev (Stepper only) 4-131
trig_buff_delim 4-132
trigger_io 4-133

Appendix A
Error Codes

Appendix B
ValueMotion Function Support

Appendix C
Customer Communication

Glossary

Index

Figures

Figure 3-1. Trapezoidal Trajectory Profile 3-8
Figure 3-2. Trapezoidal Trajectory Profile with Acceleration Factor..... 3-11

Tables

Table 3-1. Required Initialization Steps 3-2
Table 3-2. Motion Trajectory Parameter Functions 3-9
Table 3-3. Communication Buffers 3-14
Table 3-4. Trigger Buffer Size in Function Packets 3-16

Table 4-1. Primary Type Names 4-2
Table 4-2. Trigger Buffer Size in Function Packets 4-9
Table 4-3. Shipping and Non-Shipping Devices 4-26

Table A-1. Error Codes Summary A-2

Table B-1. ValueMotion Function Attribute Summary B-1
Table B-2. ValueMotion Function Call Board Support B-6

About This Manual

The *ValueMotion Software Reference Manual* is for users of the ValueMotion software. ValueMotion software is a powerful application programming interface (API) between your motion control application and the National Instruments ValueMotion boards for ISA, PCI, and PXI bus computers.

Organization of This Manual

The *ValueMotion Software Reference Manual* is organized as follows:

- Chapter 1, *Introduction*, describes the ValueMotion software.
- Chapter 2, *Building Applications with the ValueMotion Software Library*, describes the fundamentals of creating ValueMotion applications in Windows 3.1/95/98/NT, and DOS.
- Chapter 3, *Software Overview*, describes the features and functionality of the ValueMotion boards and briefly describes the ValueMotion function.
- Chapter 4, *Function Reference*, contains a detailed explanation of each ValueMotion function.
- Appendix A, *Error Codes*, describes the error codes returned by the ValueMotion software.
- Appendix B, *ValueMotion Function Support*, contains tables that summarize ValueMotion function attributes and shows which ValueMotion functions each ValueMotion board supports.
- Appendix C, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and documentation.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual:

»	The » symbol leads you through nested menu items to a final action. The sequence **File»Page Setup»Options»Substitute Fonts** directs you to pull down the **File** menu, select the **Page Setup** item, select **Options**, and finally select the **Substitute Font** options from the last dialog box.
♦	The ♦ symbol indicates that the text following it applies only to a specific board or to a specific type of motion.
	This icon to the left of bold italicized text denotes a note, which alerts you to important information.
	This icon to the left of bold italicized text denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.
bold	Bold text denotes the names of menus, menu items, parameters, dialog boxes, dialog box buttons or options, icons, or windows.
bold italic	Bold italic text denotes a note, caution, or warning.
Closed-Loop Stepper	Refers to the PC-Step-2CX, PC-Step-4CX, PCI-Step-2CX, PCI-Step-4CX, and the PXI-7324.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text from which you supply the appropriate word or value, as in Windows 3.*x*.
`monospace`	Text in this font denotes text or characters that you should literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of directories, programs, subprograms, functions, filenames and extensions, and for statements and comments taken from programs.
Open-Loop Stepper	Refers to the PC-Step-2OX, PC-Step-4OX, PCI-Step-2OX, PCI-Step-4OX, and the PXI-7314.
Servo	Refers to the PC-Servo-2A, PC-Servo-4A, PCI-Servo-2A, and PCI-Servo-4A.
Stepper	Refers to the PC-Step-2OX, PC-Step-4OX, PC-Step-2CX, PC-Step-4CX, PCI-Step-2OX, PCI-Step-4OX, PCI-Step-2CX, PCI-Step-4CX, PXI-7314, and PXI-7324.

Related Documentation

The following documents contain information that you may find helpful as you read this manual:

- *ValueMotion Hardware User Manual*
- *ValueMotion Function Reference Online Help*

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix C, *Customer Communication*, at the end of this manual.

1

Introduction

This chapter describes the ValueMotion software.

About the ValueMotion Software

ValueMotion software includes a set of functions you use to control the ValueMotion servo and stepper boards for point to point and multi-axis linear motion. ValueMotion software combined with ValueMotion servo and stepper boards provide functionality and power for integrated motion systems for use in laboratory, test, and production environments.

Your ValueMotion software package includes example programs to help get you up and running quickly, as well as a function test environment for use with Windows, called pcRunner that allows you to test each of the ValueMotion functions.

What You Need to Get Started

To set up and use your ValueMotion software, you need the following:

- ❑ ValueMotion software
- ❑ *ValueMotion Software Reference Manual*
- ❑ ValueMotion hardware

Installing a ValueMotion Board

Use the Motion Board Configuration Utility to configure and verify your ValueMotion boards. The procedure for installing your ValueMotion board depends upon your operating system and bus interface type (ISA, PCI, and PXI). Refer to the `Install.txt` for detailed information regarding ValueMotion board installation. In addition, refer to the *ValueMotion Hardware User Manual* for information on I/O jumper settings and bus address DIP switches (for ISA boards).

Software Programming Choices

You have several options to choose from when programming your National Instruments ValueMotion software. You can use National Instruments application software such as LabVIEW and LabWindows/CVI, or third-party application development environments (ADEs) such as Borland C/C++, Microsoft Visual C/C++, Microsoft Visual Basic, or any other Windows-based compiler that can call into Windows dynamic link libraries (DLLs) for use with the ValueMotion software.

National Instruments Application Software

LabVIEW features interactive graphics, a state-of-the-art user interface, and a powerful graphical programming language. The ValueMotion VI Library, a set of VIs for using LabVIEW with National Instruments ValueMotion hardware, is available separately.

LabWindows/CVI features interactive graphics, state-of-the-art user interface, and uses the ANSI standard C programming language. The functions that comprise the ValueMotion Software Library can be called from LabWindows/CVI.

Using LabVIEW or LabWindows/CVI software will greatly reduce the development time for your motion control application.

ValueMotion Language Support

The ValueMotion software library is a DLL in Windows 3.1/95/98/NT, and is a DOS library in MS-DOS. You can use the Windows DLL with any Windows development environment that can call Windows DLLs. Chapter 2, *Building Applications With the ValueMotion Software Library*, provides more specific information on building Windows applications with Microsoft Visual C/C++, Microsoft Visual Basic, and Borland C/C++.

ValueMotion software also comes with DOS libraries for Microsoft and Borland C compilers. See Chapter 2 for more specific information on how to build applications with these DOS compilers.

2

Building Applications with the ValueMotion Software Library

This chapter describes the fundamentals of creating ValueMotion applications in Windows 3.1/95/98/NT, and DOS.

The following section contains general information about building ValueMotion applications, describes the nature of the ValueMotion files used in building ValueMotion applications, and explains the basics of making applications using the following tools:

- LabWindows/CVI
- Borland C/C++
- Microsoft Visual C/C++
- Microsoft Visual Basic

If you are not using one of the tools listed, consult your development tool reference manual for details on creating applications that call DLLs.

☞ **Note** ***Refer to your*** *ValueMotion Hardware User Manual* ***for information on installing your ValueMotion boards.***

The ValueMotion Windows Libraries

The ValueMotion for Windows function libraries are dynamic link libraries (DLLs) which means that ValueMotion functions are not linked into the executable files of applications. Only the information about the ValueMotion functions in the ValueMotion import libraries is stored in the executable files.

In Windows 95/98/NT, the ValueMotion DLL is called `pcMotion32.dll`; in Windows 3.1, it is called `pcmotion.dll`. If you are programming in C or C++, link in the appropriate import library, which is different for Microsoft and Borland development tools.

Import libraries contain information about their DLL-exported functions. They indicate the presence and location of the DLL routines. Depending on the development tools you are using, you can make your compiler and linker aware of the DLL functions through import libraries or through function declarations.

Using function prototypes is a good programming practice. This is why ValueMotion is packaged with function prototype files for different Windows development tools. If you are not using any of the development tools referred to in this chapter, you may need to create your own function prototype file based on the files provided with the ValueMotion software.

Creating a 32-Bit Windows Application

In Windows 95/98/NT, your ValueMotion applications call the file `pcMotion32.dll`. The header file for C programmers is `pcMotion32.h`. Microsoft C/C++ programmers should link to the `pcMotionMS.lib` import library. Borland C/C++ programmers should link to the `pcMotionBC.lib` import library. Example programs are in the `ValueMotion\Examples` directory.

Visual Basic programmers should include the `pcMotion32.bas` header file.

Buffer Allocation

To allocate memory, use the Windows functions `GlobalAlloc()` and `GlobalFree()`.

Parameter Passing

By default, C passes parameters by value. Remember to pass pointers to the address of a variable when you need to pass by reference.

Creating a Windows 3.1 Application

In Windows 3.1, your ValueMotion applications call the file `pcmotion.dll`. The header file for C/C++ programmers is `pcmotion.h` Microsoft C/C++ programmers and Borland C/C++ programmers should link to the `pcmotion.lib` import library. Example programs are in the `Valmotn\Examples` directory.

Visual Basic programmers should include the `pcmotion.bas` header file.

Creating a LabWindows/CVI Application under Windows 95/98/NT

To create an application for Windows 95/98/NT using LabWindows/CVI, link to the appropriate ValueMotion import library. If you set the default compiler compatibility mode to Microsoft Visual C++, link to `pcMotionMS.lib`. If you set the default compiler compatibility mode to Borland C++, link to `pcMotionBC.lib`. The header file is `pcMotion32.h`.

Creating LabWindows/CVI Application under Windows 3.1

To create an application for Windows 3.1 using LabWindows/CVI, include `pcmotion.h` and `pcmotion.dll` in your project by selecting **File»Insert»File**. Use `pcmotion.h` as the header file. The LabWindows/CVI compiler generates the necessary glue code to call into the 16-bit pcmotion DLL under Windows 3.1.

The ValueMotion DOS Library

The ValueMotion DOS libraries for C programmers are in the `.lib` file, which you link to your application. Quick Basic users can use three low-level functions provided for communicating with the board.

Creating a DOS Application

To develop a C application in DOS, include the `valuedos.h` header file and link to the `pcmotms.lib` (Microsoft) on `pcmotbc.lib` (Borland) libraries.

Example programs are in the `valmotn\examples` directory.

3

Software Overview

This chapter describes the features and functionality of the ValueMotion boards and briefly describes the ValueMotion functions.

Function Types

The variety of functions available for programming the ValueMotion boards may be divided into two basic types: board-level functions and per-axis specific functions.

Board-Level Functions

Board-level functions are typically used for setting up, controlling, and reading back data for multi-axis functions and board-level functions. These functions are primarily system enhancement set up and control functions for external input and I/O port configuration. A few examples of board-level functions are: Set Limit Switch Input Polarity, Enable Limit Switch Inputs, Set I/O Port Polarity and Direction, and Multi-Axis Start.

Per-Axis Specific Functions

Call per-axis specific functions for setting up, controlling, and reading back data on an individual axis basis. These functions have the identical functionality for the control circuits on all axes.

Initialization

A large number of motion parameters need be set only once during initialization. If needed, you can change these settings during your motion control application. The following sections provide an overview of the motion parameters typically set during initialization, as well as some motion procedures that are commonly executed during set up.

Recommended Initialization Procedure

This section presents a recommended list of functions you can execute during board-level and per-axis initialization. This recommended list

covers the minimum areas of initialization for basic motion control. Additional functions must be added to this list for enhanced system configuration-initialization requirements.

Execute the following list of functions for each axis used in the motion control application. The per-axis control circuits will not function properly if these initialization steps are not performed. Table 3-1 illustrates the required initialization steps for the ValueMotion boards.

Table 3-1. Required Initialization Steps

Servo	Stepper
Set Operation Mode (`set_pos_mode`)	Set Operation Mode (`set_pos_mode`)
Set Stop Mode (`set_stop_mode`)	Set Stop Mode (`set_stop_mode`)
Load Filter Proportional Gain (`load_prop_gain`)	Set Loop Mode (`set_loop_mode`) Closed-Loop only
Load Filter Derivative Gain (`load_intg_gain`)	Load Steps and Lines/Rev (`load_steps_lines`) Closed-Loop only
Load Filter Derivative Sample Period (`load_deriv_per`)	N/A
Load Velocity (`load_vel`)	Load Steps/Sec (`load_vel`)
Load Acceleration (`load_accel`)	Load Acceleration (`load_accel`)

This is the minimum initialization list. For applications using limit and home switches, you may need to execute the following board-level functions:

- Set Limit Switch Input Polarity (`set_lim_pol`)
- Enable Limit Switch Inputs (`enable_limits`)

☞ **Note** ***If your drive requires inverting step and direction signals, you will need to call* Set Step Output Mode and Polarity *(*`set_step_mode_pol`*).***

Other functions typically used during initialization include:

- Find Home (`find_home`)
- Find Index (`find_index`) servo and closed-loop stepper only
- Reset Position (`reset_pos`)

- Load Following Error (`load_fol_err`) servo and closed-loop stepper only
- Load Filter Integral Gain (`load_intg_gain`) servo only

At power-up, all axes are *killed* with the axis circuits disabled. An axis will automatically start to servo (servo boards) or the motor will be energized (stepper boards) when the Start Motion (`start_motion`) function is executed or the Stop Motion (`stop_motion`) function is executed with Set Stop Mode (`set_stop_mode`) set to halt stop, which is the power-up default if you do not call the Set Stop Mode function. If the stop mode has been set to another mode, you can use the following sequence of functions to turn on the servo controller or energize the motor for an axis without moving anywhere:

1. Reset Position Counter to Zero
2. Load Target Position = 0
3. Start

Limit and Home Switch Configuration

If operating each motion axis includes limit switch and home switch inputs, set up and initialize the parameters for this function. The limit switch parameters may be set up by using the following functions:

- Set Limit Switch Input Polarity (`set_lim_pol`)
- Enable Limit Switch Inputs (`enable_limits`)

These functions are board-level functions and may be used to set up the limit switch and home input switches for all four axis control circuits. The limit and home switch input signal polarity defaults to inverting input (active low) for all signal sources. If limit or home switch inputs are to be used, their polarity must be properly set prior to enabling. Improperly configuring limit switch polarity when the limit switch signals are enabled causes faulty operation.

Position Modes

You can program your ValueMotion board to operate in one of three position modes—absolute position mode, relative position mode, or velocity mode. These modes may be independently programmed for each of the four axes, using the Set Operation Mode (`set_pos_mode`) function. If you do not call this function to set the position mode, your ValueMotion board defaults to absolute position mode.

Absolute Position Mode

All motion control in this mode uses the trapezoidal profile generation for motion execution. This mode is used for absolute positioning applications, with the reference to position equal to zero as determined by the power-on location or the reset position counter to zero location typically associated with a home switch and/or index position.

When the individual axis control circuits are set up for absolute position mode, loaded target position values are interpreted as the absolute position count value of the motion destination. When a Start Motion function is issued following a Load Target Position function, motion proceeds to the loaded target position. When motion is completed successfully, the Read Position function returns a position count equal to the loaded target position. The velocity and acceleration values loaded in this mode are used to generate and control the trapezoidal move profile. See Figure 3-2 for more information.

- Servo Only

  In this mode, when the destination position is reached, the servo circuitry maintains the position until a new position valued is loaded and a Start Motion function is issued, or until a Kill Motion function is issued. The position tracking servo circuitry turns off the motor position maintenance if the following error (difference between desired and actual position) exceeds a programmed count value.

Relative Position Mode

All motion control in this mode uses the trapezoidal profile generation for motion execution. This mode is used for relative positioning applications, with reference to the present position.

When the individual axis control circuit is set up for relative position mode, loaded target position values are interpreted as relative position count values. The loaded value is added to the present position count to determine the new destination position.

For example:

If the present position is 100

and the loaded target position is 200 in relative position mode,

then the new destination position is 300.

Destination Position = Present Position + Newly Loaded Target Position.

Velocity Mode

In velocity mode, motion is continuous in the direction selected at the velocity programmed. This mode uses velocity and acceleration input parameter values to perform continuous velocity control. This mode is used for continuous stable velocity control applications and velocity profiling (regular updating of velocity parameters for velocity profile generation). Position feedback information is available for positional tracking and the full functionality of limit switch and home switch input signals is available in this mode.

In this mode, when the target velocity is reached following acceleration, the axis circuit maintains the velocity until an Incremental Velocity Change (`load_vel_change`) function (stepper only) is issued, a Load Velocity (`load_vel`) and Start Motion (`start_motion`) sequence is executed (servo only) or until a Kill Motion or Stop Motion function is issued. On servo and closed-loop stepper boards, the velocity tracking loop kills motion if the following error exceeds a programmed following error count.

If the axis circuit detects a valid home or limit switch signal, the axis circuit brings motion to a halt.

I/O Port Configuration

If operating the ValueMotion boards includes the programmable I/O port bits, set up the board-level parameters. You can set this port on an individual bit basis for input or output signals. You must set the hardware jumpers to correspond to the programmed I/O direction for each bit. Refer to Chapter 4, *Signal Connections*, of your *ValueMotion Hardware User Manual*, for more information on setting the jumpers.

When set up as output signals, you can program the polarity of each bit as inverting or noninverting (active low or high) for desired output levels.

When set up as input signals, each bit may be configured as an inverting or noninverting input and may be enabled as trigger input I/O port bits.

Servo boards have eight lines of I/O while the stepper boards have four lines of I/O. Lines 5 through 8 on a stepper board are dedicated as outputs for the inhibit lines and must be configured using Set Step Output Mode and Polarity (`set_step_mode_pol`).

Use the following functions for I/O port configuration and output value reading and control:

- Set I/O Port Polarity and Direction (`set_io_pol`)
- Set I/O Port Output (`set_io_output`)
- Read I/O Port (`read_io_port`/`read_io_port_rdb`)
- Enable I/O Port Trigger Inputs (`enable_io_trig`)

Refer to Chapter 4, *Function Reference*, for detailed descriptions of each function covered in this area.

Step Output Mode and Polarity

♦ Stepper

Use Set Step Output Mode and Polarity (`set_step_mod_pol`) to configure the interface between the stepper board and external stepper motor drivers. This function is a board-level function where a single command programs all four axes simultaneously. Each axis has three output lines to control the stepper drivers. You can configure two of these lines in either step and direction or clockwise (CW) and counter-clockwise (CCW) pulse mode for compatibility with the two industry standards. The third line implements the inhibit function. All three outputs have programmable polarity for compatibility with active high or active low inputs. See the Set Step Output Mode and Polarity (`set_step_mode_pol`) function for more information.

PID Filter Parameters

♦ Servo

Determining filter parameter depends entirely upon the system dynamics for the motion control elements involved. Filter parameters include:

- Filter Derivative Sample Period (`load_deriv_per`)
- Filter Proportion Gain (`load_prop_gain`)
- Filter Integral Gain (`load_intg_gain`)
- Filter Derivative Gain (`load_deriv_gain`)
- Filter Integration Limit (`load_intg_lim`)

Set up these parameters on an individual axis basis prior to motion control. The parameter values affect loop response-and-update time, impulse response, holding torque, and many other factors. Motor size, speed of operation, acceleration requirements, position tracking, motor loading, and other variables affect the proper determination of these parameters.

See Chapter 4, *Function Reference*, for recommended starting values for these parameters. You should be able to use these starting values; however, the values may vary according to your application requirements.

☞ **Note** ***PID filter parameters are updated by the axis immediately after they are loaded.***

Find Home

The Find Home (`find_home`) function requires properly configured and enabled limit switch and home switch signals to function properly. To ensure the home position is properly set, the Find Home function uses limit switch data and home switch data. The Find Home function is ignored unless both limit and home switch inputs are enabled. The Find Home function searches the full range of motion between the forward and reverse limits looking for the home switch. If a home switch is not encountered, motion stops at either the forward or reverse limit, depending on whether you started the search going backwards or forwards.

☞ **Notes** ***The Find Home function requires that all three limit and home switches be enabled or the function will be ignored. This is a safety feature built into the ValueMotion boards.***

Unused limit or home switches must be configured into their inactive state with the Set Limit Switch Input Polarity (`set_lim_pol`) function before being enabled.

Find Index

The Find Index (`find_index`) function requires a valid encoder index signal for the per-axis control circuit programmed with this function. The Find Index function looks for an index transition within one revolution of the present position. If it encounters the index, it makes the index location its target position and moves to that position.

An optional offset value may be used with the Find Index function. A non-zero offset value will be added to the index location and become the target position for the index move.

Motion Trajectory

This section covers features related to motion trajectory, starting and stopping motion, and the circumstances under which the board automatically stops motion.

Motion Trajectory Overview

Motion trajectory programming involves trajectory parameter loading and executing the Start/Stop Motion function. All motion control on ValueMotion boards is through trapezoidal profiles; which means, the motor accelerates up to the programmed velocity, traverses at the programmed velocity, and then decelerates at the programmed acceleration rate to the target position. In closed-loop stepper systems, any lost steps are made up with a final *pull-in* move. See Figure 3-1 for a typical trapezoidal trajectory profile.

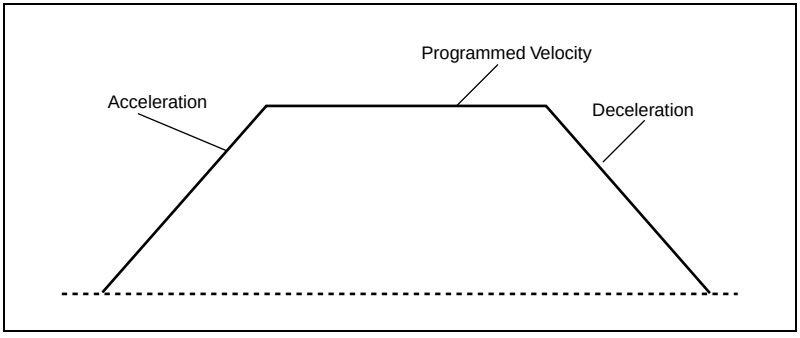


Figure 3-1. Trapezoidal Trajectory Profile

You begin motion with the Start Motion (`start_motion`) function and stop it with the Stop Motion (`stop_motion`) or Kill Motion (`kill_motion`) functions. Motion stops automatically when the programmed trajectory is complete, when an enabled limit or home switch is encountered, or when the programmed following error trip point is exceeded.

Trajectory parameter values are loaded and retained in the per-axis control circuits until new values are loaded or until the hardware is reset. In general, position values loaded in absolute mode replaces previously loaded values, and values loaded in relative mode are added to the present position to generate a new target position.

- Servo Only

  Trajectory values updated during a move take effect when the next Start Motion function is executed. This *on-the-fly* capability means that motion profiles significantly more complex than the trapezoidal profile shown in Figure 3-1 are possible. For example, you can use velocity profiling (regular updating of velocity parameters for velocity profile generation) for continuous path contouring applications. The type of control used may be as varied as the possible applications developed for use with the servo board. The servo board is provided as a generic servo motion control tool and allows full custom configuration depending upon the requirements of your individual motion system application.

- Stepper Only

  Trajectory values updated during a move take effect when the next start motion function is executed, only if the axis is in a stopped state. If the axis is in motion, the Start Motion function will be ignored. The exception to this rule is the Incremental Velocity Change (`load_vel_change`) function which takes effect immediately.

  All position values are loaded and returned in steps, and velocity values are loaded and returned in steps per second (steps/s). Acceleration is loaded in steps/s^2. Closed-loop following error value is loaded in steps. Only the Read Encoder function returns a position value in quadrature counts.

Motion Trajectory Parameters

You must set up the motion trajectory parameters properly prior to motion control on an individual axis basis. Table 3-2 describes the functions for setting the motion trajectory parameters. These parameters are double buffered on the board and will be used by the next valid start command to determine the trajectory. Since they are double buffered, these parameters do not need to be reloaded for each move, only when it is necessary to change a parameter for a subsequent move.

Table 3-2. Motion Trajectory Parameter Functions

Servo	Stepper
Load Acceleration (`load_accel`)	Load Acceleration (`load_accel`)
Load RPM (`load_rpm`)	Load Steps/Sec (`load_vel`)
Load Target Position (`load_target_pos`)	Load Target Position (`load_target_pos`)

Table 3-2. Motion Trajectory Parameter Functions (Continued)

Servo	Stepper
N/A	Load Acceleration Factor (`load_accel_fact`)
N/A	Set Base Velocity (`set_base_vel`)

Parameter values are loaded via their individual functions and must be set to a known value prior to motion control. Their values are set to defaults at power-up and, if left unchanged, will create a *do nothing* trajectory with a target position equal to zero when a Start Motion function is issued. Refer to Chapter 4, *Function Reference*, for detailed information on programming considerations for each of these functions.

Acceleration and Acceleration Factor

Acceleration is the rate of change of velocity per unit time. The acceleration value is used for both the acceleration and deceleration parts of the trapezoidal trajectory. When a stop motion command is used in conjunction with the set stop type (`set_stop_mode`) set to decelerated mode, motion stops using the programmed acceleration value.

- Stepper only

  Acceleration factor is an enhancement to stepper trajectories. Acceleration Factor is a value between 1 and 10 that determines the ratio between starting acceleration and high speed acceleration. Figure 3-2 shows a typical trapezoidal profile for three values of acceleration factor. With acceleration factor = 1, the programmed acceleration value in steps/s^2 determines the linear rate of increase in velocity during the acceleration and deceleration portions of the trapezoidal profile.

  An acceleration factor >1 creates a nonlinear acceleration profile by multiplying the programmed acceleration value at low velocities and then tapering off with increasing velocity until the programmed acceleration value is reached as the motor reaches its target step rate. This low speed acceleration boost results in markedly shorter acceleration and deceleration times without affecting high speed acceleration or exceeding the high speed torque limit characteristics of many stepper motors.

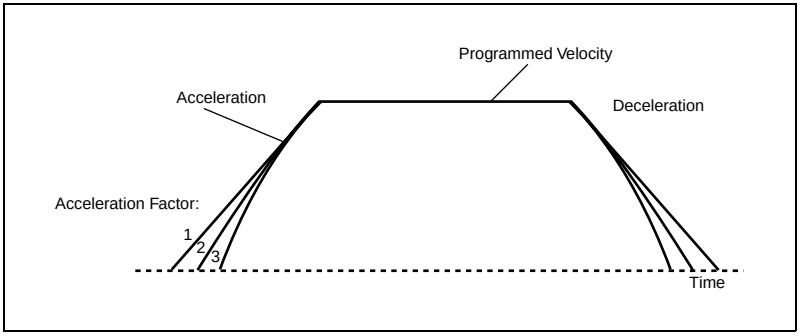


Figure 3-2. Trapezoidal Trajectory Profile with Acceleration Factor

Base Velocity

- Stepper only

For proper stepper motor operation in full-step and half-step applications, the acceleration and deceleration portions of the trapezoidal profile should begin and end with a minimum step rate. This base velocity (start-stop velocity) is programmable from 50 to 5,000 steps/s. The power-up default value of 250 steps/s should be adequate for most applications. You can change the base velocity with the `set_base_vel` function.

Stop Mode Selection

With ValueMotion boards you can preprogram the way the motor stops when Stop Motion (`stop_motion`) is issued. The stop modes are:

- Decelerated Stop
- Halt Stop (default)
- Kill Stop

The stop mode desired is set by using the Set Stop Mode (`set_stop_mode`) function. This function sets the desired stop mode to be used by the Stop Motion function execution on a per-axis basis.

Decelerated Stop

In this mode, motion immediately decelerates from the present trajectory using the programmed acceleration value until the motor comes smoothly to a complete stop.

Halt Stop

Halt stop causes an instantaneous stop in an abrupt fashion at maximum deceleration. This is the default stop type used by all limit and home input signal controlled stops. For servo boards, motion is stopped immediately by the axis controller and PID loop, which sets the destination position value equal to the present position when the Stop Motion function is executed. For stepper boards, the board stops motion immediately by terminating the step pulse generation when the Stop Motion function is executed.

Kill Stop

When this stop mode is selected and the Stop Motion function is executed on a per-axis basis, the motor is turned off. Stop commands issued with this mode selected behave just like the Kill Motion function. On servo boards, the board sets the motor drive signal to zero and a motor off condition occurs in the per-axis servo circuit. This mode turns off position tracking and allows the motor to freewheel. On stepper boards, the board stops motion immediately by terminating the step pulse generation and activating the inhibit output. You can use the inhibit output to disable the stepper driver and allow the motor to freewheel.

Stop On Following Error

- Servo and Closed-Loop Stepper Only

  You can program a following error count on a per-axis basis. Exceeding the following error causes an automatic internal Kill Motion function to execute. The following error count is programmed in quadrature counts by using the Load Following Error (`load_fol_err`) function.

  The Load Following Error function affords protection against motor run-away and position tracking or position input signal loss. This function can also be used to protect against a stalled motor or bound motion hardware as well as other mechanical failures and conditions that may cause excessive position tracking errors.

Closed-loop Functions

- Closed-Loop Stepper Only

  The closed-loop version of the stepper board adds position tracking, verification, and end of move *pull-in* features to the standard stepper functionality. The following functions are used to program these features:

  - Set Loop Mode (`set_loop_mode`)
  - Load Steps and Lines/Rev (`load_steps_lines`)
  - Load Following Error (`load_fol_err`)
  - Read Encoder (`read_encoder/read_encoder_rdb`)
  - Find Encoder Index (`find_index`)

  Closed-loop control is implemented by the board by constantly comparing the step count to the encoder quadrature count. At the end of each move, any lost steps are made up with a *pull-in* move to guarantee that the motor reaches its target position. The board will attempt up to three pull-in moves before it gives up trying to reach a target position. If these three pull-in moves fail, a following error condition occurs.

  These closed-loop functions require accurately loaded values for steps per motor revolution and encoder lines per revolution. These parameters are loaded with the Load Steps and Lines/Rev function.

  Use the Set Loop Mode function to enable and disable the closed-loop functions. You can read the encoder position in quadrature counts at any time with the Read Encoder (`read_encoder`) function, even when the closed-loop functions have been disabled.

☞ **Notes** ***Closed-loop stepper boards power-up default in open-loop mode.***

Servo boards are inherently closed-loop. There is no way to put them in open-loop mode.

Motion Interactions with Limit and Home Switches

This section covers function interactions with the Limit and Home Switches.

You must pay special attention to your ValueMotion board operation in reference to limit and home switches. The Start Motion (`start_motion`) and Multi-Axis Start (`multi_start`) functions operate in a unique fashion when used with valid limit switch input signals. Both functions check the limit switch input signals to determine if any are active prior to starting

motion. If a limit switch is active, and the desired direction of motion is further into the limit condition, as opposed to away from the limit condition, the start function is ignored. For this safety and control feature to function properly, the limit switch inputs must be properly configured and enabled including proper connection for forward and reverse limits at the appropriate end-of-travel positions.

Caution ***You must take care to ensure that the forward and reverse limits are consistent with the desired direction of motion. The limit switches must be wired so that a move set in the forward or positive direction moves the motion system towards the forward limit Switch.***

Advanced Features

This section covers advanced and auxiliary features of your ValueMotion board, and the functions used to access those features.

Communications Buffers

ValueMotion boards provide Random Access Memory (RAM) buffers for incoming function buffering and return data buffering. Altogether, there are six different buffers in the board communications section. Table 3-3 summarizes these buffers.

Table 3-3. Communication Buffers

Function Group	Buffer Size	Description
Communications Buffer	1 Buffer	For command and control functions from the host
Return Data Buffer	1 Buffer	For return data storage
I/O Trigger Buffers	4 Buffers	1 each for the four I/O trigger functions

The buffers are all structured as first-in-first-out (FIFO) where incoming functions are buffered and executed in the order that they arrive. Return data packets available for host reads are buffered in the order that they are generated.

Board-Level Communication Function Buffer

The first buffer is an input function buffer for the board-level set up and control functions. If functions are buffered, they are executed in the order that they are received.

Return Data Buffer

The sixth buffer is the Return Data Packet Buffer. The ValueMotion boards use this buffer to hold return data packets until the host is ready to read them out. The information contained in this buffer is generated by individual data requesting or generating functions. This buffer is 1800 data-packets deep on the servo boards and 30 data-packets deep on the stepper boards. The host can read this buffer at any time when the Return Data Pending status bit is valid in the Communications Status Register. To check the status of this bit, use the `read_csr` function.

For most applications, the Return Data Buffer needs to be only one packet deep. In general, data should be read from the Return Data Buffer immediately after it is requested. You can use the `read_rdb` function or `communicate` function in mode 2 for this purpose.

I/O Port Trigger Buffers

The last four buffers are the I/O port trigger buffers. Use these buffers to store functions that are automatically loaded and executed by a valid trigger input signal or by direct control via the manual Trigger I/O Port (`trigger_io`) function.

Load function sequences into these buffers by using the Begin Trigger Buffer Prestore (`begin_prestore`) and End Trigger Buffer Prestore (`end_prestore`) functions. You can load single or multiple move sequences. Multiple sequences are delimited by the Trigger Buffer Delimiter (`trig_buff_delim`) function. Once loaded, these buffers are maintained regardless of the number of times the buffer is triggered. The only way the buffer contents change is if you write new functions over the old ones or if the hardware resets due to power cycling.

Table 3-4 illustrates the size of the individual trigger buffers for both servo and stepper boards.

Table 3-4. Trigger Buffer Size in Function Packets

Trigger Buffer Number	Size in Function Packets (Servo)	Size in Function Packets (Stepper)
1	1,024	640
2	1,024/2,048[1]	640/1280[1]
3	1,024	640
4	2,048	1280

[1] All trigger buffers have permanently allocated memory space except trigger buffer number 2. It can hold up to 1,000 functions (without impacting trigger buffer number 3) or up to 2,000 function by using the space reserved for trigger buffer number 3. If you are using trigger buffer number 3, you must limit trigger buffer number 2 to 1,000 functions to avoid overwriting the functions in trigger buffer number 3.

Notes: Function packet count is based on all four word packets.

Size in function packets is given as worst case scenario.

The trigger buffer function is a powerful high level set of function tools that allow preprogrammed motion and multi-axis coordinated motion initiation via a single function or input bit trigger.

Trigger Buffer Functions

This section presents a brief description of the trigger buffer functions on the ValueMotion board.

With trigger buffer functions you can preprogram a sequence of functions and store this sequence in a trigger buffer. You then can use a single function or a previously enabled I/O port input bit that has been designated as a trigger input bit.

For example, you can prestore a sequence of functions including Load Velocity, Load Target Position, and Start Motion in a trigger buffer and execute them with a single call to the Trigger I/O Port function or by providing an active signal on an enabled I/O port trigger input.

You can prestore functions in the trigger buffers by using the following functions:

- Begin Trigger Buffer Prestore (`begin_prestore`)
- End Trigger Buffer Prestore (`end_prestore`)
- Trigger Buffer Delimiter (`trig_buff_delim`)

Caution ***You should execute the trigger buffer prestore only at a time when motion control interaction with the host computer is not required. All normal function execution is disabled during trigger buffer function sequence prestore.***

To use the I/O port bits as trigger inputs, configure the I/O port bits using the Enable I/O Port Trigger Inputs (`enable_io_trig`) function.

For trigger buffers, you can manually initiate the trigger execution function by using the Trigger I/O Port (`trigger_io`) function. The Trigger I/O Port function operates regardless of the I/O port or trigger input enable set up.

You can pre-store single or multiple function sequences by using the Trigger Buffer Delimiter (`trig_buff_delim`) function. The Trigger Buffer Delimiter function separates and defines the limits of the multiple sequences. Each time a trigger buffer is triggered, the contents of the buffer up to the next delimiter (or to the end of the buffer) are automatically transferred as if the functions were sent from the host. The trigger buffer contents remain undisturbed in their original sequence. When the end of the buffer is reached, the next trigger retriggers the first function sequence up to the first delimiter.

Breakpoint Functions

♦ Servo and Closed-Loop Stepper Only

You can program breakpoints to generate an external signal transition on a dedicated I/O port bit when a desired position or anticipation-time is reached. The two types of available breakpoints are:

- Position Breakpoint (`load_pos_brk`)
- Load Anticipation Time Breakpoint (`load_time_brk`) (Servo Only)

When breakpoints are set up and the breakpoint location is reached, a level transition occurs on a properly enabled I/O port bit. I/O port bits 1, 2, 3, and 4 are dedicated for use with the corresponding axes 1, 2, 3, and 4.

You can specify position breakpoints as either an absolute position or a relative position when the breakpoint is enabled. Relative breakpoints are relative to the starting position of the particular move.

Breakpoint data is not used by the board until an Enable Breakpoint (`enable_brk`) function is issued. The data is buffered internally to the ValueMotion board so if the same value is desired on a repetitive basis, it is not necessary to reload the position or time data value after each use.

☞ **Note** ***You have to reenable the breakpoint each time you reload the position or anticipation-time value and after each breakpoint occurrence.***

- Servo

  With anticipation breakpoints you can generate a breakpoint at a programmable time prior to completing a move in position mode. This time value is a multiple of base sample periods (341 µs).

  The two breakpoint types, position and anticipation time, share the same low level hardware and are mutually exclusive. The most recently programmed breakpoint type is the one that is used when the breakpoint function is enabled for the axis selected.

24-Bit Digital I/O

- Stepper

  Stepper boards have an additional 24 lines of digital I/O, which can be accessed via the auxiliary 50-pin connector on the board. These 24 lines are broken down into three 8-bit ports. Port 1 is a dedicated input port, port 2 is a dedicated output port, and port 3 can be configured for either input or output. Use the following functions to configure, read, and write to these digital lines:

  - Set Port C Direction (`set_portc_dir`)
  - Read Auxiliary Digital I/O Input Values (`read_gpio`)
  - Set Auxiliary Digital I/O Values (`set_gpio`)

  For details on the electrical characteristics of these digital lines refer to your *ValueMotion Hardware User Manual*.

Error Conditions

ValueMotion boards minimize error conditions. There are three possible fatal failure modes.

Fatal failure modes require system power cycling to reset the board in order to attempt recovery. These error types may indicate an internal hardware problem on the board.

The three fatal failure modes are: System Fail Status Bit Valid, Watchdog Timer Terminal Count Reached, and Communications Loss.

Fatal failure modes are unlikely but, if they occur, try to clear them by power cycling. If this procedure does not clear the problem, contact National Instruments Technical Support. Appendix A, *Error Codes*, contains a detailed listing of the error codes returned by ValueMotion functions.

4

Function Reference

This chapter contains a detailed description of each ValueMotion function.

The functions are arranged alphabetically by function name. See Appendix B, *ValueMotion Function Support*, for additional information. Table B-1, *ValueMotion Function Attribute Summary*, in Appendix B lists the descriptive name of all the ValueMotion functions. For example, Read Communications Status Register is the descriptive name for the `read_csr` function. Table B-2, *ValueMotion Function Call Board Support*, in Appendix B shows which functions are valid on each ValueMotion board, since a few functions are available on one variation of bus (e.g. PCI bus) while not available on another variation of bus (e.g. ISA bus).

Status Codes and Board IDs

Every ValueMotion function is of the following form:

status = `function_name` (parameter 1, parameter 2, … parameter *n*)

where $n \geq 0$. Each function returns a value in the **status** variable that indicates the success or failure of the function. A returned status of zero indicates that the function executed successfully. A non-zero status indicates that the function did not execute because of an error, or executed with an error.

☞ **Note** ***In Windows 95/98/NT,* status *is a 32-bit integer. In Windows 3.1 and DOS,* status *is a 16-bit integer.***

The first parameter to almost every ValueMotion function is the boardID of the ValueMotion device you want the driver to use for the given operation. The boardID is assigned by the *Motion Board Configuration Utility*, located under the **Start** menu. You can use multiple ValueMotion devices in one application; to do so, simply pass the appropriate boardID to each function.

Variable Data Types

Every function description has a parameter table that lists the data types for each parameter. The following sections describe the notation used in those parameter tables and throughout the manual for variable data types.

Primary Types

Table 4-1 shows the primary type names and their ranges.

Table 4-1. Primary Type Names

Type Name	Description	Range	Type		
			C/C++	Visual BASIC	Pascal (Borland Delphi)
u8	8-bit ASCII character	0 to 255	`char`	Not supported by BASIC. For functions that require character arrays, use string types instead.	`Byte`
i16	16-bit signed integer	–32,768 to 32,767	`short`	`Integer` (for example: `deviceNum%`)	`SmallInt`
u16	16-bit unsigned integer	0 to 65,535	`unsigned short for 32-bit compilers`	Not supported by BASIC. For functions that require unsigned integers, use the signed integer type instead. See the i16 description.	`Word`
i32	32-bit signed integer	–2,147,483,648 to 2,147,483,647	`long`	`Long` (for example: `count&`)	`LongInt`
u32	32-bit unsigned integer	0 to 4,294,967,295	`unsigned long`	Not supported by BASIC. For functions that require unsigned long integers, use the signed long integer type instead. See the i32 description.	`Cardinal` (in 32-bit operating systems). Refer to the i32 description.

Table 4-1. Primary Type Names (Continued)

Type Name	Description	Range	Type		
			C/C++	Visual BASIC	Pascal (Borland Delphi)
f32	32-bit single-precision floating point	–3.402823 x 10^{38} to 3.402823 x 10^{38}	`float`	`Single` (for example: `num!`)	`Single`
f64	64-bit double-precision floating point	–1.797683134862315 × 10^{308} to 1.797683134862315 × 10^{308}	`double`	`Double` (for example: `voltage Number`)	`Double`

Arrays

When a primary type is inside square brackets, [i16], an array of the type named is required for that parameter.

Programming Language Considerations

Apart from the data type differences, there are a few language-dependent considerations you need to be aware of when you use the ValueMotion API. Read the following sections that apply to your programming language.

☞ **Note** ***Be sure to include the ValueMotion function prototypes by including the appropriate ValueMotion header file in your source code. Refer to Chapter 2, Building Applications with the ValueMotion Software Library, for the header file appropriate to your operating system and compiler.***

Visual Basic for Windows

When you pass arrays to ValueMotion functions using Visual Basic for Windows, you need to pass the first element of the array by reference. For example, you would pass in buffer % (0) as your parameter.

ValueMotion Constants Include File

The file `MotnCnst.bas` contains definitions for constants required for `get_motion_board_info` function. You should use the constants symbols in this function; do not use the numerical values.

In Visual Basic for Windows, you can add the entire MotnCnst.bas file into your project. You then will be able to use any of the constants defined in this file in any module in your program.

To add the MotnCnst.bas file for your project in Visual Basic 3.0 and 4.0, go to the **File** menu and select the **Add File...** option. Select MotnCnst.bas, which is in the ValueMotion\include directory. Then, select **Open** to add the file to the project.

To add the MotnCnst.bas file to your project in Visual Basic 5.0, go to the **Project** menu and select **Add Module**. Click on the **Existing** tab page. Select MotnCnst.bas, which is the ValueMotion\include directory. Then, select **Open** to add the file to the project.

This procedure is identical to the procedure you follow when loading the Visual Basic file CONSTANT.TXT. Search on the word *CONSTANT* for more information from the Visual Basic online help.

_rdb Functions

There are two types of functions that return data from the board, the standard type and the _rdb type. For example, the standard function read_io_port has the following function prototype:

status = read_io_port (boardID, iodata)

where **boardID** is an input and **iodata** is an output. When you use this function, the ValueMotion software places the data retrieved from the board into the **iodata** variable that you referenced when calling the function.

The _rdb version of this function takes only one parameter as follows:

status = read_io_port_rdb (boardID)

The return data is placed in the Return Data Buffer, which can be read later. Use the communicate function in mode 2, or the read_rdb function to retrieve the data from the Return Data Buffer.

You should use the standard version (without the _rdb) of the function unless you are concerned about the extra time it will take for the ValueMotion software to read the Return Data Buffer and place the result in a function's output parameters.

You should also use the _rdb functions when storing functions in a trigger buffer. Trigger buffers provide a mechanism for storing onboard programs (or macros) to be executed later. You must use the _rdb version of the

function, since there is no output variable into which to place the return value. A separate program running on the host computer must read the data out of the Return Data Buffer.

Special Considerations for DOS Users

If you are using the DOS library and programming in C, you must pass in an extra parameter to each ValueMotion function. For example, the function

status = acquire_samples (boardID, numSamples)

becomes

status = acquire_samples (boardAddr, boardType, numSamples)

where

boardAddr is type **u16**, containing the base address of the board, and **boardType** is type **u32**, containing the board type number. The board types for each ValueMotion board are listed in the description for `get_board_type` and in the files `MotnCnst.h` and `MotnCnst.bas` in the `ValueMotion\include` directory.

The following functions have another parameter:

- Find Index (`find_index`)
- Load RPM (`load_rpm`)
- Load RPSPS (`load_rpsps`)
- Read RPM (`read_rpm`)

The additional parameter is **revdata** which is a u16 containing the lines per revolution. The description for each function contains the new prototype for the function in a special note to DOS users.

acquire_samples (Servo only)

Acquire Samples

Format

status = acquire_samples (boardID, numSamples)

Purpose

Fills the Return Data Buffer with samples of multi-axis positions, velocity, and I/O data.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
numSamples	u16	number of samples

Parameter Discussion

numSamples is the number of samples to place in the Return Data Buffer. **numSamples** has a range of:

0 to 200

☞ **Note** ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and functions normally.***

Using This Function

This function fills the Return Data Buffer with samples of multi-axis position, velocity, and I/O data with the following structure:

- Axis 1 position
- Axis 1 velocity
- Axis 2 position
- Axis 2 velocity
- Axis 3 position
- Axis 3 velocity
- Axis 4 position
- Axis 4 velocity
- I/O port value

Samples are placed in the buffer one after another with nothing in between at a rate of approximately one sample every 2.3 ms. The Return Data Buffer has space for 200 samples.

To use the Acquire Samples function, wait at least (3 ms) × (Number of Samples) after executing the function and then read out the entire Return Data Buffer.

You can read the data using Communicate (`communicate`) in mode 2 or Read Return Data Buffer (`read_rdb`). To see the format of the returned data, see Read Position (`read_pos`) for position data, Read Velocity (`read_vel`) for velocity data, and Read I/O Port (`read_io_port`) for I/O data.



Caution ***Communicating with the servo board while the*** Acquire Samples ***function is executing disrupts the uniformity of the sampling period. Executing functions that request data inserts Return Data Packets into the Return Data Buffer and corrupts the sample data format described above.***

The Acquire Samples function allows you to capture real-time servo performance data that you can use to tune the servo PID parameters for optimum performance.

begin_prestore

Begin Trigger Buffer Prestore

Format

status = begin_prestore (boardID, trigbuffnum)

Purpose

Initiates the trigger buffer storage function.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
trigbuffnum	u16	indicates in which trigger buffer to store functions

Parameter Discussion

trigbuffnum initiates the trigger buffer where functions are stored. Legal values are 1 through 4.

Using This Function

Trigger buffer function storage is a powerful feature you can use to prestore a series of functions in a trigger buffer. All functions called after this function are stored in the trigger buffer indicated by **trigbuffnum** 1 through 4. The End Trigger Buffer Prestore (`end_prestore`) function ends the trigger buffer storage function and returns the board to normal operation. Functions loaded during the trigger buffer storage function will not be executed when they are sent. They will be stored for future use by a valid, enabled trigger input or by using the Trigger I/O Port (`trigger_io`) function.

Table 4-2 illustrates the size of the individual trigger buffers for both servo and stepper boards.

Table 4-2. Trigger Buffer Size in Function Packets

Trigger Buffer Number	Size in Function Packets (Servo)	Size in Function Packets (Stepper)
1	1,024	640
2	1,024/2,048[1]	640/1280[1]
3	1,024	640
4	2,048	1280

[1] All trigger buffers have permanently allocated memory space except trigger buffer number 2. It can hold up to 1,000 functions (without impacting trigger buffer number 3) or up to 2,000 function by using the space reserved for trigger buffer number 3. If you are using trigger buffer number 3, you must limit trigger buffer number 2 to 1,000 functions to avoid overwriting the functions in trigger buffer number 3.

Notes: Function packet count is based on all four word packets.

Size in function packets is given as worst case scenario.

All trigger buffers except trigger buffer number 2 have permanently allocated memory space. Trigger buffer number 2 can hold up to 1,000 function packets without impacting trigger buffer number 3, or up to 2,000 function packets by using the space reserved for trigger buffer number 3.

Caution ***If you are using trigger buffer number 3, trigger buffer number 2 must be limited to 1,000 functions or less to avoid overwriting functions in trigger buffer number 3. The board does not check to see if trigger buffer number 2 exceeds 1,000 functions. Because excess loading of the buffers causes erroneous operation, it is recommended that you verify the overflow.***

Caution ***Multiple trigger buffers may not be loaded simultaneously. Executing the*** Begin Trigger Buffer Prestore ***function while trigger buffer storage is in process causes a function error.***

Note ***Care should be taken when using this function, since all direct control of the board is disabled during trigger buffer storage. Use this function only during system set up prior to motion control processing or when all motion is stopped.***

communicate

Communicate

Format

status = communicate (boardID, mode, wcount, axis, cmd, data)

Purpose

Sends functions to the board and/or reads data from the Return Data Buffer.

Parameter

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
mode	u8	execution mode
wcount	u8	number of words in data packet

Input/Output

Name	Type	Description
axis	u8	axis to be controlled
cmd	u8	function ID
data	u32	function specific data

Parameter Discussion

mode is the type of operation to be performed.

0: Send function only

1: Send function and read data back

2: Read data stored in the Return Data Buffer

wcount is the number of 16-bit words stored in data (for sending functions only). **wcount** is calculated as 2 + the number of data words being sent. In mode 2, this parameter is ignored.

axis is the axis affected by the function (both sending functions and receiving data).

cmd is the unique function ID value of the function.

data either passes data with a function or receives data from the Return Data Buffer when the board is in read mode. It has a maximum size of two 16-bit data words.

Using This Function

This function is a register-level programming type function. It is recommend that you use the more advanced DLL functions. Every function that can be performed by this function has a corresponding, simpler function call.

Refer to Table B-1, *ValueMotion Function Attribute Summary*, in Appendix B, *ValueMotion Function Support*, for a complete list of function names and their function IDs.

There are three modes of operation—send function only (mode 0), send function and read data back (mode 1), and read data (from Return Data Buffer) only (mode 2). For mode 0, **axis** is the axis you want affected (if applicable), **cmd** is the function ID value of the function you want performed, and **data** is information the function may need. Mode 0 leaves the return data in the Return Data Buffer. In mode 1, Communicate reads the data in the Return Data Buffer and places it in the **axis**, **cmd**, and **data** output parameters. Mode 1 does not leave anything in the Return Data Buffer. Mode 2 retrieves from the Return Data Buffer, the axis, function ID, and return data of a previously issued function.



Note ***To successfully use mode 1, the Return Data Buffer must be empty.***

Functions that have names ending in `_rdb` depend on this function to read the values they will store in the Return Data Buffer. To read from the Return Data Buffer, specify mode 2. The Return Data Buffer is a queue. As `_rdb` functions are called, their return data is queued up in the Return Data Buffer. To avoid having the buffer fill up, the values in the buffer either need to be retrieved using Communicate (`communicate`) in mode 2 or the buffer needs to be cleared using Flush Return Data Buffer (`flush_rdb`).

enable_brk (Closed-Loop Stepper and Servo only)

Enable Breakpoint Function

Format

status = enable_brk (boardID, axis, mode)

Purpose

Enables a breakpoint for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
mode	i16	position breakpoint mode

Parameter Discussion

axis is the axis to be controlled.
mode specifies the breakpoint mode for position breakpoints.

0 position breakpoint is absolute

–1 position breakpoint is relative

Using This Function

Use this function to set and enable the position at which an external breakpoint signal is generated on the corresponding digital I/O line. The position loaded by the Load Breakpoint Position (`load_pos_brk`) function is used as the desired breakpoint location.

- Axis 1 uses the I/O port 1 (stepper) or 5 (servo) bit as its dedicated breakpoint output line.
- Axis 2 uses the I/O port 2 (stepper) or 6 (servo) bit as its dedicated breakpoint output line.
- Axis 3 uses the I/O port 3 (stepper) or 7 (servo) bit as its dedicated breakpoint output line.
- Axis 4 uses the I/O port 4 (stepper) or 8 (servo) bit as its dedicated breakpoint output line.

mode is used to set the interpretation mode for breakpoints only. The mode specified here is independent of the Set Operation Mode (`set_pos_mode`) function selected for the corresponding axis and does not affect the motion trajectory profile.

If the breakpoint function is enabled using **mode** set for absolute mode, the breakpoint output signal is generated on the corresponding digital I/O line when the absolute axis position is equal to the programmed breakpoint position.

If the breakpoint function is enabled using **mode** set for relative mode, the breakpoint output signal is generated on the corresponding digital I/O line when the axis position is equal to the sum of the starting position and the loaded relative breakpoint position.

- Servo

  This function also enables breakpoints loaded by the Load Anticipation Time Breakpoint (`load_time_brk`) function.

- Stepper

  The position breakpoint function will not operate when the closed-loop stepper is in open-loop mode. See the Set Loop Mode (`set_loop_mode`) function for more information.



Caution ***This function requires the closed-loop version of the stepper board. Attempting to execute this function on the open-loop board generates a function error.***

enable_io_trig

Enable I/O Port Trigger Inputs

Format

status = enable_io_trig (boardID, trigdata)

Purpose

Enables I/O port bits 1 through 4 for use as trigger inputs.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
trigdata	u16	trigger inputs enable bitmap

Parameter Discussion

trigdata has the following format.

D15	D14	D13	D12	D11	D10	D9	D8
0	0	0	0	0	0	0	0

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	I/O Bit 4 Trig Enable	I/O Bit 3 Trig Enable	I/O Bit 2 Trig Enable	I/O Bit 1 Trig Enable

1 = Enable trigger input
0 = Disable trigger input (default)

Using This Function

Use this function to enable I/O port bits 1 through 4 for use as trigger inputs. You can use I/O port bits enabled as trigger inputs to automatically execute a preprogrammed list of functions that have been stored in a trigger buffer. You can use I/O port bits that have their direction and hardware jumpers set as inputs, as simple signal input bits, as trigger input bits or both.

When properly set for direction and polarity, and when enabled by use of this function, an active signal on the I/O port input trigger causes the stored functions relating to the I/O port bit number to be executed. At power-up, all I/O trigger inputs are disabled by default.

Trigger input functionality may also be linked by the software function Set Breakpoint To Trigger Link (`enable_pos_trig`) to the breakpoint output bits. In this software link mode, an active breakpoint output causes a valid trigger input on the corresponding trigger buffer. Use this function to enable the individual I/O port bits as event triggers.

enable_limits

Enable Limit Switch Inputs

Format

status = enable_limits (boardID, enabledata)

Purpose

Enables and disables the specified axis' limit and home switch inputs.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
enabledata	u16	limits enable bitmap

Parameter Discussion

enabledata has the following format:

D15	D14	D13	D12	D11	D10	D9	D8
0	0	0	0	Axis 1 Home Switch	Axis 2 Home Switch	Axis 3 Home Switch	Axis 4 Home Switch

D7	D6	D5	D4	D3	D2	D1	D0
Axis 1 Fwd Limit	Axis 1 Rev Limit	Axis 2 Fwd Limit	Axis 2 Rev Limit	Axis 3 Fwd Limit	Axis 3 Rev Limit	Axis 4 Fwd Limit	Axis 4 Rev Limit

1 = Enable limit switch
0 = Disable limit switch (default)

Using This Function

The limit switch inputs provide a halt stop on limit switch function, as well as prohibit motion beyond the limit (as long as the limit input remains active True). In order for the stop on limit switch function to work properly, the limit switch inputs must be correctly configured for polarity by the Set Limit Switch Input Polarity (`set_lim_pol`) function. This must occur prior to using the Enable Limit Switch Inputs (`enable_limits`) function to properly enable the limit switch inputs. At power-up, all limit and home switches are disabled by default.

When correctly set for polarity and enabled by this function, the active transition on a limit or home switch input causes the motion for the corresponding axis to be halt stopped.

☞ **Note** ***Valid transitions on enabled limit and home signal inputs are used by the*** Find Home ***(***`find_home`***) function. You must enable the limit and home switch inputs before you execute the*** Find Home ***(***`find_home`***) function. You may enable the home switch inputs for special function purposes in addition to the predetermined*** Find Home ***(***`find_home`***) function.***

enable_pos_trig (Closed-Loop Stepper and Servo only)

Set Breakpoint To Trigger Link

Format

status = enable_pos_trig (boardID, axis, trigbuffnum)

Purpose

Sets a software link between the breakpoint output for the axis specified and any trigger buffer input.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
trigbuffnum	u16	trigger buffer number

Parameter Discussion

axis is the axis to be controlled.

trigbuffnum is the number corresponding to the trigger buffer to be linked.

0 = Disabled (No link) Default
1 through 4 = trigger buffer number

Using This Function

When an enabled breakpoint is reached, the linked trigger buffer is automatically triggered. This triggering is in addition to the normal breakpoint functionality. See Load Position Breakpoint (`load_pos_brk`) and Enable Breakpoint (`enable_brk`) functions.

Since you can program a trigger buffer to set another breakpoint, this function allows you to completely automate sequences of motion trajectories to be executed without host intervention.

You can disable this function and remove the link by sending a trigger buffer number equal to zero to the axis. The power-up default is: all links disabled.

☞ **Note** ***Multiple axis breakpoints can be linked to trigger any individual trigger buffer but each breakpoint can only trigger one trigger buffer at a time.***

end_prestore

End Trigger Buffer Prestore

Format

status = end_prestore (boardID)

Purpose

Ends the trigger buffer storage function.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility

Using This Function

All functions sent after the Begin Trigger Prestore function are stored in the trigger buffer indicated by the data word sent with the Begin Trigger Buffer Prestore (`begin_prestore`) function. The End Trigger Buffer Prestore function ends the trigger buffer storage function and returns the board to normal processing. Functions loaded during the trigger buffer storage function are not executed. They are stored for future use by a valid, enabled trigger input or by using the Trigger I/O Port (`trigger_io`) function.

Trigger buffer function storage is a powerful feature that prestores a list of functions in a trigger buffer. to retrieve the stored functions, you can execute an external trigger event or use the Trigger I/O Port (`trigger_io`) function.



Caution ***Multiple trigger buffers cannot be loaded simultaneously. Executing the*** Begin Trigger Buffer Storage ***function while trigger buffer storage is already in process causes a function error.***

Before you execute this function, you must call the Begin Trigger Buffer Prestore ***(***`begin_prestore`***) function. If you do not first call*** `begin_prestore`***, a function error occurs.***

find_home

Find Home

Format

status = find_home (boardID, axis, direction)

Purpose

Starts the find home sequence on the specified axis in the specified direction.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
direction	i16	direction in which to start looking for home

Parameter Discussion

axis is the axis to be controlled.

direction is the direction the find home sequence will start in:

0 forward

–1 reverse

Using This Function

This function starts the find home sequence on the specified axis. This function requires properly connected, configured, and enabled limit switch and home switch input signals.

The Find Home function generates and executes motion profiles that search for the home switch. All motion profiles executed by this function require correctly loaded velocity and acceleration parameters.

If **direction** = 0 then

Find Home = Forward

The Find Home function starts motion in the forward direction until the home switch input signal is detected. Motion then stops. If motion reaches the forward limit switch prior to the home switch, motion reverses and proceeds in the direction of the reverse limit switch. If and

when the home switch is encountered, motion stops. If no home switch is found, motion reaches the reverse limit switch and stops.

If **direction** = –1 then

Find Home = Reverse

The Find Home function starts motion in the reverse direction until the home switch input signal is detected. Motion then stops. If motion reaches the reverse limit switch prior to the home switch, motion changes direction and proceeds toward the forward limit switch. If and when the home switch is encountered, motion stops. If no home switch is found, motion reaches the forward limit switch and stops.

Many functions are illegal and should not be used during a Find Home sequence. Attempting to execute any function other than the following valid functions generates a Function Not Executed Error status. See Read Per-Axis Hardware Status (`read_axis_stat`) and Read Communications Status (`read_csr`) for more details on error status reporting.

Valid functions during Find Home include the following:

- Kill Motion (`kill_motion`)
- Stop Motion (`stop_motion`)
- Set Stop Mode (`set_stop_mode`)
- Read Per-Axis Hardware Status (`read_axis_stat`)
- Read Position (`read_pos`)
- Read Velocity (`read_vel`)
- Set I/O Port Output (`set_io_output`)
- Read I/O Port (`read_io_port`)
- Read Communications Status (`read_csr`)
- Set I/O Port Polarity and Direction (`set_io_pol`)
- Set Limit Switch Input Polarity (`set_lim_stat`)
- Read Limit Switch Status (`read_lim_stat`)
- Read A/D Analog Input (`read_adc`)
- Read 24 Bit I/O Digital Inputs (`read_gpio`)
- Set 24 Bit I/O Digital Outputs (`set_gpio`)

☞ **Note** Find Home ***uses the last loaded velocity, for specified axis, to perform the search.***

find_index
find_index_rdb (Closed-Loop Stepper and Servo only)

Find Encoder Index

Format

status = find_index (boardID, axis, direction, offset, reserved)
status = find_index_rdb (boardID, axis, direction, offset)

Purpose

Initiates the Find Index function for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
direction	i16	search direction
offset	i16	offset from index

Output

Name	Type	Description
reserved	u16	status word

Parameter Discussion

axis is the axis to be controlled.

direction is the search direction, forward or reverse.

0 forward

–1 reverse

offset is the desired position offset from the index. It is in steps for a closed-loop stepper board, and in encoder lines for a servo board.

1 through 32767 forward index value

–1 through –32768 reverse index value

reserved is ignored for 4- and 2-axis boards. Just pass in NULL for this variable.

☞ **Note** ***For obsolete 3-axis boards,* reserved *is a returned data value. 0xFFFF means*** Find Index ***failed. 0x0000 means*** Find Index ***was successful (3-axis boards are no longer available).***

Using This Function

This function requires a properly connected and configured index input signal. The signal must be an active low signal. See your *ValueMotion Hardware User Manual* for more information on encoder signal connections.

The Find Encoder Index function generates and executes a motion profile using the loaded velocity and acceleration parameters. The initial find index motion is in the direction selected via the direction parameter. The move continues for slightly over one motor revolution. This motion profile allows the per-axis motion control circuitry to locate a single index occurrence. After recording the index location during the single revolution move, motion continues to a new target position equal to the recorded index position plus or minus the offset value. All of these moves occur at a slow velocity that is scaled from the loaded velocity value.

During the find index sequence do not attempt to call any other functions. Also the find index sequence will not start while any of the following conditions are true:

- triggers are enabled
- the axis is running
- the axis is a slave axis
- the axis is in open-loop mode
- the reverse limit is active
- the forward limit is active

☞ **Notes** ***When using*** Find Encoder Index ***(***`find_index_rdb`***) reserved *is only placed in the Return Data Buffer for obsolete 3-axis boards. 4- and 2-axis boards do not place anything in the Return Data Buffer. Use*** Communicate ***(***`communicate`***) in mode 2 to retrieve this data (if applicable).***

For DOS users, this function's prototype is:* status = `find_index` (u16 boardaddr, u16 boardtype, u8 axis, u16 revdata, u16 direction, u16 offset, u16 *indexptr). *Refer to *Special Considerations for DOS Users* ***earlier in this chapter for more information.***

♦ Servo

This function requires that the correct value for encoder lines per revolution be entered with Store Encoder Line Count (`store_elc`) function.

- Stepper

  This function requires that the correct values for motor steps per revolution and encoder lines per revolution be entered with the Load Steps and Lines/Rev (`load_steps_lines`) function and that a properly connected index signal input be present for the specified axis.



Caution ***This function requires the closed-loop version of the stepper board. Attempting to execute this function on the open-loop board generates a function error. If the closed-loop stepper board is operating in open-loop mode and the*** Find Index ***function is issued, the Find Index Status word indicates that the board failed to find the index. See the*** Set Loop Mode ***(*`set_loop_mode`***) function for more information on configuring open and closed-loop operation.***



Note ***For stepper boards, the position is reset to zero when index is found. When this function is complete, the position will be the same value as the offset specified to this function.***

flush_rdb

Flush Return Data Buffer

Format

status = flush_rdb (boardID)

Purpose

Clears all data in the Return Data Buffer.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility

Using This Function

Calling this function removes all data in the Return Data Buffer.

get_board_type

Get Board Type

Format

status = get_board_type (boardID, boardtype)

Purpose

Returns the **boardtype** of the board corresponding to **boardID**.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility

Output

Name	Type	Description
boardtype	u8	type of board specified by boardID

Parameter Discussion

boardtype is the type of board specified by **boardID**.

Using This Function

Use this function to determine what type of board exists at a specific **boardID**. The numeric value of a board's type is passed back in **boardtype**. Table 4-3 shows the board type and the corresponding device.

Table 4-3. Shipping and Non-Shipping Devices

Board Type	Device	Non-Shipping Device[1]
7	PC-Servo-2A	—
3	PC-Servo-4A	—
8	PC-Step-2OX	—
4	PC-Step-4OX	—
9	PC-Step-2CX	—

Table 4-3. Shipping and Non-Shipping Devices (Continued)

Board Type	Device	Non-Shipping Device[1]
5	PC-Step-4CX	—
16	PC-FlexMotion-6C	—
17	PCI-Servo-2A	—
11	PCI-Servo-4A	—
18	PCI-Step-2OX	—
12	PCI-Step-4OX	—
19	PCI-Step-2CX	—
13	PCI-Step-4CX	—
24	PCI-FlexMotion-6C	—
22	PXI-7312	—
20	PXI-7314	—
23	PXI-7322	—
21	PXI-7324	—
0	—	pcControl
1	—	pcStep-OL
2	—	pcStep-CL
6	—	pcEncoder-4A
14	—	pciEncoder-4A

[1] Refers to a device that is no longer available.

get_motion_board_info

Get Motion Board Info

Format

status = get_motion_board_info (boardID, infoType, infoValue)

Purpose

Retrieve information about the properties and capabilities of your motion control board.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
infoType	u32	type of information you want to retrieve

Output

Name	Type	Description
infoValue	u32	retrieved information

Parameter Discussion

infoType represents the type of information you wish to retrieve from the ValueMotion driver. Legal values are given in terns of constants defined in the `MotnErr.h` (C programmers) or `MotnErr.bas` (Basic programmers) header files.

infoValue returns the value you want to retrieve. Value is returned either in terms of constants from the header file, or as numbers, as appropriate.

The following table lists all legal values for **infoType** and the possible return values.

InfoType	Possible Return Values
`NIMC_BOARD_FAMILY`	`NIMC_FLEX_MOTION` `NIMC_VALUE_MOTION`
`NIMC_NUM_AXES`	Number of axes on the board

`NIMC_BUS_TYPE`	`NIMC_ISA_BUS`
	`NIMC_PCI_BUS`
	`NIMC_PXI_BUS`

InfoType	Possible Return Values
NIMC_BOARD_TYPE	See list of constants in get_board_type description
NIMC_VALUEMOTION_BOARD_CLASS	NIMC_SERVO NIMC_CLOSED_LOOP_STEPPER NIMC_OPEN_LOOP_STEPPER
NIMC_CLOSED_LOOP_CAPABLE	NIMC_TRUE (1) NIMC_FALSE (0)

☞ **Note** ***For C Programmers,* infoValue *is a pass-by-reference parameter.***

C Example

As an OEM providing a two board solution (PCI-Servo-4A and PCI-Step-4CX) to your customer, you want to write an application that will work regardless of the order in which the two boards have been configured. You need to figure out which board is the servo board and which is the stepper board, assuming that they will be boardIDs 1 and 2.

```
#include “MotnErr.h”
u8 boardID = 1;
i32 err = NIMC_noError;
u32 infoType = NIMC_VALUEMOTION_BOARD_CLASS;
u32 Value1, Value2;

err = get_motion_board_info (boardID, infoType, &infoValue1);
if (err == NIMC_noError)
   err = get_motion_board_info (boardID, infoType, &infoValue2);
if (err != NIMC_noError)
   bail out.
if (Value1 == NIMC_Servo && Value2 == NIMC_CLOSED_LOOP_STEPPER)
   board1 is a Servo and board 2 must be Stepper.
else if (Value1 == NIMC_CLOSED_LOOP_STEPPER && Value2 == NIMC_Servo)
   board 1 is the Stepper, board 2 must be Servo.
else
   Configuration error - the boards must not be configured
   incorrectly.
```

get_motion_board_name

Get Motion Board Name

Format

status = get_motion_board_name (boardID, charArray, sizeOfArray)

Purpose

Returns the name of the board in ASCII format in the user-supplied character array.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility

Input/Output

Name	Type	Description
sizeOfArray	u32	size of character array

Output

Name	Type	Description
charArray	[u8]	character array

Parameter Discussion

sizOfArray is an I/O parameter that specifies the size of the character array being passed into this function. the value in **sizeOfArray** is considered when the ValueMotion driver software decides whether or not it can copy the board name into the character array. If the size is insufficient, a **NIMC-insufficientSizeError** is returned by the function, and the required character array size is returned in the **sizeOfArray** parameter. If the size is sufficient, the name is copied into the character array, and the number of bytes copied (plus one for the NULL terminator) is placed in the **sizeOfArray** parameter.

charArray is an array of characters, allocated by the calling program. The ValueMotion driver will place the name of the board, referenced by boardID, in the character array, provided there is sufficient space.

Using This Function

If NULL (or 0) is passed in for the **charArray** parameter, the size of a character array required to hold the board name will be placed in the **sizeOfArray** parameter. This can be used when you want to allocate only the memory necessary to hold the board name. `get_motion_board_name` is then called twice, once to get the required array size, and once again to actually retrieve the name.

☞ **Notes** ***The number of characters required for the character string is always one more than the actual number of characters in the board name due to the NULL terminator at the end of the string. For example, the board name PXI-7324 is 8 characters long, so you must provide a 9 character array to hold this name.*** **SizeOfArray** ***must be 9 or greater as an input, and upon successful copy of the board name, a value of 9 will be placed in*** **sizeOfArray.**

For C Programmers, **sizeOfArray** ***is pass-by-reference parameter.***

in_pos

In Position

Format

status = in_pos (boardID, axis, target, deadband, result)

Purpose

Checks if present position is the same as target position or no farther away from target position than **deadband**.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
target	i32	target position
deadband	u32	allowable variance

Output

Name	Type	Description
result	u32	results from comparison

Parameter Discussion

axis is the axis to be controlled.

target is the position you want to match against the axis' present position.

deadband is the delta distance from the target position that is allowable.

result is 0 if present position is **target** or within the **deadband**. Otherwise, **result** is the distance between the present position and **target**.

Using This Function

This function reads the axis' present position. It compares this position with the target position to see if it is less than **deadband** away.

kill_motion

Kill Motion

Format

status = kill_motion (boardID, axis)

Purpose

Kills motion on a specific axis.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled

Parameter Discussion

axis is the axis to be controlled.

Using This Function

Use this function to kill the motion for the axis specified by **axis**. This function stops the motion for the axis specified using the kill stop mode. The kill stop mode stops motion by disabling the motor control signal. When motion is killed and the per-axis inhibit output signal is connected to a driver/amplifier, the inhibit output signal controls the driver/amplifier disable (inhibit) function.

Typically, use this function to evoke an emergency stop situation and clear any pending buffered functions for the axis specified.



Caution ***This function purges the individual axis specific input function trigger buffer of any pending functions, including functions that have been loaded by a valid trigger buffer condition.***

- Stepper

  For stepper output, stepping stops immediately and the step or CW/CCW pulse outputs stop.

load_accel

Load Acceleration

Format

status = load_accel (boardID, axis, acceleration)

Purpose

Loads the desired acceleration on a per-axis basis.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
acceleration	u32	acceleration value

Parameter Discussion

axis is the axis to be controlled.

acceleration is the acceleration value that the axis' motion trajectory control circuitry uses. See *servo* and *stepper* sections for correct ranges.

- Servo

  The acceleration is loaded in counts per sample per sample.

  The maximum acceleration values loaded must remain within the following boundaries:

  1 through 16,777,215

  The default value for acceleration at power-up is 524,288.

- Stepper

  The acceleration is loaded in steps per second per second.

  The maximum acceleration value loaded must remain within the following range:

  1 through 5,242,879

  The default value for acceleration at power-up is: 3.

Note ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

Using This Function

The acceleration loaded is used by the motion trajectory control circuitry to control acceleration and deceleration for all motion trajectories requiring this function. Acceleration is loaded on a per-axis basis.

Caution ***You cannot load the acceleration value when a motion trajectory executes and the motor is running for the axis. If an acceleration function is received while the axis is in motion, it is ignored. Acceleration values must only be loaded when motion is stopped.***

load_accel_fact (Stepper only)

Load Acceleration Factor

Format

status = load_accel_fact (boardID, axis, accfactor)

Purpose

Sets the acceleration factor for the axis specified.

Parameter

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
accfactor	u16	acceleration factor

Parameter Discussion

axis is the axis to be controlled.

accfactor is the acceleration factor value to be set.

The acceleration factor must be in the range of:

1 through 10

The default value for acceleration factor at power-up is 1.

☞ **Note** ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

Using This Function

The acceleration factor boosts the low speed acceleration and creates a nonlinear acceleration profile. The acceleration factor may be used to optimize the acceleration portion of a stepper trajectory. This may help to overcome the torque curve limits and mechanical system constraints on velocity/acceleration for stepper systems. Markedly shorter acceleration and deceleration times can be achieved without affecting high speed acceleration or exceeding the high speed torque limit of the stepper motor.

The acceleration factor value multiplies the acceleration value programmed with the Load Acceleration (`load_accel`) function at low velocities and then tapers off with increasing velocity until the programmed acceleration value is reached as the motor reaches its target step rate.

The resulting acceleration profile resembles an enhanced high-acceleration/velocity profile for the first part of acceleration and a reduction in the acceleration/velocity profile for the second part of the acceleration profile.

For example:

Loaded Velocity = 2000 steps/s

Loaded Acceleration = 4000 steps/s^2

Acceleration Factor = 4

The motor starts its acceleration at: $4 \times 4000 = 16{,}000$ steps/s^2. As it passes through 1000 steps/s it is accelerating at the slower rate of: $2.5 \times 4000 = 10{,}000$ steps/s^2. When it reaches its target velocity of 2000 steps/s it is accelerating at a rate of:

$1 \times 4000 = 4000$ steps/s^2.

load_break_mod (Closed-Loop Stepper and Servo only)

Load Breakpoint Modulus

Format

status = load_break_mod (boardID, axis, breakmod)

Purpose

Programs a repeat period or modulus for absolute breakpoint values.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
breakmod	u32	breakpoint repeat period

Parameter Discussion

axis is the axis to be controlled.

breakmod is loaded as a quadrature count value and must be within the following range:

0 through 16,777,215 (2^{24}-1)

0 disables the repeat period functionally (default).

☞ **Note** ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

Using This Function

If a nonzero breakpoint repeat period value is loaded to an axis, all subsequent enable breakpoint-absolute functions to that axis will interpret the loaded position breakpoint value to be within the present breakpoint repeat period. The breakpoint is set and enabled with respect to the beginning of the breakpoint repeat period.

The breakpoint repeat period is used only when breakpoints are enabled as absolute breakpoints.

Upon power-up, the default value for breakpoint repeat period is zero for all axes. A zero value is specially interpreted and disables the repeat period functionality. Once a nonzero value is loaded, the board maintains this breakpoint repeat period value until a new value is

loaded or the board is reset. The breakpoint repeat period function on an axis can be disabled at any time by loading a zero value.

This function causes the loaded position breakpoint value to be added to the repeat position at the beginning of the current repeat period. When the present position is negative, the breakpoint period begins at the most negative value for the period.

Examples

Breakpoint Repeat Period = 2,000 quadrature counts

If the present position is 2,530

and the loaded breakpoint is 1,000

the breakpoint is enabled at 3,000

Breakpoint Repeat Period = 40,000 quadrature counts

If the present position is 50,000

and the loaded breakpoint is 2,500

the breakpoint is enabled at 42,500

Breakpoint Repeat Period = 2,000 quadrature counts

If the present position is 6,234

and the loaded breakpoint is –1,000

the breakpoint is enabled at 5,000

Breakpoint Repeat Period = 10, 000 quadrature counts

If the present position is –25,000

and the loaded breakpoint is 2,500

the breakpoint is enabled at –27,500

load_deriv_gain (Servo only)

Load Filter Derivative Gain

Format

status = load_deriv_gain (boardID, axis, gain)

Purpose

Sets the value of the derivative gain used by the digital filter servo control loop for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
gain	u16	derivative gain value

Parameter Discussion

axis is the axis to be controlled.

gain is the derivative gain value. Gain has the range of:

0 through 32,767

The default value is 3,500.

☞ **Note** ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

Using This Function

The derivative gain determines the contribution of a force proportional to the rate of change of position error. This force acts much like viscous damping in a damped spring and mass mechanical system (similar to a shock absorber). The digital filter computes the derivative every derivative sample period. This value, multiplied by the programmed derivative gain, is used by the PID loop every base sample period (341 ms) to upgrade the motor control output signal. This value may be changed at any time to tune the digital filter.

load_deriv_per (Servo only)

Load Filter Derivative Sample Period

Format

status = load_deriv_per (boardID, axis, period)

Purpose

Sets the value of the derivative sampling period of the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
period	u16	sample period selector

Parameter Discussion

axis is the axis to be controlled.

period is the sample period selector. The sample period range is:

0 through 255

The default value is 0.

☞ **Note** ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

Using This Function

Use this function to set the value of the derivative sampling period for the axis specified. The sample period selector word is used as a multiplier of the base sample period (341 ms). The derivative sampling period determines the update rate of the derivative term in the PID loop.

The derivative sample period may be updated at any time to tune the digital filter servo loop for system dependent motion control. Adjusting this value provides greater flexibility in tuning the derivative term of the PID loop. Determining the actual value to be used depends upon motion system configuration.

The actual derivative sampling period is related to the sample period selector word by the following formula:

$$\text{Derivative Sampling Period} = \frac{(\text{Sample Period Selector} + 1) \times (2048)}{(6{,}000{,}000)}$$

The sampling period for the default value of the sample period selector of 00 is:

$$\text{Derivative Sampling Period} = \frac{(0 + 1) \times (2048)}{(6{,}000{,}000)}$$

$$\text{Derivative Sampling Period} = 341 \text{ E-6 s or } 341 \text{ ms}$$

Since the maximum value for the sample period selector is 0xFF (function) or 255 (decimal), the maximum derivative sampling period is:

$$\text{Derivative Sampling Period} = \frac{(255 + 1) \times (2048)}{(6{,}000{,}000)}$$

$$\text{Derivative Sampling Period} = 87.4 \text{ E-3 s or } 87.4 \text{ ms}$$

The overall range of sample period selector values for the derivative sampling period is:

$0x00 \leq \text{Sample Period Selector Word} \leq 0xFF$

The corresponding derivative sampling period range is:

$341 \text{ E-6 s} \leq \text{Sampling Period} \leq 87.4 \text{ E-3 s}$

It is recommended that you start with a short period (Sample Period Selector = 00 or 01) and tune the value according to system requirements. Larger values are useful for higher inertia systems.

load_fol_err (Closed-Loop Stepper and Servo only)

Load Following Error

Format

status = load_fol_err (boardID, axis, errorcount)

Purpose

Sets the following error for a closed-loop system.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
errorcount	u32	following error count

Parameter Discussion

axis is the axis to be controlled.

- Servo

errorcount is a word (16-bit) value in the range of:

0 through 32,767

If values greater than the maximum values are programmed, then the function defaults to the maximum value and functions normally.

- Stepper

errorcount is a word (16-bit) value in the range of:

1 through 65,535

The default value for **errorcount** is 32,767 (0x7FFF).

☞ **Note** ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

This function requires the closed-loop version of the stepper board. Attempting to execute this function on the open-loop board generates an error. In addition, this function returns an error when the closed-loop stepper is operating in open-loop

mode. See the Set Loop Mode ***(***`set_loop_mode`***) function for more information. This function requires that the correct values for motor steps per revolution and encoder lines per revolution be entered with the*** Load Steps and Lines/Rev ***(***`load_steps_lines`***) function.***

Using This Function

Use this function to set the following error value that causes an automatic kill stop for the axis specified. The internal stop that is issued when the following error exceeds the value programmed is a kill stop. See Kill Motion (`kill_motion`) function for more information.

Use this function to help protect the motion hardware and associated system components when the position counter and the encoder counter disagree by more than this programmed following error value.

☞ **Note** ***You cannot turn off Following Error. Setting the following error count value to its maximum may approximate turning off the function.***

load_intg_gain (Servo only)

Load Filter Integral Gain

Format

status = load_intg_gain (boardID, axis, gain)

Purpose

Sets the value of the integral gain used by the digital filter servo control loop for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
gain	u16	integral gain value

Parameter Discussion

axis is the axis to be controlled.

gain is the integral gain value. **gain** has a range of:

0 through 32,767

The default value is 0.

☞ **Note** ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

Using This Function

The integral gain determines the contribution of a restoring force that grows with time and thus ensures that the static position error in the digital filter servo loop is zero. This restoring force works against constant torque loads to help achieve zero position error.

In applications with very small or nonexistent torque loads, this value may remain at its default value of zero (0). For systems having high torque loads, this value should be tuned to maintain zero position error. This value may be changed at any time to tune the digital filter servo control loop.

Each sample period, the servo loop position error is added to the accumulation of previous position errors to form the integration sum. You can scale the integration sum by dividing 256 by the integration sum then multiplying by the integral gain. Therefore, the resulting contribution to the motor control output is:

$$\frac{\text{Integration Sum}}{256} \times \text{Integral Gain}$$

☞ **Note** ***The scaling by 1/256 allows you to use the integer values for the integral gain when only a small amount of integral contribution is required.***

load_intg_lim (Servo only)

Load Filter Integration Limit

Format

status = load_intg_lim (boardID, axis, limit)

Purpose

Sets the integration limit used by the digital filter servo control loop for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
limit	u16	integration limit

Parameter Discussion

axis is the axis to be controlled.

limit is the integration limit. **limit** has the value of:

0 through 32,767

The default value is 0.

☞ **Note** ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

Using This Function

The integration limit is used to restrict the contribution of the integral term in the digital filter servo loop. The integration limit is compared to the scaled integration sum and the lessor of the two values is multiplied by the integral gain and then used in the generation of the motor control signal. This integration limit can be used to limit excessive restoring forces. This value may be programmed in conjunction with the integral gain to tune the digital filter servo loop for system dependent torque load conditions. This value may be changed at any time to the digital filter servo control loop.

Upon power-up, the integration limit function is disabled until this function is executed.

load_pos_brk (Closed-Loop Stepper and Servo only)

Load Position Breakpoint

Format

status = load_pos_brk (boardID, axis, breakpos)

Purpose

Loads a breakpoint for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
breakpos	i32	breakpoint position

Parameter Discussion

axis is the axis to be controlled.

breakpos must be in the following range:

–1,073,741,824 (-20^{30}) through 1,073,741,823 (20^{30}-1)

The default value is 0.

Using This Function

This function allows you to generate an external signal on a dedicated digital I/O line when the motion position equals the preprogrammed breakpoint position.

- Axis 1 uses the I/O port 1 (stepper) or 5 (servo) bit as its dedicated breakpoint output line.
- Axis 2 uses the I/O port 2 (stepper) or 6 (servo) bit as its dedicated breakpoint output line.
- Axis 3 uses the I/O port 3 (stepper) or 7 (servo) bit as its dedicated breakpoint output line.
- Axis 4 uses the I/O port 4 (stepper) or 8 (servo) bit as its dedicated breakpoint output line.

The Load Position Breakpoint function is used to load the desired step position count at which an external breakpoint signal is generated for each axis.

This function uses output lines in the digital I/O port that are dedicated on a per-axis basis. You must take care to configure the corresponding I/O port bits as outputs with the desired

polarity and initial output value. When the breakpoint position is reached for a properly set up axis and I/O port configuration, the output level on the digital I/O line changes state. If a particular transition is desired, the output should be preset prior to using the breakpoint function and reset following the breakpoint position execution. You can poll the read I/O port function to determine whether the breakpoint function has occurred.

The typical control and execution sequence for using this function is:

1. Setup and configure the I/O port bits as required.
2. Load the breakpoint position (`load_pos_brk`).
3. Enable the breakpoint function and set the breakpoint mode. The breakpoint position may be entered in absolute or relative mode (`enable_brk`).
4. The I/O port bit output value will change state (toggle) when the breakpoint position is reached.
5. (Optional) Reset the output value to a desired state (`set_io_output`).

You can change the breakpoint position loaded by this function at any time. The value loaded is double buffered and not actually used until you execute the Enable Breakpoint (`enable_brk`) function. To repeatedly use the value loaded by this function, execute the Enable Breakpoint (`enable_brk`) function after you have successfully executed a breakpoint position match. It is not necessary to reload the breakpoint position with this function for each use, if the breakpoint position remains the same.

☞ **Note** ***Absolute-position breakpoints uses the user-programmed scale and offset parameters and relative position breakpoints use the scaling parameters programmed. (See functions*** Load Position Reference Offset ***(***`load_pos_ref`***)***Load Position Scale Factor ***(***`load_pos_scale`***) and*** Set Scale Factor Sequence ***(***`set_scale_seq`***)).***

You can link the trigger input functionality by a software function (`enable_pos_trig`) to the breakpoint output bits. In this software link mode, a breakpoint output causes a valid trigger input on the corresponding trigger buffer, causing the trigger buffer stored functions (if any) to execute. You must enable this additional link function prior to use.

- Servo

  The value loaded by this function is the breakpoint position in quadrature counts.

- Stepper

  The value loaded by this function is the breakpoint position in Steps.

☞ **Note** ***This function requires the closed-loop version of the stepper board. Attempting to execute this function on the open-loop board generates a function error. In addition, the position breakpoint function will not operate when the closed-loop***

stepper board is in open-loop mode. See the Set Loop Mode ***(***`set_loop_mode`***) function for more information.***

load_pos_ref (Closed-Loop Stepper and Servo only)

Load Position Reference Offset

Format

status = load_pos_ref (boardID, axis, offset)

Purpose

Sets a user-selected offset for motion position and trajectory reference.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
offset	i32	offset value

Parameter Discussion

axis is the axis to be controlled.

offset must be within the following range:

–1,073,741,824 (-2^{30}) through 1,073,74 1,823 (2^{30}-1)

The default value is 0.

Using This Function

This function allows you to use a programmable offset for system specific applications to relocate the zero position. The board maintains this reference offset. The offset zero coincides with the absolute zero position.

You can use this function to set an offset zero position at a point other than the absolute zero position. Typically the absolute zero position is set at the home switch and/or index position and the offset zero position may be located elsewhere in the system.

The position reference offset is programmed independently for each axis. The offset value is applied to all absolute positions, including the absolute breakpoint position and read position values. The position reference offset does not affect relative position values. Anticipation time breakpoints function independent of the position reference offset value. You can use new position reference offset values at any time.

- Stepper

  The reference offset is loaded in steps.

☞ **Note** ***This function requires the closed-loop version of the stepper board. Attempting to execute this function on the open-loop board generates a function error. This function operates in both open and closed-loop modes on closed-loop stepper boards. See the*** Set Loop Mode ***(***`set_loop_mode`***) function for more information.***

load_pos_scale (Closed-Loop Stepper and Servo only)

Load Position Scale Factor

Format

status = load_pos_scale (boardID, axis, scalefactor)

Purpose

Loads a scale factor for the axis specified.

Parameters

Input

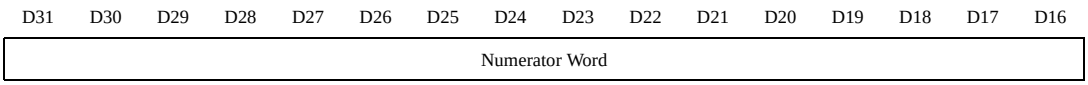
Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
scalefactor	u32	scale factor value

Parameter Discussion

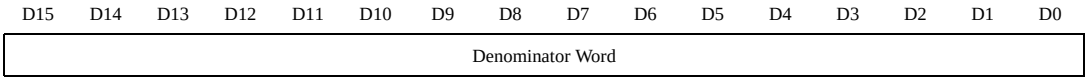
axis is the axis to be controlled.

scalefactor has the following format:

1 through 65,535: Numerator (upper 16-bits)



1 through 65,535: Denominator (lower 16-bits)



The default value is 1 for both the numerator and denominator.

Using This Function

This function, when used alone or in conjunction with the Set Scale Factor Sequence (`set_scale_seq`) function, loads a position scale factor for the axis specified. This scale factor is used to scale all position, breakpoint and read position values. The numerator and denominator value loaded by this function must be nonzero, positive integer values. The Set

Scale Factor Sequence (`set_scale_seq`) function controls the order of execution of data value scaling; you can either multiply then divide or divide then multiply.

The scale factor has the following format:

$$\text{Scale Factor} = \frac{\text{Numerator Word}}{\text{Denominator Word}}$$

If identical values are loaded for the numerator and denominator, then the value 1 is substituted for both values. If a zero is loaded for either value it is ignored and the existing value remains intact.

The board calculates all position values as 32-bit integers. Special care should be taken when choosing scale factor values, to ensure that overflows do not occur during the multiply portion of the scaling and that truncation of fractional results during the divide portion do not adversely affect resolution. In some cases it may be desirable to divide before multiplying. The Set Scale Factor Sequence (`set_scale_seq`) function allows you to set the order of operation. Please note that the mathematical precision may be reduced when you divide before multiplying. The correct selection of scaling factors allow you to maintain scaled position accuracy within one step (stepper) or one quadrature count (servo).

Scale factors may be set and changed at any time. Loaded Target Position (`load_target_pos`) Breakpoint Position (`load_pos_brk`) and Read Position (`read_pos`) values always reflect the most recent scale factor programmed.

- Stepper

☞ **Note** ***This function requires the closed-loop version of the stepper board. Attempting to execute this function on the open-loop board generates a function error. This function operates in both open and closed-loop modes on closed-loop stepper boards. See the*** Set Loop Mode ***(***`set_loop_mode`***) function for more information.***

load_prop_gain (Servo only)

Load Filter Proportional Gain

Format

status = load_prop_gain (boardID, axis, gain)

Purpose

Sets the value of the proportional gain used by the digital filter servo control loop for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
gain	u16	proportional gain

Parameter Discussion

axis is the axis to be controlled.

gain is the proportional gain. **gain** has the range of:

0 through 32,767

The default value for the proportional gain is 200 (0x00C8).

☞ **Note** ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

Using This Function

The proportional gain determines the contribution of a restoring force proportional to the position error. This restoring force functions in much the same way as a spring in a mechanical system. This value maybe changed at any time to tune the digital filter.

load_rot_counts (Closed-Loop Stepper and Servo only)

Load Rotary Counts

Format

status = load_rot_counts (boardID, axis, rotarycnts)

Purpose

Programs the total number of quadrature counts per revolution or modulo for a Rotary axis.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
rotarycnts	u32	rotary counts

Parameter Discussion

axis is the axis to be controlled.

rotarycnts is loaded as a quadrature count value and must be within the following range:

0 through 16,777,215 (2^{24}-1)

The default value is 0.

Using This Function

If a nonzero rotary count value is loaded to an axis, all subsequent Load Target Position functions to that axis are interpreted modulo the loaded rotary counts.

The axis will move to the function position by taking the shortest path, either forward or backwards, while remaining within the one revolution defined by the loaded rotary counts. Similarly, the value read with the Read Present Position function is a positive value that is modulo the loaded rotary count value.

A zero value is specially interpreted and disables the rotary axis functionality. Once a nonzero value is loaded, the board maintains the rotary count value until a new value is loaded or the board is reset. The rotary modulo function can be disabled for any axis, at any time by loading a zero value.

This function can be used in conjunction with the user-programmed scaling and offset parameters. (See Load Position Reference Offset (`load_pos_ref`) Load Position Scale Factor (`load_pos_scale`) and Set Scale Factor Sequence (`set_scale_seq`) functions. If scaling and offset parameters are used, a loaded target position value is first scaled and then offset prior to being calculated modulo and finally loaded to the counter circuits on the board.

Example 1

Position Mode–Absolute

Rotary Counts = 2,000 quadrature counts for a full rotation of the axis system.

If the present position is: 930

and the target position is: 3,600

then the actual move is: +670

and the final position is: 1,600

Example 2

Position Mode–Relative

Rotary Counts = 40,000 quadrature counts for a full rotation of the axis system

If the present position is: 15,000

and the target position is: –30,000 (relative)

then the actual move is: +10,000 (relative) in the forward direction

and the final position is: 25,000

load_rpm

Load RPM

Format

status = load_rpm (boardID, axis, rpm)

Purpose

Loads velocity in Revolutions Per Minute (RPM).

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
rpm	f64	RPM value

Parameter Discussion

axis is the axis to be controlled.

rpm is a velocity value in RPM.

Using This Function

This function takes an input value in RPM and converts this value to either steps/s for stepper motors or quadrature encoder counts per sample period for servo motors.

☞ **Note** ***The maximum velocity value supported by a given board will not be exceeded.***

For DOS users, this functions's prototype is: status = `load_rpm` ***(u16 boardaddr, u16 boardtype, u8 axis, u16 revdata, f64 rpm). Refer to*** *Special Considerations for DOS Users* ***earlier in this chapter for more information.***

- Servo

You must assign the encoder lines per revolution before issuing this function Store Encoder Line Count (`store_elc`).

- Stepper

  You must assign the steps per revolution before issuing this function Store Steps Per Rev (`store_steps_rev`) or Load Steps and Lines/Rev (`load_steps_lines`).

load_rpsps

Load Revolutions Per Second Per Second

Format

status = load_rpsps (boardID, axis, rpsps)

Purpose

Loads acceleration in Revolutions Per Second Per Second (RPSPS).

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
rpsps	f64	Revolutions Per Second Per Second value

Parameter Discussion

axis is the axis to be controlled.

rpsps is an acceleration value in RPSPS.

Using This Function

This function takes an input value in RPSPS and converts this value to be loaded as an acceleration value to the board.

☞ **Notes** ***This maximum acceleration value supported by a given board will not be exceeded.***

For DOS users, this functions's prototype is: status = `load_rpsps` ***(u16 boardaddr, u16 boardtype, u8 axis, u16 revdata, f64 rpsps). Refer to*** *Special Considerations for DOS Users* ***earlier in this chapter for more information.***

- Servo

For servo boards, the converted value is loaded in counts/sample period/sample period. You must assign the encoder lines per revolution before issuing this function Store Encoder Line Count (`store_elc`).

- Stepper

  For stepper boards, the RPSPS value is converted to steps/sec/sec. You must assign the steps per revolution before issuing this function Store Steps Per Rev (`store_steps_rev`) or Load Steps and Lines/Rev (`load_steps_lines`).

load_steps_lines (Closed-Loop Stepper only)

Load Steps and Lines/Rev

Format

status = load_steps_lines (boardID, axis, stepslines)

Purpose

Sets the steps/revolution and lines/revolution for a closed-loop stepper board.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
stepslines	u32	holds lines per revolution and steps per revolution

Parameter Discussion

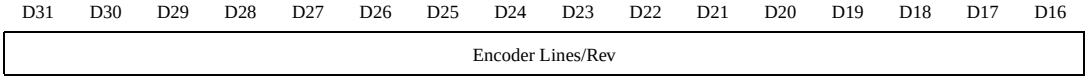
axis is the axis to be controlled.

stepslines holds the value of encoder lines/rev in its upper 16 bits and steps/rev in its lower 16 bits.

The value for encoder Lines/Rev is an unsigned word (16-bit) in the range of:

1 through 16,383

Encoder lines/rev on quadrature encoders is one-fourth the number of encoder counts/rev. For example, if you read the encoder count (`read_encoder`), then turn the motor one revolution, and read the encoder count again, and the difference in values is 200, you should enter a value of 50 for encoder lines/rev.



The default value for the Lines/Rev Data Word is 200 lines per revolution.

The value for Steps/Rev is an unsigned word (16-bit) in the range of:

1 through 32,767

If your motor is being controlled in a half-step or microstepping manner, you must consider the microstepping factor. For example, for a motor that has 200 steps per revolution that is being microstepped by a 10 to 1 factor, you should enter a value of 2,000.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Steps/Rev															

The default value for the Steps/Rev Data Word is 200 steps per revolution.

Note ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and functions normally.***

Using This Function

This function is used to specify the number of motor steps per revolution, based upon the motor/driver configuration and encoder lines per revolution, for the components used with the axis specified. This information is necessary for the stepper board to function correctly in closed-loop mode. This function should be sent to the board prior to executing any closed-loop functions.

For nonrotary systems using linear motors and/or linear encoders, steps and lines per unit of measure (i.e., steps per inch and lines per inch or steps per cm and lines per cm, etc.) should be entered in place of steps per rev and lines per rev.

Note ***This function requires the closed-loop version of the stepper board. Attempting to execute this function on the open-loop board generates a function error.***

Caution ***The values for motor steps per revolution and encoder lines per revolution must be accurately loaded for the board to function correctly in closed-loop mode. Incorrect values may result in apparently erratic operation of the board.***

load_target_pos

Load Target Position

Format

status = load_target_pos (boardID, axis, position)

Purpose

Loads the desired target position value.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
position	i32	target position for axis

Parameter Discussion

axis is the axis to be controlled.

position is a 32-bit value representing either an absolute or a relative position to be moved to.

Using This Function

The function of the loaded value varies with the position mode selected. For example, in absolute position mode, the loaded value will be the absolute target position for a subsequent Start Motion function. Motion proceeds to the loaded absolute position when a start is issued. However, in relative position mode, the loaded value will be the move distance relative to the last stopped position. Motion proceeds to the relative position when a start is issued. The position mode must be set prior to the loading of a position value for the value to be interpreted in the desired mode.

Position Mode Target Examples

Position Mode–Absolute:

If the present position is: 10,000

and the target position is: 40,000

then, the actual move will be in the forward direction: 30,000

The final position at move complete (same as target position): 40,000

Position Mode–Relative:

If the present position is: 10,000

and the target position value is: –10,000

then, the actual move will be in the reverse direction: 10,000

The final position at move complete (previous position plus relative position): 0

Position values are signed 32-bit values (long word) and indicate a desired position and direction of motion with respect to the zero position (absolute) or present position (relative). In order to specify direction of motion, the loaded target position must be specified as a positive value for forward motion with relation to the zero position or present position and as a negative value for reverse motion with relation to the zero position or present position.

When a new target position is loaded, it is buffered in the axis specific control circuit as the new target position value until a Start Motion function is issued for that axis. New values loaded constantly overwrite old values until a Start Motion function is executed for that axis.

Each time a new position value, velocity value, acceleration value, or acceleration factor is loaded, a new trajectory calculation is enabled on the board. You can accomplished trajectory calculation autonomously when you execute the Start Motion (`start_motion`) function. Trajectory calculation can take from 1 to 100 ms to complete, prior to motion starting.

- Stepper

  The value loaded by this function is the target position in step counts. In open-loop mode, the correct number of steps is generated to move to the desired target position. In closed-loop mode, if a proper value for step and lines per revolution has been loaded (see Load Steps and Lines/Rev (`load_steps_lines`) then the axis moves to the desired position using the encoder counter feedback to verify both the target and actual positions. With the proper value for steps/lines, a closed-loop move operation moves the axis to the desired position in steps.

  When velocity mode is selected, this function must be used to set the desired direction of motion for constant velocity.

  Velocity Mode: (This function sets direction.):

  1 = forward (or any positive position)

  –1 = reverse (or any negative position)

  Send a Start Motion function to begin velocity mode motion in the desired direction.

  The maximum number of steps for any single move is: $\pm 2^{21} = 2{,}097{,}152$

  In relative mode, this limits the input position value for a single move to:

  –2,097,152 (-2^{21}) through 2,097,152 (2^{21})

In absolute mode, this limit applies to the distance between the starting position and target position of the move. The total usable absolute position range is much larger, however. During multiple move sequences, the absolute position can be anywhere within the following boundaries:

–4,294,967,296 (-2^{32}) through 4,294,967,295 (2^{32}-1)

☞ **Note** ***All position values are loaded in steps. The loaded position values are affected by user-programmed scale and offset parameters, if required. See functions*** Load Position Reference Offset ***(***`load_pos_ref`***),*** Load Position Scale Factor ***(***`load_pos_scale`***), and*** Set Scale Factor Sequence ***(***`set_scale_seq`***).***

- Servo

The maximum position values loaded must remain within the following boundaries:

–1,073,741,824 (-2^{30}) through 1,073,741,823 (2^{30}-1)

☞ **Note** ***All position values must be loaded in quadrature encoder counts. The loaded position values are affected by user-programmed scaling and offset parameters. See functions*** Load Position Reference Offset ***(***`load_pos_ref`***),*** Load Position Scale Factor ***(***`load_pos_scale`***), and*** Set Scale Factor Sequence ***(***`set_scale_seq`***).***

A forward move (relative) of 20,000 counts would be sent as: 20,000.

A reverse move (relative) of 20,000 counts would be sent as: –20,000.

load_time_brk (Servo only)

Load Anticipation Time Breakpoint

Format

status = load_time_brk (boardID, axis, time)

Purpose

Generates an external signal on a dedicated digital I/O line when the axis position equals the preprogrammed anticipation time breakpoint position.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
time	u32	anticipation time in sample periods

Parameter Discussion

axis is the axis to be controlled.

time is anticipation time in sample periods relative to the end of a programmed position trajectory in position mode. Time has a range of:

1 through 65,535 + (2^{16}-1)

☞ **Note** ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

Using This Function

The anticipation breakpoint function allows you to generate an external signal on a dedicated digital I/O line when the axis position equals the preprogrammed anticipation time breakpoint equivalent position.

- Axis 1 uses the I/O port 5 bit as its dedicated breakpoint output line.
- Axis 2 uses the I/O port 6 bit as its dedicated breakpoint output line.
- Axis 3 uses the I/O port 7 bit as its dedicated breakpoint output line.
- Axis 4 uses the I/O port 8 bit as its dedicated breakpoint output line.

The Load Anticipation Time Breakpoint function is used to preload a desired anticipation time relative to the end of a programmed motion trajectory in position mode. When the position corresponding to the desired anticipation time is reached, an external breakpoint signal transition occurs on the dedicated I/O port output line for the axis selected. The anticipation breakpoint function is similar to, and shares the same resources as, the position breakpoint function. Because they share the same resources, these two breakpoint configuration functions are mutually exclusive and the last one issued is the one that is used by the Enable Breakpoint (`enable_brk`) function.

The anticipation time is the time (measured in sample periods) prior to the completion of the preprogrammed motion trajectory. The value is converted by the servo board from a number of sample periods (Sample Periods = 341 ms) to the corresponding position during the move. The anticipation time breakpoint function causes a change of state on the I/O port output bit for the axis selected if executed successfully.

Example of the Anticipation Breakpoint Function

An I/O port output signal transition is required as an anticipation breakpoint indicator when the motion trajectory is 30 ms from its completion point. The anticipation time multiplier closest to 30 ms is determined by dividing 341 µs into 30 ms. This results in an anticipation time multiplier of 0x0058 (function) or 88 decimal. The multiplier value is sent as the data word for this function. Prior to enabling the breakpoint function with the Enable Breakpoint (`enable_brk`) function, all of the motion trajectory parameters must be loaded for the move that triggers the breakpoint. When the programmed motion reaches the position corresponding to the desired anticipation time, the corresponding I/O port output bit changes state.

This function only functions with motion trajectory profiles loaded in position mode. The anticipation breakpoint function operates with respect to the end of the move regardless of the operation mode selected (absolute or relative).

It is recommended you complete the following sequence when using this function. (This assumes a properly initialized and configured servo board.)

☞ **Note** ***Be sure that the I/O port bit corresponding to the axis selected has been properly configured as an output port.***

1. Load Trajectory
2. Load Anticipation Breakpoint
3. Enable Breakpoint Function
4. Start Motion

An output transition occurs on the I/O port bit at the programmed anticipation time. Each occurrence of a breakpoint changes the state of the output signal level (toggle). If you desire a particular output level transition then you must preset the output bit.

To repeat the anticipation breakpoint function using the same anticipation time, it is not necessary to reload the anticipation breakpoint time. You only have to re-enable the breakpoint function. The sequence is:

1. Load Trajectory
2. Enable Breakpoint Function
3. Start Motion

Note ***If you attempt to program an anticipation time that would occur prior to the start of motion, then the function defaults to a breakpoint 20 quadrature counts after motion start.***

In order to maintain the greatest accuracy in calculating the anticipation time breakpoint, be careful when you select trajectory parameters. The values for acceleration and velocity must remain within the following ranges when using this function:

Velocity ≤ 0x007F FFFF (8388607)

Acceleration ≤ 0x0000 FFFF (65535)

load_vel

Load Velocity

Format

status = load_vel (boardID, axis, velocity)

Purpose

Loads the desired velocity for the axis specified. If you are using a servo board, it is recommended that you use the Load RPM (`load_rpm`) function.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
velocity	u32	velocity value

Parameter Discussion

axis is the axis to be controlled.

velocity is the velocity value loaded for the specified axis. For velocity ranges, refer to the following *servo* and *stepper* sections.

- Servo

  This function is used to load the desired velocity on a per-axis basis. The value loaded by this function is the velocity in quadrature encoder counts per sample period. Use the formula in the servo section of *Using This Function* to calculate the velocity value to load.

  In all cases, the maximum velocity values loaded must never exceed the following boundaries:

  0 through 16,777,215

  The default value for velocity at power-up is 32,768 (0x0000 8000).

- Stepper

  This function is used to load the desired velocity on a per-axis basis. The value loaded by this function is the maximum trajectory or constant velocity in steps per second.

The loaded velocity value must be within the following range:

1 through 1,048,575

The default value of steps/s at power-up is: 3.

Note ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

Using This Function

The velocity value loaded is used differently in the various operation modes (See Set Operation Mode (`set_pos_mode`)). For example, in position mode, the loaded velocity value will be the maximum velocity allowed during the trapezoidal move to the target position. In velocity mode, the loaded velocity value is the functioned velocity issued at the execution of a Start Motion function. Motion continues at the last function velocity until a new velocity is loaded and updated by a Start Motion function or a Stop Motion function is executed.

Velocity Mode Example

If the present functioned velocity is: 10,000

and the new loaded velocity is: 20,000

and a Start Motion function is issued,

then, the actual velocity will be (assuming proper system set up): 20,000

Velocity values loaded are 32-bit (long word) values and indicate a desired motion velocity. Actual velocity attained is determined by many factors not limited to programmed motion control parameters. Filter parameters as well as target acceleration values have an affect upon a system's ability to attain a desired velocity. The major limiting factor on motion velocity is the mechanical system configuration and motor/amplifier constraints.

Caution ***Do not program velocity values that exceed the safe operating range for the electrical, mechanical, and motion components involved.***

When a new velocity value is loaded, it is buffered in the axis specific servo control circuit as the new velocity value until a Start Motion (`start_motion`) function is issued for that axis. New values loaded constantly overwrite old values until the Start Motion function or Stop Motion (`stop_motion`) function is executed.

- Servo

Note ***The Load RPM (`load_rpm`) function is recommended because of its ease-of-use.***

The velocity value to be loaded as a long word can be calculated using the following simple formula:

Velocity Value Long Word (32 bits) = (R) × (Ts) × (Ct) × (Vrpm) × (65,536)

Where:

R:	Quadrature Encoder Counts Per Revolution
Ts:	341 E-6 (Filter Sample Period)
Ct:	1/60 (Conversion Factor–Minutes to Seconds)
Vrpm:	Desired Velocity in RPM
65,536:	Integer/Fraction Scaling Factor

☞ **Note** ***This formula can also be applied to linear motors by changing the (R) value from Quadrature Counts Per Revolution to Quadrature Counts Per Index Period or Counts Per Unit Measure (in, mm, etc.) and by changing the (Vrpm) value to a (Vipm) value for desired velocity in Indexes Per Minute. (If you use Indexes Per Second, then the (Ct) value is no longer necessary.)***

Velocity Value Determination Example

Encoder = 1,000 Lines Per Revolution

Thus: R = 4 x 1,000 = 4,000 Quadrature Counts

Filter Sample Period = 341 E-6 (Seconds)

Thus: Ts = 341 E-6

Desired Velocity (Vrpm) = 2,250 RPM

Velocity Value To Load = (4,000) × (341 E-6) × (1/60) × (2,250) × (65,536)

Velocity Value To Load = (51.15 Counts Per Sample Period) × (65,536)

Velocity Value To Load = 3,352,166 (Scaled and Rounded)

load_vel_change (Stepper only)

Incremental Velocity Change

Format

status = load_vel_change (boardID, axis, adjvelby)

Purpose

Changes the current velocity for a moving axis by the specified amount.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
adjvelby	i16	velocity change value

Parameter Discussion

axis is the axis to be controlled.

adjvelby is the incremental value used to adjust the current velocity.

Range: –20,000 steps/s to 20,000 steps/s.

☞ **Note** ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

Using This Function

This function allows you to change velocity by a loaded increment while a move is in progress. Although this function may be used in any mode, during any move, it is primarily designed for use in velocity mode, where incremental changes to velocity may be needed during constant velocity motion. This function does not execute and is ignored by the board if it is sent during acceleration, deceleration, or when motion is stopped.

When this function is executed, the incremental velocity change value is added to or subtracted from the present target velocity programmed with the Load Steps/sec (`load_vel`). The velocity changes immediately and without any acceleration or deceleration ramping.

The incremental velocity change is not cumulative, therefore, subsequent incremental changes replace previous changes. The incremental velocity change is also used for the present move, and is not stored for subsequent moves. When a move completes or is stopped, the incremental velocity change is lost and the value is cleared.

master_slave_cfg (Servo only)

Master Slave Configure

Format

status = master_slave_cfg (boardID, axis, mode)

Purpose

Configures an axis to be a master/slave.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
mode	u16	master/slave mode

Parameter Discussion

axis is the axis to be controlled.

mode is the master/slave mode to be set for this axis:

0	Normal Axis
0x00FF	Normal Slave
0xFF00	Reverse Slave
0xFFFF	Master

Using This Function

When setting up a master/slave configuration, this function is called twice. Once to specify a slave and again to specify a master.

☞ **Note** ***This function is valid only on 2- and 4-axis servo boards.***

This function is used to configure an axis for master-slave operation. This function should be issued as part of a system initialization/configuration routine. Since all motion trajectory execution depends upon the master-slave mode, you should execute this function before you load the motion trajectory parameter.

Only one master axis is allowed per board. The last axis designated as a master is the one used. Slave axes can be either normal or reverse where reverse slaves rotate opposite to the master axis.

The master axis operates identical to a normal axis. It can be controlled in any operation mode and all functions operate normally including scale factors and offset, rotary counts, breakpoints, start and stop, reset position, etc. The absolute position of the master axis encoder is the source of the target position for the slave axes. The master axis does not have to be serving or even controlling a motor to function as a master.

Axes configured as slaves function in a special mode that is quite different from normal axis operation. The gear ratio for the slave is set by its position scale factor (see Load Position Scale Factor (`load_pos_scale`) and Set Scale Factor Sequence (`set_scale_seq`)). A scale factor of 3/2 (numerator = 3 and denominator = 2) results in the slave axis rotating three revolutions for every two of the master. Different gear ratios can be set by way of its function for each slave axis. When sent to a slave, the Load Position Reference Offset (`load_pos_ref`) function causes the slave to make an offset move in addition to the position defined by the master and the gear ratio.

Slaves always function in position mode by default. Any attempt to set a slave axis to velocity mode generates a function error. Slaves can be operated in either absolute or relative position mode, however. The operation mode determines how the slaves react to the position function coming from the master. In absolute mode, the master and slave are geared relative to absolute zero position. In relative mode, the master and slave are geared relative to the position of the master (and slave) when the mode is switched from absolute to relative and/or when the gear ratio is changed while in relative mode. Relative mode allows the gear ratio of the slave to be changed *on the fly.* The slave can always be brought back into absolute gearing by switching back to absolute mode.

To guarantee synchronization, a Reset Position (`reset_pos`) function sent to the master axis resets all slave axes as well as the master.

A slave axis uses the programmed values of velocity and acceleration as well as its PID parameters to control the dynamics of the slave following and offset moves. In most application, the velocity and acceleration parameters should be set high enough to avoid limiting the slaves ability to follow the master.

If a slave axis is stopped for any reason other than the fact that the master has stopped moving, it must be restarted with a Start Motion (`start_motion`) function. This safety interlock functions when the slave hits an enabled limit or home switch or trips out on following error.

☞ **Note** ***After configuring a slave axis, a*** Start Motion ***function is required to start the*** Master-Slave ***function.***

multi_start

Multi-Axis Start

Format

status = multi_start (boardID, startdata)

Purpose

This function allows you to start multiple axes simultaneously.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
startdata	u16	bitmap of axes to start

Parameter Discussion

axis is the axis to be controlled.

startdata has the following format:

D15	D14	D13	D12	D11	D10	D9	D8
0	0	0	0	0	0	0	0

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	Axis 4	Axis 3	Axis 2	Axis 1

1 = Start Axis
0 = Do Not Start Axis

Using This Function

This function is used to issue a simultaneous start to the axes indicated in the **startdata**. If the normal starting conditions are met for all of the axes in this function, then a start function is issued to all axes specified. If any of the specified axes fails the starting condition test then this function is terminated and no starts are issued on any axis.

This function cannot be used to start a single axis. Attempting to execute a Multi-Axis Start with less then two axes selected generates a function error.

If one or more of the selected axes cannot be started due to a limit switch condition, the function Not Executed status bit will be set in the Communications Status Register and in the Per-Axis Hardware Status for each axis that cannot be started. See Read Per-Axis Hardware Status (`read_axis_stat`) and Read Communications Status (`read_csr`) functions for more details.

☞ **Note** ***The start function to each axis occurs simultaneously within 100 µs.***

read_adc (Closed-Loop Stepper and Servo only)

Read A/D Converter Analog Input Value

Format

status = read_adc (boardID, channel, adcval)

Purpose

Reads the voltage on the specified A/D channel and returns it as an 8-bit binary value.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
channel	u16	A/D channel number

Output

Name	Type	Description
adcval	u16	A/D value

Parameter Discussion

channel is the A/D channel number.

Range: 1 to 8

adcval is the 8 Bit digital conversion value of the analog input signal measured on the selected input channel. Values range from 0 to 255.

☞ **Note** ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

Using This Function

The range of voltages allowed for each input is 0 through 5V and an external reference may be used to further control the measured analog signal value. For a range of 0 through 5 V, the 8 Bit A/D converter resolves to 256 distinct values with each LSB value equal to 19.5 mV/LSB.

Refer to your *ValueMotion Hardware User Manual* for more information on setting up and connecting to the A/D Converter input circuitry.

☞ **Note** ***This function is valid on all 2- and 4-axis stepper boards and only on the PC-Servo-4A and PC-Servo-2A servo boards.***

read_axis_stat
read_axis_stat_rdb

Read Per-Axis H/W Status

Format

status = read_axis_stat (boardID, axis, axisstatus)
status = read_axis_stat_rdb (boardID, axis)

Purpose

Reads the detailed hardware level status for each axis controller circuit.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled

Output

Name	Type	Description
axisstatus	u16	bitmap of hardware information for axis

Parameter Discussion

axis is the axis to be controlled.

axisstatus has the following format:

D15	D14	D13	D12	D11	D10	D9	D8
0	0	function Execution Status	Present Direction (Vel Mode Only)	Operation Mode	Profile Complete	0	0

D13: 1 = Not Executed 0 = Executed Successfully
D12: 1 = Forward 0 = Reverse
D11: 1 = Velocity Mode 0 = Position Mode
D10: 1 = True 0 = False

D7	D6	D5	D4	D3	D2	D1	D0
Run Status	0	Position Err Status	Wrap around Error Cnt Status	Index Status	Motor Status	Home Switch Status	Limit Switch Status

D7: 1 = Running 0 = Stopped
D5: 1 = Position Error 0 = OK
D4: 1 = Wraparound Error 0 = OK
D3: 1 = Index Found 0 = Index Not Found
D2: 1 = Off 0 = On
D1: 1 = True 0 = False
D0: 1 = True 0 = False

Using this Function

The function execution status bit indicates that the last function to this axis was received but was determined to be illegal or inappropriate due to the operating condition of the axis. The following is a list of illegal conditions that will set this bit:

- Attempting to execute a set direction function in position mode.
- Attempting to execute a load target position function in velocity mode.
- Attempting to execute a find home function when
  - already executing a find home sequence.
  - per-axis limits and home are not enabled.
  - any trigger inputs are enabled.
  - both forward and reverse limits are active.
- Attempting to execute a Find Index function when
  - the axis is running.
  - any trigger inputs are enabled.
  - any limits are active. (home active is OK.)
- Attempting to execute a start function or Multi-Axis Start function and the desired direction of motion of any axis is blocked by a limit condition.
- Attempting to execute an Enable Breakpoint function for a loaded time breakpoint, and
  - not in position mode
  - the acceleration value is too high
  - the move distance is < 32 quadrature counts

- Attempting to execute an Enable Breakpoint function for a loaded position breakpoint, and
  - the breakpoint value is equal to the present position for an absolute breakpoint
  - the breakpoint value is equal to zero for a relative breakpoint
- During a find home sequence on any axis, attempting to execute any function other than the following:
  - Kill Motion (`kill_motion`)
  - Stop Motion (`stop_motion`)
  - Set Stop Mode (`set_stop_mode`)
  - Read Per-Axis Hardware Status (`read_axis_stat`)
  - Read Position (`read_pos`)
  - Read Velocity (`read_vel`)
  - Set I/O Port Output (`set_io_output`)
  - Read I/O Port (`read_io_port`)
  - Read Communications Status (`read_csr`)
  - Set I/O Port Polarity and Direction (`set_io_pol`)
  - Set Limit Switch Input Polarity (`set_lim_stat`)
  - Read Limit Switch Status (`read_lim_stat`)
  - Read A/D Analog Input (`read_adc`)
  - Read 24 Bit I/O Digital Inputs (`read_gpio`)
  - Set 24 Bit I/O Digital Outputs (`set_gpio`)
  - 24 Bit Auxiliary Digital I/O functions
  - A to D Input functions

axisstatus contains important hardware status information about each axis which may be used for many reporting and control functions.

☞ **Note** ***When using*** `read_axis_stat_rdb`**, axisstatus** ***is stored in the Return Data Buffer. Use*** Communicate ***(***`communicate`***) in mode 2 to retrieve this data.***

read_csr

Read Communication Status Register

Format

status = read_csr (boardID, csrdata)

Purpose

Returns the data from the Communications Status Register.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility

Output

Name	Type	Description
csrdata	u16	communications status register copy

Parameter Discussion

csrdata has the following format:

D15	D14	D13	D12	D11	D10	D9	D8
Axis 4	Axis 3	Axis 2	Axis 1	Axis 4	Axis 3	Axis 2	Axis 1

D15 - D12 1 = Move Complete 0 = Move In Process
D11 - D8 1 = Motor Running 0 = Motor Stopped

D7	D6	D5	D4	D3	D2	D1	D0
Last Cmd Status	Hardware Status	Pwr Offset/ Watchdog	Function Err Status	Function Status	Not Used	Data Pending	Ready for Next Cmd

D7: 1 = Executed 0 = Not Executed
D6: 1 = Failure 0 = OK
D5: 1 = True 0 = False (Cal Reset Position (`reset_pos`) functions to clear.)
D4: 1 = Function Error 0 = OK
D3: 1 = Function in Process 0 = Function Complete
D1: 1 = Data in RDB 0 = RDB Empty
D0: 1 = Ready 0 = Busy

Using This Function

The move complete status bits indicate that the axis has either reached its target position or has stopped trying to move due to an error. This information is similar to and derived from the profile complete status bits available through the Read Per-Axis Hardware Status (`read_axis_stat`) function.

The last function not executed status bit indicates that the function was received successfully but could not be executed because the board was performing some action that temporarily made the function invalid. For example, issuing a Multi-Axis Start while one axis cannot start due to a limit condition generates the function not executed status. The last function not executed bit available through the Read Per-Axis Hardware Status (`read_axis_stat`) function can be used to further isolate the problem to a specific axis.

The function-in-process status bit indicates that the last function sent is in the process of being executed. Any function errors are guaranteed to be set by the time the function is complete.

read_encoder
read_encoder_rdb (Closed-Loop Stepper only)

Read Encoder

Format

status = read_encoder (boardID, axis, encoderval)
status = read_encoder_rdb (boardID, axis)

Purpose

Reads the value of the encoder position counter for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled

Output

Name	Type	Description
encoderval	i32	encoder value

Parameter Discussion

axis is the axis to be controlled.

encoderval is a signed long word value (32 bits) in the range of:

–2,147,483,648 (-2^{31}) through 2,147,483,647 (2^{31}-1)

Using This Function



Caution ***This function requires the closed-loop version of the stepper board. Attempting to execute this function on an open-loop board generates a function error.***

This function can be executed while the closed-loop board is operating in open-loop mode to read and fully use the encoder readback capability independent of the per-axis function. See the Set Loop Mode (`set_loop_mode`) function.

The value read is the encoder count at the moment of function execution. This function may be executed at any time to determine the value of the encoder counter for the axis in question. The value returned is a signed hexadecimal long word value (32 bits).

The value returned is an instantaneous read of the encoder counter in the control loop for the specified axis. If motion is in process when this function is executed, the encoder position count will be changing and the value will only reflect the count at the moment it was read.

All encoder position values are in unscaled quadrature encoder counts. This function is not affected by the values of Lines/Rev loaded by the Load Steps and Lines/Rev (`load_steps_lines`) function. It is also not affected by the values of user programmed scale and offset parameters Load Position Reference Offset (`load_pos_ref`), Load Position Scale Factor (`load_pos_scale`), and Set Scale Factor Sequence (`set_scale_seg`).



Note ***When using*** `read_encoder_rdb`**, encoderval** ***is stored in the Return Data Buffer. Use*** Communicate ***(***`communicate`***) in mode 2 to retrieve this data.***

read_gpio
read_gpio_rdb (Stepper only)

Read Aux Digital I/O Input Values

Format

status = read_gpio (boardID, iodata)
status = read_gpio_rdb (boardID)

Purpose

Reads the 24-bit digital lines on ports A, B, and C and returns the direction indicator status for port C.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility

Output

Name	Type	Description
iodata	u32	readback data

Parameter Discussion

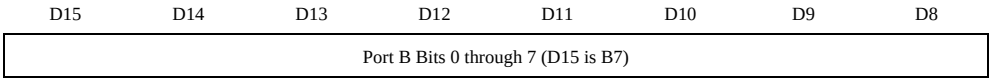
iodata is 32-bit data represented by 4 independent byte segments.

D31	D30	D29	D28	D27	D26	D25	D24
Port C Direction							

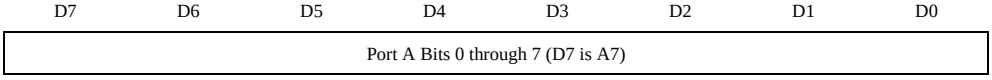
0xFF = Inputs
0x00 = Outputs

D23	D22	D21	D20	D19	D18	D17	D16
Port C bits 0 through 7 (D23 is C7)							

1 = High Level Signal Input/Output Bit
0 = Low Level Signal Input/Output Bit



1 = High Level Signal Output Bit
0 = Low Level Signal Output Bit



1 = High-Level Signal Input Bit
0 = Low-Level Signal Input Bit

iodata shows user-programmed (or default) output values for port B at all times, and for port C when it is properly configured as an output port.

Using This Function

This function will read back a 32-bit value, consisting of four (4) byte wide data values corresponding to the 3 bytes (24-bits) of auxiliary digital I/O and one (1) data direction selector byte. The 24-bit auxiliary digital I/O port is comprised of three distinct I/O groups of 8 bits (1 byte) each.

The three distinct I/O groups include, a dedicated input port (8 bits), a dedicated output port (8 bits), and a user-selectable I/O port (8-bits). The direction of this user-selectable port must be set by the user in both of the following ways:

1. The user must set the DIP switch or jumper on the board to configure the hardware for input or output operation. See Chapter 3, *Hardware Overview*, in your *ValueMotion Hardware User Manual* for more information on setting this DIP switch.
2. The user must also call Set Port C Direction (`set_portc_dir`) to indicate the selected I/O direction for the user selectable I/O port for proper configuration of the output and readback circuits.

Caution ***Failure to complete both of the above steps causes erroneous operation and incorrect readback values for the User-Selectable I/O port byte.***

Notes ***I/O port bit polarity set by the*** Set Auxiliary Digital I/O Values ***(***`set_gpio`***) function affect the values read by this function.***

When using `read_gpio_rdb`**, iodata** ***is stored in the Return Data Buffer. Use*** Communicate ***(***`communicate`***) in mode 2 to retrieve this data.***

read_io_port
read_io_port_rdb

Read I/O Port

Format

status = read_io_port (boardID, iodata)
status = read_io_port (boardID)

Purpose

Returns the logical values for all input and output bits.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility

Output

Name	Type	Description
iodata	u16	logical values of I/O lines

Parameter Discussion

The upper byte of **iodata** is always zero. The lower byte contains the value of the individual I/O port bits and has the following binary format.

D15	D14	D13	D12	D11	D10	D9	D8
0	0	0	0	0	0	0	0

D7	D6	D5	D4	D3	D2	D1	D0
I/O Bit 8	I/O Bit 7	I/O Bit 6	I/O Bit 5	I/O Bit 4	I/O Bit 3	I/O Bit 2	I/O Bit 1

1 = True 0 = False

☞ **Note** ***For stepper boards, bits D4 through D7 read the values of the inhibit lines.***

- Stepper

  Bits marked as inhibit output bit, have a true output when a Kill Motion function is issued or if a following error trip condition occurs in closed-loop control. The inhibit may be used with an amplifier/driver unit to disable the motor/drive.

Using This Function

☞ **Note** ***When using*** `read_io_port_rdb`***,*** **iodata** ***is stored in the Return Data Buffer. Use*** Communicate ***(***`communicate`***) in mode 2 to retrieve this data.***

read_lim_stat
read_lim_stat_rdb

Read Limit Switch Status

Format

status = read_lim_stat (boardID, switchstat)
status = read_lim_stat_rdb (boardID)

Purpose

Reads the current state of the limit switches.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility

Output

Name	Type	Description
switchstat	u16	hold states of limit switches

Parameter Discussion

switchstat holds a bitmap of the states the limit switches are in. As described below, each bit represents the state of one limit switch.

switchstat has the following format:

D15	D14	D13	D12	D11	D10	D9	D8
0	0	0	0	Axis 1 Home	Axis 2 Home	Axis 3 Home	Axis 4 Home

D7	D6	D5	D4	D3	D2	D1	D0
Axis 1 Fwd Limit	Axis 1 Rev Limit	Axis 2 Fwd Limit	Axis 2 Rev Limit	Axis 3 Fwd Limit	Axis 3 Rev Limit	Axis 4 Fwd Limit	Axis 4 Rev Limit

D11-D0: 1 = True 0 = False

Notes ***The logical value (True or False) read back for limit switch status may not directly correspond to the input signal level (high or low). Actual input signals may be programmed as inverted for active low limit switch configurations. (See the*** Set Limit Switch Polarity ***(***`set_lim_pol`***) function.)***

When using `read_lim_stat_rdb`**, switchstat** ***is stored in the Return Data Buffer. Use*** Communicate ***(***`communicate`***) in mode 2 to retrieve this data.***

read_pos
read_pos_rdb

Read Position

Format

status = read_pos (boardID, axis, position)
status = read_pos_rdb (boardID, axis)

Purpose

Reads the value of the position counter for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled

Output

Name	Type	Description
position	i32	position value

Parameter Discussion

axis is the axis to be controlled.

position is an instantaneous read of the position counter in the control loop for each axis. If motion is in process when this function is executed, the position count is constantly changing and the value only reflects the position at the moment it was read. The position value is placed in the return Return Data Buffer and held until it is read out by the host.

- Servo

The readback position value is a signal long word value (32-bits) in the range of:

–1,073,741,824 (-20^{30}) through 1,073,741,823 (2^{30}-1).

☞ **Note** ***All position values are in quadrature encoder counts.***

- Stepper

  The readback position value is a signed long word value (32-bits) in the range of:

  –2,147,483,648 (-2^{31}) through 2,147,483,647 (2^{31}-1).

☞ **Note** ***All position values are returned in steps.***

Using This Function

The value read is the position at the moment of actual function execution. You can execute this function at any time to determine the present position of the axis in question. The value returned is a signed hexadecimal long word value (32-bits).

☞ **Note** ***The position readback values are affected by user-programmed scale and offset parameters, if used. See*** Load Position Reference Offset ***(***`load_pos_ref`***)*** Load Position Scale Factor ***(***`load_pos_scale`***) and*** Set Scale Factor Sequence ***(***`set_scale_seg`***).***

When using `read_pos_rdb`**, position** ***is stored in the Return Data Buffer. Use*** Communicate ***(***`communicate`***) in mode 2 to retrieve this data.***

read_rdb

Read Return Data Buffer

Format

status = read_rdb (boardID, numofpckts, axis, cmdID, rdbdata)

Purpose

Reads packets from the return Return Data Buffer. This function is for advanced users only.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility

Output

Name	Type	Description
numofpckts	u16	number of packets to read
axis	[u16]	axis affected by function(s)
cmdID	[u16]	function ID(s)
rdbdata	[u32]	data word(s)

Parameter Discussion

numofpckts is the number of packets to read from the Return Data Buffer. Values must be 1 or greater.

axis is an array of axes. Each axis in the array has a corresponding function ID in **cmdID** and data in **rdbdata**.

cmdID is an array of function IDs. Each function ID in the array has a corresponding axis in **axis** and data in **rdbdata**.

rdbdata is an array of data word pairs. Each array entry is a set of two 16-bit words. Each array entry has a corresponding axis in **axis** and function ID in **cmdID**.

If **numofpckts** is > 1, then **axis**, **cmdID**, and **rdbdata** must be arrays of size **numofpckts**. The *Nth* position in each array correspond to each other, and forms one packet.

Using This Function

This function is similar to using Communicate (`communicate`) in mode 2.

This function allows one or more data packets to be read from the Return Data Buffer. Each packet contains information on what function was executed, what axis was effected, and what data, if any, was returned. The number of packets read is less than requested if there are fewer than request packets in the Return Data Buffer.

read_rpm

Read RPM

Format

status = read_rpm (boardID, axis, rpm)

Purpose

Reads the velocity in Revolutions Per Minute (RPM).

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled

Output

Name	Type	Description
rpm	f64	RPM value

Parameter Discussion

axis is the axis to be controlled.

rpm is a velocity value in RPM.

Using This Function

The Read RPM function returns the instantaneous RPM of a given axis. Although you can control any valid RPM value, the value read back is quantized because only the integer portion of velocity can be read back from the board.

- Servo

In order for the software to convert the current velocity into RPM, you must load Encoder Lines Per Revolution (`store_elc`) before trying this function.

- Stepper

  In order for the software to interpret the current velocity into RPM, you must load Steps Per Rev (`store_steps_rev`) or Load Steps and Lines/Rev (`load_steps_lines`) before trying this function.

☞ **Note** ***For DOS users, this function's prototype is: status*** = `read_rpm` ***(u16 boardaddr, u16 boardtype, u8 axis, u16 revdata f64 *rpm). Refer to*** *Special Considerations for DOS Users* ***earlier in this chapter for more information.***

read_steps_vel (Stepper only)

Read Steps/Sec

Format

status = read_steps_vel (boardID, axis, velocity)

Purpose

Retrieves current velocity in steps per second.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled

Output

Name	Type	Description
velocity	u32	velocity value

Parameter Discussion

axis is the axis to be controlled.

velocity is an unsigned long word (32 bits) in the range of:

1 through 1,048,576 steps/s

Using This Function

This function returns the magnitude of the instantaneous velocity set by the axis counter circuitry, in steps per second, at the moment the function is executed. The direction of motion can be obtained from the Read Per-Axis H/W Status (`read_axis_stat`) function.

The velocity valued returned is the steps per second presently being executed by the per-axis control circuitry. This value may be compared to the value programmed with the Load Steps/s (`store_steps_rev`) function to verify actual output velocity obtained.

☞ **Note** ***Actual motor velocity obtained may be less than programmed values due to system and mechanical constraints prohibiting the motor from attaining the programmed maximum velocity. Some mechanical system configurations may cause the motor***

to move at velocities greater or less than the programmed value. The actual step output velocity is controlled and should never exceed the programmed maximum value.

read_vel
read_vel_rdb (Servo only)

Read Present Velocity

Format

status = read_vel (boardID, axis, velocity)
status = read_vel_rdb (boadID, axis)

Purpose

Retrieves current velocity.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled

Output

Name	Type	Description
velocity	i16	velocity value

Parameter Discussion

axis is the axis to be controlled.

velocity has a velocity value returned in it. This value is a signed word (16-bit) in the range of:

–16,384 through 16,383

Velocity read is in counts/sample period.

Using This Function

This function shows only the integer portion of the present velocity being executed by the axis specified. Although velocity values are programmed as long word (32-bit) unsigned scaled values including an integer portion (upper 16-bits) and a fractional portion (lower 16-bits) this function returns the actual velocity integer portion only.

The absolute value of **velocity** is the integer portion showing present instantaneous velocity at the moment the function was executed. The sign information derived from this word shows actual direction of motion, either forward (positive values) or reverse (negative values).

The velocity value returned is the real velocity presently being executed by the per-axis servo control circuitry. This value may be compared with the integer portion (upper 16-bits) of the programmed velocity value to verify actual velocity obtained.

☞ **Notes** ***Actual velocity values obtained may be less than programmed values due to system and mechanical constraints prohibiting the servo loop from attaining the programmed velocity value.***

When using `read_vel_rdb`**, velocity** ***is stored in the Return Data Buffer. Use*** Communicate ***(***`communicate`***) in mode 2 to retrieve this data.***

reset_pos

Reset Position

Format

status = reset_pos (boardID, axis)

Purpose

Resets position counter for the specified axis to zero.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled

Parameter Discussion

axis is the axis to be controlled.

Using This Function

This function is used to reset the position counter to zero for the axis specified. This function sets the step position count value to zero immediately upon execution.

This function can only be executed when motion is stopped. If a motion trajectory is in process when this function is sent, the function is ignored, and the counters retain their original values.

- Stepper

For the closed-loop version of the stepper board, the encoder counter for the specified axis is also set to zero at the same time.

☞ **Note** ***The position count is set to zero during the Find Index function in closed-loop operation. The count then changes depending upon the actual index location. You can use the*** Reset Position ***function after the*** Find Index ***function to ensure counter alignment with the Index/Offset location.***

send_command

Send Command

Format

status = send_command (boardID, axis, cmd, wcount, cmddata)

Purpose

Sends any function specified by the ValueMotion board.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
cmd	u8	function ID
wcount	u8	number of data words being sent
cmddata	u32	data words

Parameter Discussion

axis is the axis to be controlled.

cmd is the function ID.

wcount is the number of data words being sent in **cmddata**. Possible values are 0 through 2.

cmddata holds the data words (up to 2 data words). A data word is 16-bits.

Using This Function

This function is just like Communicate (`communicate`) in mode 0. It is a way to send functions to the ValueMotion board. It is recommended you use the more advanced C API functions because of their ease-of-use. Every function that can be performed by `send_command` has a corresponding, simpler to use, function call.

set_base_vel (Stepper only)

Set Base Velocity

Format

status = set_base_vel (boardID, axis, basevelocity)

Purpose

Sets the base velocity used by the trajectory control circuitry for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
basevelocity	u16	base velocity

Parameter Discussion

axis is the axis to be controlled.

basevelocity is loaded in steps/s and is a 16-bit data word in the range of:

50 through 5000

The default value is 250.

☞ **Note** ***If parameter values exceed lower or upper limits, the function clamps to the respective limit and performs normally.***

Using This Function

Base velocity is the minimum step rate used by the trajectory generator during acceleration and deceleration. The default base velocity value is 250 steps/s. Larger or smaller values can be used to optimize the low frequency performance of certain stepper motors.

If the target velocity loaded with the Load Steps/Sec (`load_vel`) function is less than the programmed base velocity, the move occurs at the fixed velocity value loaded in steps/s velocity and will not follow a typical trapezoidal trajectory.

set_direction (Servo only)

Set Direction

Format

status = set_direction (boardID, axis, direction)

Purpose

Sets the desired direction of motion for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
direction	u16	velocity mode direction

Parameter Discussion

axis is the axis to be controlled.

direction is the direction of motion for the specified axis.

0 forward

–1 reverse

The default direction at power-up is reverse.

Using This Function

This function is only valid when the selected mode is velocity, and it is ignored if the selected mode is position. The Set Direction function must be followed by a Start Motion (`start_motion`) function to initiate the programmed change of direction request.

If the Set Direction function is issued when a velocity mode motion trajectory is being executed, and is followed by a start motion function, and the direction is opposite from that presently being executed, the motion automatically decelerates, changes to the new direction, and accelerates to the preprogrammed trajectory velocity.

☞ **Note** ***This function is valid for velocity mode only.***

set_gpio (Stepper only)

Set Aux Digital I/O Values

Format

status = set_gpio (boardID, data)

Purpose

Sets the 24-bit digital outputs on ports B and C.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
data	u32	digital data

Parameter Discussion

data contains the output data for ports B and C, in the following format:

Byte 0 (D7-D0) don't care

Byte 1 (D15-D8) port B output bits

Byte 2 (D23-D16) port C output bits

Byte 3 (D31-D24) don't care

Using This Function

If port C is configured for input, byte 2 is don't care.

Example

To set port B to binary 0110 1001 and port C to binary 1111 0001, you should set data to 0x0069F100.

☞ **Note** ***Port C must be configured correctly for output both on the hardware and in the software.***

set_io_output

Set I/O Port Output

Format

status = set_io_output (boardID, podata)

Purpose

Sets the output values for the I/O port.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
podata	u16	bitmap of output values

Parameter Discussion

podata has the following format. Use this function to set the desired logic level for the programmable output bits. The actual output signal level is dependent on the polarity selected.

D15	D14	D13	D12	D11	D10	D9	D8
0	0	0	0	0	0	0	0

D7	D6	D5	D4	D3	D2	D1	D0
I/O Bit 8	I/O Bit 7	I/O Bit 6	I/O Bit 5	I/O Bit 4	I/O Bit 3	I/O Bit 2	I/O Bit 1

D7-D0: 1 = True 0 = False (D7-D4: inhibit bits axis 4 through 1, stepper only)

- Servo

  The power-up default configuration of the I/O port has all bits set as inverting inputs.

- Stepper

  The power-up default configuration of the I/O port has bits 1 through 4 set as inverting inputs and bits 5 through 8 set as inverting inhibit outputs.

☞ **Note** ***I/O lines 5 through 8 are inhibit lines on the stepper boards. This function will not change the value on these lines.***

Using This Function

This function is used to set the logical (True or False) output value of the individual I/O port bits. The functionality of this function is dependent upon the proper configuration of the I/O port by use of the Set I/O Port Polarity and Direction (`set_io_pol`) function and the correct setting of the I/O port hardware jumpers.

Output values set using this function are held until changed by this function or the active occurrence of a breakpoint output. Breakpoint outputs are only available for closed-loop operation. Active breakpoint occurrence causes the associated I/O bit to transition, changing state. This applies to bits 1 through 4 corresponding to axes 1 through 4.

set_io_pol

Set I/O Port Polarity and Direction

Format

status = set_io_pol (boardID, pddata)

Purpose

Sets the directions and polarities of motion related I/O port signals.

Parameters

Input

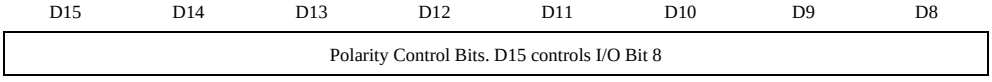
Name	Type	Description
boardID	u8	assigned by configuration utility
pddata	u16	bitmap of polarity and direction settings

Parameter Discussion

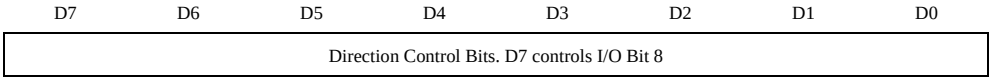
pddata is a bit map of polarity and direction states for an 8-bit I/O port. The upper 8 bits control the polarity on a per line basis. The lower 8 bits control the direction on a per line basis.

pddata has the following format. Use this function to set up the desired direction (input/output) and polarity (inverting/noninverting) configuration for the programmable I/O ports on a individual bit basis. The default configuration on power-up for all I/O bits is: inverting inputs.

Polarity and Direction set up word = 0xPPDD



D15-D8: 1 = Inverting 0 = Noninverting (D15 through D12 ignored on stepper boards)



D7-D0: 1 = Input 0 = Output (D7 through D4 ignored on stepper boards)

- Stepper

Notes ***I/O lines 5 through 8 are inhibit lines on the stepper board. This function does not change their polarity or direction. Use*** Set Step Output Mode and Polarity ***(***`set_step_mode_pol`***) to configure the polarity of the inhibit lines.***

Set I/O Port Polarity and Direction ***is for the set up and control of the motion related I/O port signals. The auxiliary 24 bit I/O port is also available for general purpose digital I/O. See*** Read Auxiliary Digital I/O Input Values ***(***`read_gpio`***) and*** Set Aux Digital I/O Output Values ***(***`set_gpio`***).***

Using This Function

Use this function to set the polarity and direction of the programmable I/O port on an individual bit basis. This function allows you to custom tailor the I/O port bits for system hardware signal direction and polarity requirements.

It is important to note that the proper function of the I/O port as a trigger input is dependent upon the correct set up of this function, the Enable I/O Port Trigger Inputs (`enable_io_trig`) function and the proper positioning of the I/O signal jumpers JP1 through JP8 and JP9 through JP16. Refer to your *ValueMotion Hardware User Manual* for more information on setting the hardware jumpers.

Caution ***If an I/O bit is set as an input port, and is enabled as a trigger input, but its polarity is set incorrectly, it causes degraded system performance. It may also erroneously load stored trigger functions. Be sure to check signal polarity set up prior to enabling the I/O port bits as trigger inputs.***

set_lim_pol

Set Limit Switch Input Polarity

Format

status = set_lim_pol (boardID, polarity)

Purpose

Sets the polarity of the limit switches (forward, reverse, and home).

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
polarity	u16	bitmap of polarity settings

Parameter Discussion

polarity has the following format:

D15	D14	D13	D12	D11	D10	D9	D8
0	0	0	0	Axis 1 Home Polarity	Axis 2 Home Polarity	Axis 3 Home Polarity	Axis 4 Home Polarity

D7	D6	D5	D4	D3	D2	D1	D0
Axis 1 Fwd Limit Pol	Axis 1 Rev Limit Pol	Axis 2 Fwd Limit Pol	Axis 2 Rev Limit Pol	Axis 3 Fwd Limit Pol	Axis 3 Rev Limit Pol	Axis 4 Fwd Limit Pol	Axis 4 Rev Limit Pol

D11-D0: 1 = Inverting 0 = Noninverting

Using This Function

This function is used to set the input polarity for the limit switch (forward and reverse) and home switch input signals for each axis on an individual signal basis. By using this function in conjunction with the Enable Limit Switch Inputs function, you may custom tailor the limit switch inputs according to your system requirements.

The default configuration for limit and home switch polarity is inverting inputs. Inverting input signals provide a true state for an active low input signal configuration. Noninverting input signals provide a true state for an active high input signal configuration.



Note ***Limit and home signals are used by the*** Find Home ***(***`find_home`***) function and must be properly configured before calling the*** Find Home ***function.***

set_loop_mode (Closed-Loop Stepper only)

Set Loop Mode

Format

status = set_loop_mode (boardID, axis, mode)

Purpose

Sets the mode of the stepper control circuitry for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
mode	i16	loop mode

Parameter Discussion

axis is the axis to be controlled.

mode is the loop mode for the specified axis.

0 open-loop (default)

–1 closed-loop

Using This Function

This function allows the closed-loop version of the stepper board to be used in both open and closed-loop modes. In open-loop mode, encoder feedback is not required for operation. However, the encoder counter is still active and can be read by executing a Read Encoder (`read_encoder`) function. In closed-loop mode, the encoder feedback is used to verify and *pull-in* motion to the target position.



Caution ***This function requires the closed-loop version of the stepper board. Attempting to execute this function on the open-loop board generates a function error.***

For proper closed-loop operation, the correct values for motor Steps/Rev ***and*** Encoder Lines/Rev ***must be set with the*** Set Steps and Lines/Rev ***(***`set_steps_lines`***) function. Incorrect values for these parameters may result in failure to reach the desired target position and erroneous stepper operation.***

set_portc_dir (Stepper only)

Set Port C Direction

Format

status = set_portc_dir (boardID, dir)

Purpose

Sets the direction of the user-selectable 24-bit digital port C I/O byte in conjunction with the DIP switch or jumper setting on the board.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
dir	u8	port direction

Parameter Discussion

dir specifies the direction of Port C.

0: output

1: input

Using This Function

For port C, the DIP switch setting 8 on ISA bus based stepper boards should be set to the ON state for port C to be configured as outputs.

In case of PCI bus based stepper boards jumper JP-6 should be on pins 1and 2 for port C to be configured as outputs and on pins 3 and 4 for port C to be configured as inputs. Refer to your *ValueMotion Hardware User Manual* for more information on setting the DIP switch or jumper.

☞ **Note** ***It is recommended that this function be used only during start-up initialization to set the direction of port C. When you use this function to configure the direction for port C, all the bits in port B and port C are cleared (set to low output value).***

set_pos_mode

Set Operation Mode

Format

status = set_pos_mode (boardID, axis, posmode)

Purpose

Sets the desired operation mode for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
posmode	u16	operation mode

Parameter Discussion

axis is the axis to be controlled.

posmode is the operation mode. There are three possible moves:

0x0000 position mode–absolute

0x00FF position mode–relative

0xFFXX velocity mode–absolute (XX are don't cares)

Using This Function

In absolute mode, target positions are programmed with respect to an origin or zero position. The origin is typically set at a home switch, end of travel limit switch, or encoder index position.

In relative mode, target positions are programmed with respect to the present position.

In velocity mode, motion occurs at the programmed velocity using programmed acceleration and maintains that velocity until a stop is executed, a limit is encountered, or an incremental velocity change is issued. The operation mode may be set differently for each axis. The mode is usually set as part of an initialization routine but may also be changed at any time and take affect at the next start function for the axis.

- Servo

  These modes may be set differently for each axis. The operation mode may be changed at any time, but only parameters loaded after the change will be interpreted in the newly selected mode. Refer to Chapter 3, *Software Overview,* for more information on operation modes.

☞ **Note** ***When velocity mode is selected, the*** Set Direction ***(***`set_direction`***) function must be used to set the desired direction of motion.***

- Stepper

  Target positions loaded after a mode change are interpreted in the newly selected mode.

☞ **Note** ***When velocity mode is selected, the*** Load Target Position ***(***`load_target_pos`***) function must be used to set the desired direction of motion.***

The direction select function should be used as follows:

1. Select velocity mode
2. Load Position, 1 = Forward, –1 = Reverse
3. Start Motion

set_rs_pulse

Set Run/Stop Status Pulse

Format

status = set_rs_pulse (boardID, axis, pulse)

Purpose

Determines the minimum duration of the *run* state of the run/status indication.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
pulse	u16	pulse width in Run/Stop update increments

Parameter Discussion

axis is the axis to be controlled.

pulse is an unsigned 16-bit word in multiples of the basic run/stop status update rate. For example:

0 = No minimum pulse width. Run/Stop status is output immediately.

1 = Run/Stop status is forced high for approximately 20 ms regardless of the true run/stop status.

Pulse has a range of:

0 through 65,535

Using This Function

This function is used to determine the minimum duration of the *run* state of the run/status indication. (See Read Communication Status Register (`read_csr`) and Read Per-Axis H/W Status (`read_axis_stat`).) The run/stop status bits are set to *run* immediately upon execution of any start function and remains in the *run* state for at least this minimum programmed pulse width. At the end of this time, the run/stop status bit reverts to the normal mode of determining status by measuring change of position approximately every 20 ms.

☞ **Note** ***The run/stop status bits indicates run for this minimum pulse width even if the axis does not move, as in the case of a zero move distance. This is the primary usage***

for this function. The run/stop status bit may be used to sequence motion operations on all moves including zero length moves.

set_scale_seq (Closed-Loop Stepper and Servo only)

Set Scale Factor Sequence

Format

status = set_scale_seq (boardID, axis, sequence)

Purpose

This function, when used in conjunction with the Load Position Scale Factor (`load_pos_scale`) function, determines the order of operation of the scaling function.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
sequence	i16	sequence specified

Parameter Discussion

axis is the axis to be controlled.

sequence set which operation to do first (multiply or divide).

0 = multiply first, then divide (default)

–1 = divide first, then multiply

Using This Function

This function, when used in conjunction with the Load Position Scale Factor (`load_pos_scale`) function, determines the order of operation of the scaling function. Depending upon the selection of scale factors, it may be desirable to reverse the order of mathematical operation to optimize the accuracy of the result.

This function may be issued at any time, but will have no affect unless valid numerator and denominator scale factors with a factor other than one (1) have been set with the Load Position Scale Factor (`load_pos_scale`) function.

- Stepper



Note ***This function requires the closed-loop version of the stepper board. Attempting to execute this function on the open-loop board generates a function error. This***

function operates in both open and closed-loop modes on closed-loop stepper boards. See the Set Loop Mode ***(***`set_loop_mode`***) function for more information.***

set_step_mode_pol (Stepper only)

Set Step Output Mode and Polarity

Format

status = set_step_mode_pol (boardID, mpdata)

Purpose

Sets output mode and polarity for all axes.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
mpdata	u16	output mode and polarity settings

Parameter Discussion

mpdata contains information on what output mode to use and polarity settings.

mpdata has the following format:

D15	D14	D13	D12	D11	D10	D9	D8
0	0	0	0	0	0	0	0

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	Inhibit Output Polarity	Step Output Polarity	Step Output Mode

D2-D1: 1 = Inverting (Default) 0 = Noninverting
D0: 1 = Step and Direction (Default) 0 = CW/CCW

Note ***The configuration for this function applies to all 4 axes on the stepper board.***

Using This Function

This function is used to configure the step outputs on all four axes of the stepper board. The output mode is programmable to assure compatibility with the majority of stepper drivers available from third-party vendors.

Step and direction or independent CW and CCW outputs can be selected with either active high or active low polarity. This function is also used to set the polarity of the Inhibit output. Setup and configuration of output and inhibit bits by this function affect all four axes on the board.

set_stop_mode

Set Stop Mode

Format

status = set_stop_mode (boardID, axis, mode)

Purpose

Sets the stop modes for a specific axis.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
mode	u16	stop mode

Parameter Discussion

This function takes a compound data word consisting of two bytes. Each byte sets a different stop mode and can be thought of independently. The lower byte sets the stop mode used during the Stop Motion (`stop_motion`) function. The upper byte sets the stop mode used when the home switch is reached during the Find Home (`find_home`) function.

axis is the axis to be controlled.

mode has the following format:

mode = 0xYY ZZ

where:

YY = Home Switch Mode Byte and ZZ = Stop Motion mode Byte

00 = Halt Stop (for compatibility, Home Switch Mode Only)

01 = Decelerate to Stop

02 = Halt Stop (Default for Both Home Switch and Stop Motion)

03 = Kill Stop

Using This Function

This function is used to set the stop modes used by the axis specified. These modes are used by the Stop Motion (`stop_motion`) function to cause different stop types and by the Find Home (`find_home`) function to determine the type of stop executed when the home switch is activated. These stop modes are not used at a limit switch occurrence (limit switches always cause a halt stop, when the limit is enabled) or when a position error is generated (position/following error always causes a kill stop, closed-loop mode only).

Once the modes are programmed, subsequent stops are executed using the stop modes selected with this function. There are three possible stop modes:

- Decelerate To Stop
- Halt Stop (Default)
- Kill Stop

The decelerate to stop mode causes motion to stop by decelerating from the present trajectory being executed. The rate of deceleration is the same as the preprogrammed acceleration value. Mechanical system configurations may affect the dynamics of stopping motion.

The halt stop mode causes motion to stop by instantaneously terminating the present trajectory being executed. Mechanical system configurations may affect the actual stopped position and the dynamics of stopping motion. Mechanical system configurations may affect the actual stopped position and the dynamics of stopping motion.

The kill stop mode causes motion to stop by instantaneously terminating the present trajectory being executed and activating the inhibit output. If the inhibit output is used to disable the stepper driver, the motor may coast to a stop with only frictional forces causing the actual stop. Mechanical system configurations may affect the actual stopped position and the dynamics of stopping motion. This mode purges the individual axis specific input function buffer of any pending functions.

The stop modes selected by this function remains in effect until new stop modes are programmed. The default stop modes on power-up are halt stop. If this function is not executed to set the stop modes, the Stop Motion (`stop_motion`) function and the Find Home (`find_home`) function uses the default halt stop mode.

☞ **Note** ***If a home switch is encountered during normal operation and it is enabled (not during a find home sequence) it causes a halt stop to be executed. If you expect motion to travel through the home switch during normal operation, you should disable the home switch after the find home operation is complete.***

start_motion

Start Motion

Format

status = start_motion (boardID, axis)

Purpose

Starts motion for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled

Parameter Discussion

axis is the axis to be controlled.

Using This Function

When this function is executed and valid conditions exist for starting motion on the axis specified, the loaded motion trajectory parameters become the active motion control parameters and motion begins using these new parameter values. If no new parameter values (velocity, acceleration, and position) were loaded since the last start motion function, then the previous values are used.

This function verifies that the requested motion is valid by checking enabled limit switch inputs and other parameters. This prevents unwanted motion beyond a valid limit condition.

- Servo

This function may be issued at any time. If this function is issued while motion is in progress, a new motion trajectory relating to the most recently loaded position and velocity parameters will be executed, overriding the present motion trajectory being executed. Acceleration parameters may not be updated while motion is in progress. See the Load Acceleration (`load_accel`) function.

Example

If velocity and acceleration values were loaded and a destination position of:

400,000

was loaded for an axis configured in position–absolute mode, and the Start Motion function was issued, then motion would proceed towards the programmed position value.

However, if a new destination position of:

100,000

was loaded and a Start Motion function was issued, then motion would now automatically proceed to the new destination position without stopping. If motion had already proceeded beyond the new destination position then it would reverse direction to find the new position.

- Stepper

The start motion function may only be issued when the axis is stopped. If this function is issued while motion is in progress, it is ignored and nothing happens.

To determine if a previous move has completed, look at the move complete bit in the **csrdata** parameter returned by the Read Communications Status Register (`read_csr`) function.

stop_motion

Stop Motion

Format

status = stop_motion (boardID, axis)

Purpose

Stops motion on a specific axis.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled

Parameter Discussion

axis is the axis to be controlled.

Using This Function

This function is used to stop motion for the axis specified by axis. This function stops the motion for the axis specified using the stop mode selected by the Set Stop Mode (`set_stop_mode`) function. There are three possible stop modes:

- Decelerate to a stop using the programmed acceleration value
- Halt stop immediately
- Kill motion by setting the motor control signal to zero.

store_elc (Servo only)

Store Encoder Line Count

Format

status = store_elc (boardID, axis, elc)

Purpose

Stores the number of encoder lines per revolution for later calculations.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
elc	u16	number of encoder lines per revolution

Parameter Discussion

axis is the axis to be controlled.

elc is the number of encoder lines per revolution, for a given axis of a servo board.

Encoder lines/rev on quadrature encoders is one-fourth the number of encoder counts/rev. For example, if you read the encoder count (`read_encoder`), then turn the motor one revolution, and read the encoder count again, and the difference in values is 200, you should enter a value of 50 for encoder lines/rev.

Using This Function

This function is required previous to any velocity or acceleration functions.

☞ **Note** ***For stepper boards, use*** Store Steps Per Rev ***(***`store_steps_rev`***) if you have an open-loop stepper or*** Load Steps and Lines Per Rev ***(***`load_steps_lines`***) if you have a closed-loop stepper.***

store_steps_rev (Stepper only)

Store Steps Per Rev

Format

status = store_steps_rev (boardID, axis, steps)

Purpose

Stores the steps per revolution for the axis specified.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
axis	u8	axis to be controlled
steps	u16	steps per revolution

Parameter Discussion

axis is the axis to be controlled.

steps is the number of steps per revolution. If your motor is being controlled in a half-step or microstepping manner, you must consider the microstepping factor. For example, for a motor that has 200 steps per revolution that is being microstepped by a 10 to 1 factor, you should enter a value of 2,000.

Using This Function

This function is required previous to calling the Load RPM (`load_rpm`), Load RPSPS (`load_rpsps`), or Read RPM (`read_rpm`) functions, since the number of steps per revolution must be known to calculate RPM properly.

☞ **Notes** ***This function is not necessary if the*** Load Steps and Line Per Rev ***(***`load_steps_lines`***) function has already been called.***

For servo boards, use Store Encoder Line Count ***(***`store_elc`***).***

trig_buff_delim

Trigger Buffer Delimiter

Format

status = trig_buff_delim (boardID)

Purpose

This function is used during trigger buffer loading as a delimiter between triggered function sequences.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility

Using This Function

Functions previously stored in a trigger buffer are executed when a Trigger I/O Port (`trigger_io`) function is sent from the host, or when a valid transition occurs on a properly configured and enabled trigger input I/O bit.

Multiple function sequences may be loaded into a trigger buffer, separated by the trigger buffer delimiter. Each time the trigger buffer is fired, all functions are executed until a trigger buffer delimiter is encountered or the end of the buffer is reached. Trigger buffers are cyclical, so that when the last function in the buffer is executed, a subsequent firing of the buffer begins with functions located at the top (beginning) of the buffer.

Placing two trigger buffer delimiters in a row in a trigger buffer causes a NO-OP like function, and firing the trigger buffer does nothing.



Caution ***This function is designed for use in trigger buffer function sequence storage only. Executing this function during normal function processing causes a function error condition.***

trigger_io

Trigger I/O Port

Format

status = trigger_io (boardID, portnum)

Purpose

Use this function to manually trigger the execution of a prestored function or list of functions in a trigger buffer.

Parameters

Input

Name	Type	Description
boardID	u8	assigned by configuration utility
portnum	u16	trigger number 1– 4

Parameter Discussion

portnum is the number of the trigger buffer you want to execute. **portnum** has a range of:

1 through 4

Using This Function

This function allows you to implement the same trigger function caused by a properly configured and enabled I/O port input trigger bit, without actually using the I/O port hardware input signal. The I/O port event trigger system allows you to prestore functions in a trigger buffer, for execution by a valid trigger event input.

When using this function, only a single event trigger may be initiated by each function call.

Trigger input functionality may also be linked by a software function Set Breakpoint To Trigger Link (`enable_pos_trig`) to the breakpoint output bits. In this software link mode, a breakpoint output causes a valid trigger input on the corresponding trigger buffer.

☞ **Note** ***This function does not require that the I/O port bits be set up as inputs or that they be enabled as trigger inputs. This function overrides all I/O port settings and automatically executes previously buffered trigger functions.***

A

Error Codes

This appendix describes the error codes returned by the ValueMotion software.

Each ValueMotion function returns a status code that indicates whether the function was performed successfully. When a ValueMotion function returns a code that is non-zero, it means that the function did not execute, or did execute, but with a potentially serious side effect.

When checking for errors in your application, it is recommended that you use the symbolic name rather than the actual number. The symbolic names are defined in the files `motnerr.h` (for C/C++ users) and `motnerr.bas` (for Basic users) in the `Valuemotion\Include` directory.

For performance reasons, the ValueMotion functions do not always return errors, even when a function has not been executed, since the extra hardware reads would penalize all function invocations. If a function does not appear to be executing properly, use the `read_axis_stat` function to check the status of an axis. A summary of the error codes is listed in Table A-1.

Table A-1. Error Codes Summary

Error Code	Symbolic Name	Description
0	**NIMC_noError**	No Error.
1	**NIMC_readyToReceiveTimeoutError**	Ready to Receive Timeout. The board is still not ready to receive commands after the specified timeout period. This error may occur if the board is busy processing previous commands. If this error persists, even when the board should not be busy, contact National Instruments.
2	**NIMC_currentPacketError**	Packet error on current packet. The board receives an incomplete command packet and cannot execute the command unless it receives a terminator indicating the end of a packet.
3	**NIMC_noReturnDataBufferError**	No data in the Return Data Buffer. The kernel driver returns an error if it runs out of time waiting for the board to return data to the Return Data Buffer. For FlexMotion boards this error can also be returned if the board has not been initialized. At a minimum, you must clear the power-up status.
4	**NIMC_halfReturnDataBufferError**	Partial readback packet. The data returned by the board is incomplete. The kernel driver timed out after getting partial data.
5	**NIMC_boardFailureError**	Hardware failure. The hardware fails to respond to the specified operation. Please contact National Instruments.

Table A-1. Error Codes Summary (Continued)

Error Code	Symbolic Name	Description
6	**NIMC_badDeviceOrAxisError**	The device is invalid. This error is returned if an invalid axis number is passed in a function call made to a motion control board. For FlexMotion, this error is returned if an invalid device number is passed in a function call to the board; functions calls can be made to an axis, an encoder, a DAC, an ADC, etc. Device numbers are a super set of axis numbers; each device has a unique number.
7	**NIMC_CIPBitError**	Command in process is not being reset. Please contact National Instruments Technical Support.
8	**NIMC_previousPacketError**	Packet error on previous packet. The previous function call made to the board was incomplete. The board did not receive a terminator indicating the end of the packet.
9	**NIMC_packetErrBitNotClearedError**	Packet error bit not cleared by terminator (hardware error).
10	**NIMC_badCommandError**	Command ID not recognized. Invalid command sent to the motion control board.
11	**NIMC_badReturnDataBufferPacketError**	Corrupt readback data. The data returned by the motion control board is corrupt.
12	**NIMC_badBoardIDError**	Illegal board ID. You must use the board ID assigned to your board in the configuration utility, otherwise a bad slot number is passed to the board.
13	**NIMC_packetLengthError**	Command packet length is incorrect. The board received a command that had an incorrect command packet length.

Table A-1. Error Codes Summary (Continued)

Error Code	Symbolic Name	Description
14	**NIMC_closedLoopOnlyError**	This command is valid only on closed-loop axes (closed-loop stepper and servo).
15	**NIMC_returnDataBufferFlushError**	Unable to flush the Return Data Buffer.
16	**NIMC_servoOnlyError**	This command is valid only on servo axes.
17	**NIMC_stepperOnlyError**	This command is valid only on stepper axes.
18	**NIMC_closedLoopStepperOnlyError**	This command is valid only on closed-loop stepper axes.
19	**NIMC_noBoardConfigInfoError**	Board configuration information is missing or corrupt.
20	**NIMC_countsNotConfiguredError**	Steps/rev and/or Lines/rev not loaded for this board.
21	**NIMC_systemResetError**	System reset did not occur in maximum time allowed.
22	**NIMC_functionSupportError**	This command is not supported by this board.
23	**NIMC_parameterValueError**	One of the parameters passed into the function has an illegal value.
24	**NIMC_motionOnlyError**	Motion command sent to pcPosition (encoder) board.
25	**NIMC_returnDataBufferNotEmptyError**	The Return Data Buffer is not empty. Commands that expect data returned by the board cannot be sent until the Return Data Buffer is cleared. To clear the Return Data Buffer, use the Flush Return Data Buffer (`flush_rdb`) function or the Clear Return Data Buffer VI.
27	**NIMC_processTimeoutError**	Under Windows NT, a function call made to the motion control board timed out waiting for driver access.

Table A-1. Error Codes Summary (Continued)

Error Code	Symbolic Name	Description
28	**NIMC_insufficientSizeError**	The resource is not large enough to supported the specified operation.
29	**NIMC_reserved4Error**	Reserved for obsolete product.
30	**NIMC_reserved5Error**	Reserved for obsolete product.
31	**NIMC_reserved6Error**	Reserved for obsolete product.
32	**NIMC_reserved7Error**	Reserved for obsolete product.
33	**NIMC_badPointerError**	A NULL pointer has been passed into a function inappropriately.
34	**NIMC_wrongReturnDataError**	Incorrect data has been returned by the board. This data does not correspond to the data on the command sent to the board.
35	**NIMC_watchdogTimeoutError**	A fatal error has occurred on the board. You must reset the board by power cycling your computer. Please contact National Instruments Technical Support if this problem persists.
37	**NIMC_irrelevantAttributeError**	The specified attribute is not relevant.
38	**NIMC_internalSoftwareError**	An unexpected error has occurred internal to the driver. Please contact National Instruments with the name of the function or VI that returned this error.

B

ValueMotion Function Support

This appendix contains tables that summarize the ValueMotion function attributes and shows which ValueMotion functions each ValueMotion board supports. The ValueMotion functions are listed alphabetically according to function name.

Table B-1 summarizes the ValueMotion function attributes. Input Data Size (Words) Word Count, Dest Axis ID, Return Data Size, and Function ID # (hex) columns are for those using the `communicate` function.

Table B-1. ValueMotion Function Attribute Summary

Function Name	Descriptive Name	Servo	Stepper	Input Data Size (Words)	Word Count	Dest Axis I.D.	Return Data Size	Function ID # (hex)
acquire_samples	Acquire Samples	✓	—	1	3	0	Vari-able	5E
begin_prestore	Begin Trigger Buffer Prestore	✓	✓	1	3	0	None	5F
communicate	Communicate	✓	✓	—	—	—	—	—
enable_brk	Enable Breakpoint Function	✓	✓ (C.L. Only)	1	3	1,2,3,4	None	66
enable_io_trig	Enable I/O Port Trigger Inputs	✓	✓	1	3	0	None	46
enable_limits	Enable Limit Switch Inputs	✓	✓	1	3	0	None	4B
enable_pos_trig	Set Breakpoint To Trigger Link	✓	✓ (C.L. Only)	1	3	1,2,3,4	None	6E
end_prestore	End Trigger Buffer Prestore	✓	✓	—	2	0	None	60

Table B-1. ValueMotion Function Attribute Summary (Continued)

Function Name	Descriptive Name	Servo	Stepper	Input Data Size (Words)	Word Count	Dest Axis I.D.	Return Data Size	Function ID # (hex)
find_home	Find Home	✓	✓	1	3	1,2,3,4	None	51
find_index/find_index_rdb	Find Encoder Index	✓	✓ (C.L. Only)	2	4	1,2,3,4	None	52
flush_rdb	Flush Return Data Buffer	✓	✓	—	—	—	—	—
get_board_type	Get Board Type	✓	✓	—	—	—	—	—
get_motion_board_info	Get Motion Board Info	✓	✓	—	—	—	—	—
get_motion_board_name	Get Motion Board Name	✓	✓	—	—	—	—	—
in_pos	In Position	✓	✓	—	—	—	—	—
kill_motion	Kill Motion	✓	✓	—	2	1,2,3,4	None	4D
load_accel	Load Acceleration	✓	✓	2	4	1,2,3,4	None	62
load_accel_fact	Load Acceleration Factor		✓	1	3	1,2,3,4	None	63
load_break_mod	Load Breakpoint Modulus	✓	✓ (C.L. Only)	2	4	1,2,3,4	None	6F
load_deriv_gain	Load Filter Derivative Gain	✓	—	1	3	1,2,3,4	None	5A
load_deriv_per	Load Filter Derivative Sample Period	✓	—	1	3	1,2,3,4	None	57
load_fol_err	Load Following Error	✓	✓ (C.L. Only)	1	3	1,2,3,4	None	56
load_intg_gain	Load Filter Integral Gain	✓	—	1	3	1,2,3,4	None	59
load_intg_lim	Load Filter Integration limit	✓	—	1	3	1,2,3,4	None	5B
load_pos_brk	Load Position Breakpoint	✓	✓ (C.L. Only)	2	4	1,2,3,4	None	65

Table B-1. ValueMotion Function Attribute Summary (Continued)

Function Name	Descriptive Name	Servo	Stepper	Input Data Size (Words)	Word Count	Dest Axis I.D.	Return Data Size	Function ID # (hex)
load_pos_ref	Load Position Reference Offset	✓	✓ (C.L. Only)	2	4	1,2,3,4	None	68
load_pos_scale	Load Position Scale Factor	✓	✓ (C.L. Only)	2	4	1,2,3,4	None	69
load_prop_gain	Load Filter Proportional Gain	✓	—	1	3	1,2,3,4	None	58
load_rot_counts	Load Rotary Counts	✓	✓ (C.L. Only)	2	4	1,2,3,4	None	6C
load_rpm	Load RPM	✓	✓	—	—	—	—	—
load_rpsps	Load RPSPS	✓	✓	—	—	—	—	—
load_steps_lines	Load Steps and Lines/Rev	—	✓ (C.L. Only)	2	4	1,2,3,4	None	57
load_target_pos	Load Target Position	✓	✓	2	4	1,2,3,4	None	4E
load_time_brk	Load Anticipation Time Breakpoint	✓	—	1	3	1,2,3,4	None	67
load_vel	Load Velocity (load steps/sec for Stepper)	✓	✓	2	4	1,2,3,4	None	4F
load_vel_change	Incremental Velocity Change	—	✓	1	3	1,2,3,4	None	5E
master_slave_cfg	Master Slave Configure	✓	—	1	3	1,2,3,4	None	6D
multi_start	Multi-Axis Start	✓	✓	1	3	0	None	5C
read_adc	Read A/D Converter Analog Input Value	✓	✓ (C.L. Only)	1	3	0	1	70
read_axis_stat/read_axis_stat_rdb	Read Per-Axis H/W Status	✓	✓	—	2	1,2,3,4	1	49
read_csr	Read Communication Status Register	✓	✓	—	2	0	1	41

Table B-1. ValueMotion Function Attribute Summary (Continued)

Function Name	Descriptive Name	Servo	Stepper	Input Data Size (Words)	Word Count	Dest Axis I.D.	Return Data Size	Function ID # (hex)
read_encoder/read_encoder_rdb	Read Encoder	—	✓ (C.L. Only)	—	2	1,2,3,4	2	59
read_gpio/read_gpio_rdb	Read Auxiliary Digital I/O Input Values	—	✓	2	4	0	2	71
read_io_port/read_io_port_rdb	Read I/O Port	✓	✓	—	2	0	1	47
read_lim_stat/read_lim_stat_rdb	Read Limit Switch Status	✓	✓	—	2	0	1	42
read_pos/read_pos_rdb	Read Position	✓	✓	—	2	1,2,3,4	2	53
read_rdb	Read Return Data Buffer	✓	✓	—	—	—	—	—
read_rpm	Read RPM	✓	✓	—	—	—	—	—
read_steps_vel	Read Steps/Sec	—	✓	—	2	1,2,3,4	2	54
read_vel/read_vel_rdb	Read Present Velocity	✓	—	—	2	1,2,3,4	1	54
reset_pos	Reset Position	✓	✓	—	2	1,2,3,4	None	50
send_command	Send Command	✓	✓	—	—	—	—	—
set_base_vel	Set Base Velocity	—	✓	1	3	1,2,3,4	None	5B
set_direction	Set Direction	✓		1	3	1,2,3,4	None	63
set_gpio	Set Auxiliary Digital I/O Values	—	✓	2	4	0	None	72
set_io_output	Set I/O Port Output	✓	✓	1	3	0	None	45
set_io_pol	Set I/O Port Polarity and Direction	✓	✓	1	3	0	None	44
set_lim_pol	Set Limit Switch Input Polarity	✓	✓	1	3	0	None	4A
set_loop_mode	Set Loop Mode	—	✓ (C.L. Only)	1	3	1,2,3,4	None	5A
set_portc_dir	Set Port C Direction	—	✓	2	4	0	None	72

Table B-1. ValueMotion Function Attribute Summary (Continued)

Function Name	Descriptive Name	Servo	Stepper	Input Data Size (Words)	Word Count	Dest Axis I.D.	Return Data Size	Function ID # (hex)
set_pos_mode	Set Operation Mode	✓	✓	1	3	1,2,3,4	None	64
set_rs_pulse	Set Run/Stop Status Pulse	✓	✓	1	3	1,2,3,4	None	6B
set_scale_seq	Set Scale Factor Sequence	✓	✓ (C.L. Only)	1	3	1,2,3,4	None	6A
set_step_mode_pol	Set Step Output Mode and Polarity	—	✓	1	3	0	None	43
set_stop_mode	Set Stop Mode	✓	✓	1	3	1,2,3,4	None	55
start_motion	Start Motion	✓	✓	—	2	1,2,3,4	None	61
stop_motion	Stop Motion	✓	✓	—	2	1,2,3,4	None	4C
store_elc	Store Encoder Line Count	✓	—	—	—	—	—	—
store_steps_rev	Store Steps Per Rev	—	✓	—	—	—	—	—
trig_buff_delim	Trigger Buffer Delimeter	✓	✓	—	2	0	None	5D
trigger_io	Trigger I/O Port	✓	✓	1	3	0	None	48

Table B-2 lists the ValueMotion boards that support the ValueMotion function. A check mark indicates the hardware that the function supports. If you attempt to call a ValueMotion function using a device that the function does not support, ValueMotion returns an error.

Table B-2. ValueMotion Function Call Board Support

Function		Board													
Function Name	Descriptive Name	PC-Servo-2A	PC-Servo-4A	PC-Step-2OX	PC-Step-4OX	PC-Step-2CX	PC-Step-4CX	PCI-Servo-2A	PCI-Servo-4A	PCI-Step-2OX	PCI-Step-4OX	PCI-Step-2CX	PCI-Step-4CX	PXI-7314 (Step-4OX)	PXI-7324 (Step-4CX)
acquire_samples	Acquire Samples	✓	✓					✓	✓						
begin_prestore	Begin Trigger Buffer Prestore	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
communicate	Communicate	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
enable_brk	Enable Breakpoint Function	✓	✓			✓	✓	✓	✓			✓	✓		✓
enable_io_trig	Enable I/O Port Trigger Inputs	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
enable_limits	Enable Limit Switch Inputs	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
enable_pos_trig	Set Breakpoint To Trigger Link	✓	✓			✓	✓	✓	✓			✓	✓		✓
end_prestore	End Trigger Buffer Prestore	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
find_home	Find Home	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
find_index/find_index_rdb	Find Encoder Index	✓	✓			✓	✓	✓	✓			✓	✓		✓
flush_rdb	Flush Return Data Buffer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
get_board_type	Get Board Type	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
get_motion_board_info	Get Motion Board Info	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
get_motion_board_name	Get Motion Board Name	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
in_pos	In Position	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
kill_motion	Kill Motion	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
load_accel	Load Acceleration	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
load_accel_fact	Load Acceleration Factor			✓	✓	✓	✓			✓	✓	✓	✓	✓	✓
load_break_mod	Load Breakpoint Modulus	✓	✓			✓	✓	✓	✓			✓	✓		✓
load_deriv_gain	Load Filter Derivative Gain	✓	✓					✓	✓						
load_deriv_per	Load Filter Derivative Sample Period	✓	✓					✓	✓						
load_fol_err	Load Following Error	✓	✓			✓	✓	✓	✓			✓	✓		✓
load_intg_gain	Load Filter Integral Gain	✓	✓					✓	✓						
load_intg_lim	Load Filter Integration limit	✓	✓					✓	✓						
load_pos_brk	Load Position Breakpoint	✓	✓			✓	✓	✓	✓			✓	✓		✓

Table B-2. ValueMotion Function Call Board Support (Continued)

Function		Board													
Function Name	Descriptive Name	PC-Servo-2A	PC-Servo-4A	PC-Step-2OX	PC-Step-4OX	PC-Step-2CX	PC-Step-4CX	PCI-Servo-2A	PCI-Servo-4A	PCI-Step-2OX	PCI-Step-4OX	PCI-Step-2CX	PCI-Step-4CX	PXI-7314 (Step-4OX)	PXI-7324 (Step-4CX)
load_pos_ref	Load Position Reference Offset	✓	✓			✓	✓		✓	✓		✓	✓		✓
load_pos_scale	Load Position Scale Factor	✓	✓			✓	✓		✓	✓		✓	✓		✓
load_prop_gain	Load Filter Proportional Gain	✓	✓					✓	✓						
load_rot_counts	Load Rotary Counts	✓	✓			✓	✓		✓	✓		✓	✓		✓
load_rpm	Load RPM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
load_rpsps	Load RPSPS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
load_steps_lines	Load Steps and Lines/Rev					✓	✓					✓	✓		✓
load_target_pos	Load Target Position	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
load_time_brk	Load Anticipation Time Breakpoint	✓	✓					✓	✓						
load_vel	Load Velocity (load steps/sec for Stepper)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
load_vel_change	Incremental Velocity Change			✓	✓	✓	✓			✓	✓	✓	✓	✓	✓
master_slave_cfg	Master Slave Configure	✓	✓					✓	✓						
multi_start	Multi-Axis Start	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
read_adc	Read A/D Converter Analog Input Value	✓	✓			✓	✓					✓	✓		✓
read_axis_stat/read_axis_stat_rdb	Read Per-Axis H/W Status	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
read_csr	Read Communication Status Register	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
read_encoder/read_encode_rdb	Read Encoder					✓	✓					✓	✓		✓
read_gpio/read_gpio_rdb	Read Auxiliary Digital I/O Input Values			✓	✓	✓	✓			✓	✓	✓	✓	✓	✓
read_io_port/read_io_port_rdb	Read I/O Port	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
read_lim_stat/read_lim_stat _rdb	Read Limit Switch Status	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
read_pos/read_pos_rdb	Read Position	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
read_rdb	Read Return Data Buffer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
read_rpm	Read RPM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
read_steps_vel	Read Steps/Sec			✓	✓	✓	✓			✓	✓	✓	✓	✓	✓

Table B-2. ValueMotion Function Call Board Support (Continued)

Function		Board													
Function Name	Descriptive Name	PC-Servo-2A	PC-Servo-4A	PC-Step-2OX	PC-Step-4OX	PC-Step-2CX	PC-Step-4CX	PCI-Servo-2A	PCI-Servo-4A	PCI-Step-2OX	PCI-Step-4OX	PCI-Step-2CX	PCI-Step-4CX	PXI-7314 (Step-4OX)	PXI-7324 (Step-4CX)
read_vel/read_vel_rdb	Read Present Velocity	✓	✓					✓	✓						
reset_pos	Reset Position	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
send_function	Send Function	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
set_base_vel	Set Base Velocity			✓	✓	✓	✓			✓	✓	✓	✓	✓	✓
set_direction	Set Direction	✓	✓					✓	✓						
set_gpio	Set Auxiliary Digital I/O Values			✓	✓	✓	✓			✓	✓	✓	✓	✓	✓
set_io_output	Set I/O Port Output	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
set_io_pol	Set I/O Port Polarity and Direction	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
set_lim_pol	Set Limit Switch Input Polarity	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
set_loop_mode	Set Loop Mode					✓	✓					✓	✓		✓
set_portc_dir	Set Port C Direction			✓	✓	✓	✓			✓	✓	✓	✓	✓	✓
set_pos_mode	Set Operation Mode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
set_rs_pulse	Set Run/Stop Status Pulse	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
set_scale_seq	Set Scale Factor Sequence	✓	✓			✓	✓	✓	✓			✓	✓		✓
set_step_mode_pol	Set Step Output Mode and Polarity			✓	✓	✓	✓			✓	✓	✓	✓	✓	✓
set_stop_mode	Set Stop Mode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
start_motion	Start Motion	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
stop_motion	Stop Motion	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
store_elc	Store Encoder Line Count	✓	✓					✓	✓						
store_steps_rev	Store Steps Per Rev			✓	✓	✓	✓			✓	✓	✓	✓	✓	✓
trig_buff_delim	Trigger Buffer Delimeter	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
trigger_io	Trigger I/O Port	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

C

Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a fax-on-demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call 512 795 6990. You can access these services at:

United States: 512 794 5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as `anonymous` and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at 512 418 1111.

E-Mail Support (Currently USA Only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

`support@natinst.com`

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

Country	Telephone	Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Brazil	011 288 3336	011 288 8528
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Québec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 6120092	03 6120095
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
United Kingdom	01635 523545	01635 523154
United States	512 795 8248	512 794 5678

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax (___) _____Phone (___) _____

Computer brand_____ Model _____Processor_____

Operating system (include version number) _____

Clock speed _____MHz RAM _____MB Display adapter _____

Mouse ___yes ___no Other adapters installed_____

Hard disk capacity _____MB Brand_____

Instruments used _____

_____

National Instruments hardware product model_____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

_____

_____

_____

_____

List any error messages: _____

_____

_____

The following steps reproduce the problem: _____

_____

_____

_____

_____

ValueMotion Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products

Hardware revision _____

Base I/O address of hardware _____

Programming choice _____

National Instruments software _____

Other boards in system _____

Base I/O address of other boards _____

Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Operating system mode _____

Programming language _____

Programming language version _____

Other boards in system _____

Base I/O address of other boards _____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *ValueMotion™ Software Reference Manual*

Edition Date: May 1998

Part Number: 321939A-01

Please comment on the completeness, clarity, and organization of the manual.

_____

_____

_____

_____

_____

_____

_____

If you find errors in the manual, please record the page numbers and describe the errors.

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

E-Mail Address _____

Phone (___) _____ Fax (___) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, Texas 78730-5039

Fax to: Technical Publications
National Instruments Corporation
512 794 5678

Glossary

Prefix	Meanings	Value
µ-	micro-	10^{-6}
m-	milli-	10^{-3}
k-	kilo-	10^{3}
M-	mega-	10^{6}
c-	centi-	10^{-2}

Numbers/Symbols

%	percent
±	plus or minus
+	positive of, or plus
–	negative of, or minus
/	per
°	degree
Ω	ohm
%	percent
+	positive of, or plus

A

A	amperes
absolute mode	treat the target position loaded as position relative to zero (0) while making a move
absolute position	position relative to zero

active high	if a switch is active when its value goes high
active low	if a switch is active when its value goes low
A/D	analog-to-digital
address	character code that identifies a specific location (or series of locations) in memory
amplifier	the drive that delivers power to operate the motor in response to low level control signals. In general, the amplifier is designed to operate with a particular motor type-you cannot use a stepper drive to operate a DC brush motor, for instance.
anticipation time breakpoint	pre-loads a desired anticipation time relative to the end of a programmed motion trajectory. When the position corresponding to the desired anticipation time is reached, an external breakpoint signal transitions on a dedicated I/O port output line for the axis selected. The anticipation breakpoint function is similar to, and shares the same resources as, the position breakpoint function.
axis	the unit which is used to control a motor or any similar device

B

b	bit—one binary digit, either 0 or 1
backplane	an assembly, typically a PCB, with 96-pin connectors and signal paths that bus the connector pins. VXIbus systems will have two sets of bused connectors, called the J1 and J2 backplanes, or have three sets of bused connectors, called the J1, J2, and J3 backplane.
base address	a memory address that serves as the starting address for programmable registers. All other addresses are located by adding to the base address.
binary	a number system with a base of 2
buffer	temporary storage for acquired or generated data (software)
bus	the group of conductors that interconnect individual circuitry in a computer. Typically, a bus is the expansion vehicle to which I/O or other devices are connected. Examples of PC buses are the ISA and PCI bus.

byte	eight related bits of data, an eight-bit binary number. Also used to denote the amount of memory required to store one byte of data.

C

CCW	counter-clockwise—implies direction of rotation of the motor
CL	closed-loop—A broadly applied term relating to any system where output is measured and compared to input. The output is then adjusted to reach the desired condition. In motion control this term applies to a system using an encoder or any feedback device.
CPU	central processing unit
crossover data corruption	indicates that data being returned by the motion control board has been corrupted
crosstalk	an unwanted signal on one channel due to an input on a different channel
CW	clockwise—implies direction of rotation of the motor

D

DC	direct current
dedicated	assigned totally for a particular function
delimiter	group of functions stored in trigger buffers can be segregated into different sections using delimiters. Each time the buffer is triggered a section of functions is executed.
DGND	digital ground
digital I/O port	a group of digital input/output signals
DIP	dual inline package
DLL	the motion control dynamic link library for Windows. Provides the API (Application programming interface) for the motion control boards.
drivers	software that controls a specific hardware device such as a DAQ board or a GPIB interface board

E

encoder	a device that translates mechanical motion into electrical signals used for monitoring position or velocity

F

FIFO	first-in-first-out memory buffer—the first data stored is the first data sent to the acceptor. FIFOs are often used on DAQ devices to temporarily store incoming or outgoing data until that data can be retrieved or output. For example, an analog input FIFO stores the results of A/D conversions until the data can be retrieved into system memory, a process that requires the servicing of interrupts and often the programming of the DMA controller. This process can take several milliseconds in some cases. During this time, data accumulates in the FIFO for future retrieval. With a larger FIFO, longer latencies can be tolerated. In the case of analog output, a FIFO permits faster update rates, because the waveform data can be stored on the FIFO ahead of time. This again reduces the effect of latencies associated with getting the data from system memory to the DAQ device.
filter parameters	indicates the control loop parameter gains (PID gains) for a given axis
filtering	a type of signal conditioning that allows you to filter unwanted signals from the signal you are trying to measure
following error trip point	following error is the difference between the instantaneous commanded trajectory position and the feedback position. If the following error increases beyond the maximum allowable value entered (called the following error trip point), the motors trip out on following error (i.e. are killed, thus preventing the axis from running away).
freewheel	when power to a motor is stopped it is said to be in a freewheel condition
full-step	full-step mode of a stepper motor—for a two phase motor this is done by energizing two windings or phases at a time
function declaration	a specification showing the return value and parameters for a function
function library	a collection of related functions packaged into a dynamic link library (DLL) for Windows, or a library file for DOS

G

Gnd	ground
GND	ground

H

half-step	half-step mode of a stepper motor—for a two phase motor this is done by alternately energizing two windings and then only one. In half step mode, alternate steps are strong and weak but there is significant improvement in low-speed smoothness over the full-step mode.
hex	hexadecimal
home switch (input)	a reference position in a motion control system derived from a mechanical datum or switch. Often designated as the *zero* position. The motion controller halts the motor if it finds this switch active while doing a find home sequence.
host computer	computer into which the motion control board is plugged
Hz	hertz—the number of scans read or updates written per second

I

I/O	input/output—the transfer of data to/from a computer system involving communications channels, operator interface devices, and/or data acquisition and control interfaces
ID	identification
import library	a Windows specific file that contains information about the functions contained in a companion dynamic link library (DLL). Windows applications are typically linked to one or more import libraries.
in	inches
index	marker between consecutive encoder revolutions
interlock	5 volts bus interlock voltage

inverting	defines the polarity of a switch (limit switch, home switch etc.) when it is in its *active* state. If these inputs are active-low they are said to have inverting polarity.
IRQ	interrupt request
ISA	industry standard architecture

K

k	kilo—the standard metric prefix for 1,000, or 10^3, used with units of measure such as volts, hertz, and meters
K	kilo—the prefix for 1,024, or 2^{10}, used with B in quantifying data or computer memory

L

latching	a signal that maintains its value while in a given state, as opposed to a signal that momentarily pulses when entering or exiting a state
limit switch (input)	properly designed motion control systems have sensors called limit switches that alert the control electronics that physical end of travel is being approached and that the motion should stop

M

m	meters
microstep	micro-stepping mode of a stepper motor—subdividing the basic motor step by proportioning the current in the windings. In this way the step size is reduced and low speed smoothness is dramatically improved.
modulo position	treat the target position as total quadrature counts per revolution for an axis

N

noise	an undesirable electrical signal—Noise comes from external sources such as the AC power line, motors, generators, transformers, fluorescent lights, soldering irons, CRT displays, computers, electrical storms, welders, radio transmitters, and internal sources such as semiconductors, resistors, and capacitors. Noise corrupts signals you are trying to send or receive.
noninverting	defines the polarity of a switch (limit switch, home switch etc.) when it is in its *active* state. If these inputs are active-high low they are said to have non-inverting polarity.

O

OL	open-loop—refers to a motion control system where no external sensors (feedback devices) are used to provide position or velocity correction signals

P

packets	a packet implies a function sent to the motion control board or data for a function returned to the host computer by the motion control board
PID	Proportional, Derivative, Integral control loop.
port	(1) a communications connection on a computer or a remote controller (2) a digital port, consisting of four or eight lines of digital input and/or output
position breakpoint	position breakpoint for an encoder can be set in absolute or relative quadrature counts. When the encoder reaches a position breakpoint, the associated high-speed breakpoint output immediately transitions.
power cycling	implies turning the host computer off and then back on. This allows the motion control board to reset.
prestore	indicates trigger buffer function storage, which is a powerful feature providing the ability to store a series of functions in a buffer. These functions can be executed later by triggering the buffer.

pull-in move	when stepper motors are run in closed-loop mode, the encoder feedback is used to verify the position of an axis when the motion ends. The motion controller then commands the axis to do a final move so that it is at the desired target position.
PWM	Pulse Width Modulation—a method of controlling the average current in a motors phase windings by varying the on-time (duty cycle) of transistor switches

Q

quadrature counts	the encoder resolution times four. The encoder resolution is the number of encoder lines between consecutive encoder indexes (marker or Z-bit). If the encoder does not have an index output the encoder resolution can be referred to as lines per revolution.

R

RAM	random-access memory
relative breakpoint	sets the position breakpoint for an encoder in relative quadrature counts
relative mode	treat the target position loaded as position relative to current position while making a move
relative position	position relative to current position
ribbon cable	a flat cable in which the conductors are side by side
rotary axis	an axis for which rotary counts are loaded. The axis moves to the target position by taking the shortest path, either forward or backwards, while remaining within the one revolution defined by the loaded rotary counts. Similarly, the value read with the Read Present Position function will be a positive value that is modulo the loaded rotary count value.
RPM	revolutions per minute. Units for velocity.
RPSPS or RPS/S	revolutions per second square. Units for acceleration and deceleration.

S

s	seconds

servo	(1) specifies an axis that controls a servo motor (2) specifies when a servo motor becomes active
slot	a position where a module can be inserted into a VXIbus backplane. Each slot provides the 96-pin J connectors to interface with the board P connectors. A slot can have one, two, or three connectors.
stepper	specifies an axis that controls a stepper motor.

T

toggle	changing state from high to low, back to high and so on
torque	force tending to produce rotation
trapezoidal profile	a typical motion trajectory, where a motor accelerates up to the programmed velocity using the programmed acceleration, traverses at the programmed velocity and then decelerates at the programmed acceleration to the target position.
trigger	any event that causes or starts some form of data capture
TTL	transistor-transistor logic

V

V	volts
velocity mode	move the axis continuously at the specified velocity

W

watchdog	a timer task that shuts down (resets) the motion control board if any serious error occurs
word	the standard number of bits that a processor or memory manipulates at one time. Microprocessors typically use 8-, 16-, or 32-bit words.

Z

Z-bit	marker between consecutive encoder revolutions

Index

Numbers and Symbols

[ ] (brackets), indicating arrays, 4-3
24-bit digital I/O, 3-18
32-bit Windows application development, 2-2

A

absolute position mode, 3-4
acceleration and acceleration factor parameters, 3-10 to 3-11
acceleration functions
    load_accel, 3-2, 3-9, 4-34 to 4-35
    load_accel_fact, 4-36 to 4-37
    load_rpsps, 4-60 to 4-61
acquire_samples function, 4-6 to 4-7
A/D converter analog input values, reading, 4-79 to 4-80
application development
    ValueMotion DOS library, 2-3
    ValueMotion Windows libraries, 2-1 to 2-3
        32-bit Windows application, 2-2
        LabWindows/CVI applications, 2-3
        Windows 3.1 application, 2-2
arrays, 4-3
axes, killed on power-up, 3-3
axis functions
    multi_start, 3-13, 4-77 to 4-78
    per-axis specific functions, 3-1
    read_axis_stat, 4-81 to 4-83
    read_axis_stat_rdb, 4-81 to 4-83
    set_direction, 4-107

B

base velocity parameters, 3-11
begin_prestore function, 3-15, 4-8 to 4-9
board IDs
    first parameter for functions, 4-1
    get_board_type function, 4-26 to 4-27
    shipping and non-shipping devices (table), 4-26 to 4-27
board information functions
    get_board_type, 4-26 to 4-27
    get_motion_board_info, 4-28 to 4-29
    get_motion_board_name, 4-30 to 4-31
board-level communication function buffer, 3-15
board-level functions, 3-1
Borland Delphi (Pascal), primary type names and ranges (table), 4-2 to 4-3
brackets [ ], indicating arrays, 4-3
breakpoint functions, 3-17 to 3-18
    enable_brk, 3-18, 4-11 to 4-13
    enable_pos_trig, 4-18
    load_break_mod, 4-38 to 4-39
    load_pos_brk, 3-17, 4-48 to 4-50
    load_time_brk, 3-17, 4-67 to 4-69
    purpose and use, 3-17 to 3-18
buffer allocation for Windows 32-bit applications, 2-2
buffers. *See* communications buffers; return data buffer; trigger buffer functions.
building applications
    ValueMotion DOS library, 2-3
    ValueMotion Windows libraries, 2-1 to 2-3
        32-bit Windows application, 2-2
        LabWindows/CVI applications, 2-3
        Windows 3.1 application, 2-2
bulletin board support, C-1

C

C/C++, primary type names and ranges (table), 4-2 to 4-3
closed-loop functions, 3-12 to 3-13

communicate function
description, 4-10 to 4-11
reading data immediately, 3-15
Communication Status register, reading, 4-84 to 4-85
communications buffers, 3-14 to 3-16
board-level communication function buffer, 3-15
I/O port trigger buffers, 3-15 to 3-16
return data buffer, 3-15
constants include file for ValueMotion, 4-3 to 4-4
conventions used in manual, *xii-xiii*
creating applications. *See* building applications.
customer communication, *xiii*, C-1 to C-2

D

data types
arrays, 4-3
primary type names and ranges (table), 4-2 to 4-3
decelerated stop mode, 3-11 to 3-12
derivative gain. *See* load_deriv_gain function.
derivative sample period. *See* load_deriv_per function.
digital I/O
24-bit digital I/O, 3-18
read_gpio function, 3-18, 4-88 to 4-89
read_gpio_rdb function, 4-88 to 4-89
set_gpio function, 3-18, 4-108
set_portc_dir, 3-18, 4-116
documentation
conventions used in manual, *xii-xiii*
organization of manual, *xi*
related documentation, *xiii*
DOS
building applications, 2-3
programming considerations, 4-5

E

electronic support services, C-1 to C-2
e-mail support, C-2
enable_brk function, 3-18, 4-12 to 4-13
enable_io_trig function
description, 4-14 to 4-15
I/O port configuration, 3-6, 3-17
enable_limits function
description, 4-16 to 4-17
limit switch configuration, 3-3
using in initialization, 3-2
enable_pos_trig function, 4-18
encoder count
read_encoder function, 3-13, 4-86 to 4-87
read_encoder_rdb function, 3-13, 4-86 to 4-87
store_elc function, 4-130
end_prestore function, 3-15, 4-19
error codes
overview, A-1
summary (table), A-2 to A-5
error conditions, 3-19

F

fatal failure modes, 3-19
fax and telephone support numbers, C-2
Fax-on-Demand support, C-2
filter parameters, PID, 3-6 to 3-7
find_home function
configuring home switch, 3-7
description, 4-20 to 4-21
using in initialization, 3-2, 3-7
valid functions during Find Home sequence, 4-21
find_index function
closed-loop control, 3-13
description, 4-22 to 4-24
requirements, 3-7
using in initialization, 3-2

find_index_rdb function, 4-22 to 4-24
flush_rdb function, 4-25
following error, 3-12. *See also* load_fol_err function.
FTP support, C-1
function prototypes
    included with ValueMotion software, 2-2
    including in source code (note), 4-3
function reference
    acquire_samples, 4-6 to 4-7
    begin_prestore, 3-15, 4-8 to 4-9
    communicate, 3-15, 4-10 to 4-11
    enable_brk, 3-18, 4-12 to 4-13
    enable_io_trig, 3-6, 3-17, 4-14 to 4-15
    enable_limits, 3-2, 3-3, 4-16 to 4-17
    enable_pos_trig, 4-18
    end_prestore, 3-15, 4-19
    find_home, 3-2, 3-7, 4-20 to 4-21
    find_index, 3-2, 3-7, 3-13, 4-22 to 4-24
    find_index_rdb, 4-22 to 4-24
    flush_rdb, 4-25
    get_board_type, 4-26 to 4-27
    get_motion_board_info, 4-28 to 4-29
    get_motion_board_name, 4-30 to 4-31
    in_pos, 4-32
    kill_motion, 3-8, 4-33
    load_accel, 3-2, 3-9, 4-34 to 4-35
    load_accel_fact, 3-9, 4-36 to 4-37
    load_break_mod, 4-38 to 4-39
    load_deriv_gain, 3-6, 4-40
    load_deriv_per, 3-2, 3-6, 4-41 to 4-42
    load_fol_err, 3-3, 3-12, 3-13, 4-43 to 4-44
    load_intg_gain, 3-2, 3-6, 4-45 to 4-46
    load_intg_lim, 3-6, 4-47
    load_pos_brk, 3-17, 4-48 to 4-50
    load_pos_ref, 4-51 to 4-52
    load_pos_scale, 4-53 to 4-54
    load_prop_gain, 3-2, 3-6, 4-55
    load_rot_counts, 4-56 to 4-57
    load_rpm, 3-9, 4-58 to 4-59
    load_rpsps, 4-60 to 4-61
    load_steps_lines, 3-2, 3-13, 4-62 to 4-63
    load_target_pos, 3-9, 4-64 to 4-66
    load_time_brk, 3-17, 4-67 to 4-69
    load_vel, 3-2, 3-9, 4-70 to 4-72
    load_vel_change, 3-9, 4-73 to 4-74
    master_slave_cfg, 4-75 to 4-76
    multi_start, 3-13, 4-77 to 4-78
    read_adc, 4-79 to 4-80
    read_axis_stat, 4-81 to 4-83
    read_axis_stat_rdb, 4-81 to 4-83
    read_csr, 3-15, 4-84 to 4-85
    read_encoder, 3-13, 4-86 to 4-87
    read_encoder_rdb, 3-13, 4-86 to 4-87
    read_gpio, 3-18, 4-88 to 4-89
    read_gpio_rdb, 4-88 to 4-89
    read_io_port, 3-6, 4-90 to 4-91
    read_io_port_rdb, 4-90 to 4-91
    read_lim_stat, 4-92 to 4-93
    read_lim_stat_rdb, 4-92 to 4-93
    read_pos, 4-94 to 4-95
    read_pos_rdb, 4-94 to 4-95
    read_rdb, 3-15, 4-96 to 4-97
    read_rpm, 4-98 to 4-99
    read_steps_vel, 4-100 to 4-101
    read_vel, 4-102 to 4-103
    read_vel_rdb, 4-102 to 4-103
    reset_pos, 4-104
    send_command, 4-105
    set_base_vel, 3-9, 3-11, 4-106
    set_direction, 4-107
    set_gpio, 3-18, 4-108
    set_io_output, 3-6, 4-109 to 4-110
    set_io_pol, 3-5, 4-111 to 4-112
    set_lim_pol, 3-2, 3-3, 3-7, 4-113 to 4-114
    set_loop_mode, 3-2, 3-13, 4-115
    set_portc_dir, 3-18, 4-116
    set_pos, 3-2
    set_pos_mode, 3-2, 3-3, 4-117 to 4-118
    set_rs_pulse, 4-119 to 4-120
    set_scale_seq, 4-121 to 4-122

set_step_mode_pol, 3-5, 3-6, 4-123 to 4-124
set_stop, 3-2
set_stop_mode, 3-2, 3-10, 3-11, 4-125 to 4-126
start_motion, 3-8, 3-13, 4-127 to 4-128
stop_motion, 3-11, 4-129
store_elc, 4-130
store_steps_rev, 4-131
trig_buff_delim, 3-15, 3-17, 4-132
trigger_io, 3-15, 3-17, 4-133
function types
board-level functions, 3-1
per-axis specific functions, 3-1
functions
call board support (table), B-6 to B-8
function attribute summary (table), B-1 to B-5
primary type names and ranges (table), 4-2 to 4-3
status codes and board IDs, 4-1

G

get_board_type function, 4-26 to 4-27
get_motion_board_info function, 4-28 to 4-29
get_motion_board_name function, 4-30 to 4-31

H

halt stop mode, 3-12
header file, including in source code (note), 4-3
home switch
configuration, 3-3
enable_limits function, 4-16 to 4-17
motion interactions, 3-13 to 3-14

I

import libraries, 2-2
include file for ValueMotion constants, 4-3 to 4-4
initialization, 3-1 to 3-7
absolute position mode, 3-4
find_home function, 3-7
find_index function, 3-7
inverting steps and direction signals (note), 3-2
I/O port configuration, 3-5 to 3-6
limit and home switch configuration, 3-3
PID filter parameters, 3-6 to 3-7
position modes, 3-3 to 3-5
recommended procedure, 3-1 to 3-3
relative position mode, 3-4
required steps (table), 3-2
step output mode and polarity, 3-6
velocity mode, 3-5
in_pos function, 4-32
installing ValueMotion boards, 1-1
integral gain. *See* load_intg_gain function.
integration limit. *See* load_intg_lim function.
I/O port configuration
enable_io_trig function, 4-14 to 4-15
overview, 3-5 to 3-6
read_io_port function, 4-90 to 4-91
set_io_output function, 4-109 to 4-110
set_io_pol function, 4-111 to 4-112
I/O port trigger buffers, 3-15 to 3-16

K

kill stop mode, 3-12
kill_motion function, 3-8, 4-33

L

LabVIEW software, 1-2

LabWindows/CVI software
building applications
Windows 3.1, 2-3
Windows 95/98/NT, 2-3
features, 1-2
languages. *See* programming languages.
limit switch
configuration, 3-3
enable_limits function, 4-16 to 4-17
motion interactions, 3-13 to 3-14
read_lim_stat function, 4-92 to 4-93
read_lim_stat_rdb function, 4-92 to 4-93
set_lim_pol function, 4-113 to 4-114
load_accel function
description, 4-34 to 4-35
required for initialization (table), 3-2
setting motion trajectory parameters (table), 3-9
load_accel_fact function
description, 4-36 to 4-37
setting motion trajectory parameters (table), 3-9
load_break_mod function, 4-38 to 4-39
load_deriv_gain function
description, 4-40
PID filter parameter, 3-6
load_deriv_per function
description, 4-41 to 4-42
PID filter parameter, 3-6
required for initialization (table), 3-2
load_fol_err function
closed-loop control, 3-13
description, 4-43 to 4-44
programming following error count, 3-12
using in initialization, 3-3
load_intg_gain function
description, 4-45 to 4-46
PID filter parameter, 3-6
required for initialization (table), 3-2
load_intg_lim function
description, 4-47
PID filter parameter, 3-6
load_pos_brk function, 3-17, 4-48 to 4-50
load_pos_ref function, 4-51 to 4-52
load_pos_scale function, 4-53 to 4-54
load_prop_gain function
description, 4-55
PID filter parameter, 3-6
required for initialization (table), 3-2
load_rot_counts function, 4-56 to 4-57
load_rpm function
description, 4-58 to 4-59
setting motion trajectory parameters (table), 3-9
load_rpsps function, 4-60 to 4-61
load_steps_lines function
closed-loop control, 3-13
description, 4-62 to 4-63
required for initialization (table), 3-2
load_target_pos function
description, 4-64 to 4-66
setting motion trajectory parameters (table), 3-9
load_time_brk function, 3-17, 4-67 to 4-69
load_vel function
description, 4-70 to 4-72
required for initialization (table), 3-2
setting motion trajectory parameters (table), 3-9
load_vel_change function, 3-9, 4-73 to 4-74
loop functions
closed-loop functions, 3-12 to 3-13
set_loop_mode, 3-2, 4-115

M

manual. *See* documentation.
master_slave_cfg function, 4-75 to 4-76
Motion Board Configuration Utility, 1-1
motion trajectory, 3-8 to 3-14
closed-loop functions, 3-12 to 3-13

motion interactions with limit and home switches, 3-13 to 3-14
overview, 3-8 to 3-9
parameters, 3-9 to 3-11
acceleration and acceleration factor, 3-10 to 3-11
base velocity, 3-11
stop mode selection, 3-11 to 3-12
decelerated stop, 3-11 to 3-12
halt stop, 3-12
kill stop, 3-12
stop on following error, 3-12
trapezoidal trajectory profile (figure), 3-8
MotnCnst.bas file, 4-3 to 4-4
multi_start function
description, 4-77 to 4-78
motion interaction, 3-13

N

National Instruments application software, 1-2

O

operation mode, setting, 4-117 to 4-118

P

parameter passing, for Windows 32-bit applications, 2-2
parameters
acceleration and acceleration factor, 3-10 to 3-11
base velocity, 3-11
motion trajectory parameters, 3-9 to 3-10
PID filter parameters, 3-6 to 3-7
Pascal (Borland Delphi), primary type names and ranges (table), 4-2 to 4-3
pcMotion32.dll, 2-1
pcmotion.dll, 2-1
per-axis specific functions, 3-1
PID filter parameters, 3-6 to 3-7
polarity
set_io_pol function, 4-111 to 4-1112
set_lim_pol function, 4-113 to 4-114
set_step_mode_pol function, 4-123 to 4-124
step output mode and polarity, 3-6
Port C, setting, 4-116
position functions
in_pos, 4-32
load_pos_ref, 4-51 to 4-52
load_pos_scale, 4-53 to 4-54
load_target_pos, 3-9, 4-64 to 4-66
read_pos, 4-94 to 4-95
read_pos_rdb, 4-94 to 4-95
reset_pos, 4-104
position modes, 3-3 to 3-5
absolute position mode, 3-4
relative position mode, 3-4
velocity mode, 3-5
prestore functions
begin_prestore, 3-15, 4-8 to 4-9
end_prestore, 3-15, 4-19
primary type names and ranges (table), 4-2 to 4-3
programming languages
using functions, 4-3 to 4-4
ValueMotion software support for, 1-2
proportional gain. *See* load_prop_gain function.
pull-in moves, 3-8, 3-12, 3-13

R

_rdb functions, 4-4 to 4-5
find_index_rdb, 4-22 to 4-24
flush_rdb, 4-25
read_axis_stat_rdb, 4-81 to 4-83
read_encoder_rdb, 4-86 to 4-87
read_gpio_rdb, 4-88 to 4-89
read_io_port_rdb, 4-90 to 4-91

read_lim_stat_rdb, 4-92 to 4-93
read_pos_rdb, 4-94 to 4-95
read_rdb, 4-96 to 4-97
read_vel_rdb, 4-102 to 4-103
using _rdb functions, 4-4 to 4-5
read_adc function, 4-79 to 4-80
read_axis_stat function, 4-81 to 4-83
description, 4-81 to 4-83
illegal conditions, 4-82 to 4-83
valid functions in find home sequence, 4-83
read_axis_stat_rdb function, 4-81 to 4-83
read_csr function
checking status of Return Data Pending bit, 3-15
description, 4-84 to 4-85
read_encoder function
closed-loop control, 3-13
description, 4-86 to 4-87
read_encoder_rdb function
closed-loop control, 3-13
description, 4-86 to 4-87
read_gpio function, 3-18, 4-88 to 4-89
read_gpio_rdb function, 4-88 to 4-89
read_io_port function
description, 4-90 to 4-91
I/O port configuration, 3-6
read_io_port_rdb function, 4-90 to 4-91
read_lim_stat function, 4-92 to 4-93
read_lim_stat_rdb function, 4-92 to 4-93
read_pos function, 4-94 to 4-95
read_pos_rdb function, 4-94 to 4-95
read_rdb function
description, 4-96 to 4-97
reading data immediately, 3-15
read_rpm function, 4-98 to 4-99
read_steps_vel function, 4-100 to 4-101
read_vel function, 4-102 to 4-103
read_vel_rdb function, 4-102 to 4-103
relative position mode, 3-4
reset_pos function, 4-104
return data buffer
flushing (flush_rdb function), 4-25
purpose and use, 3-15
reading packets (read_rdb function), 4-96 to 4-97
rotary counts, loading, 4-56 to 4-57
run/stop status pulse, setting, 4-119 to 4-120

S

scaling functions
load_pos_scale, 4-53 to 4-54
set_scale_seq, 4-121 to 4-122
send_command function, 4-105
set_base_vel function
changing base velocity, 3-11
description, 4-106
setting motion trajectory parameters (table), 3-9
set_direction function, 4-107
set_gpio function, 3-18, 4-108
set_io_output function
description, 4-109 to 4-110
I/O port configuration, 3-6
set_io_pol function
description, 4-111 to 4-112
I/O port configuration, 3-5
set_lim_pol function
description, 4-113 to 4-114
limit switch configuration, 3-3, 3-7
using in initialization, 3-2
set_loop_mode function
closed-loop control, 3-13
description, 4-115
required for initialization (table), 3-2
set_portc_dir function, 3-18, 4-116
set_pos_mode function
description, 4-117 to 4-118
required for initialization (table), 3-2
setting position mode, 3-3
set_rs_pulse function, 4-119 to 4-120

set_scale_seq function, 4-121 to 4-122
set_step_mode_pol function
description, 4-123 to 4-124
I/O port configuration, 3-5
setting stepper interface, 3-6
set_stop_mode function
description, 4-125 to 4-126
preprogramming stop mode, 3-11
required for initialization (table), 3-2
using with stop motion command, 3-10
software overview. *See* ValueMotion software overview.
software programming choices, 1-2
start_motion function
description, 4-127 to 4-128
motion interaction, 3-8, 3-13
step functions
load_steps_lines, 3-2, 3-13, 4-62 to 4-63
read_steps_vel, 4-100 to 4-101
set_step_mode_pol, 3-5, 3-6, 4-123 to 4-124
store_steps_rev, 4-131
step output mode and polarity, 3-6
stop mode selection, 3-11 to 3-12
decelerated stop, 3-11 to 3-12
halt stop, 3-12
kill stop, 3-12
set_stop_mode function, 4-125 to 4-126
stop on following error, 3-12. *See also* load_fol_err function.
stop_motion function
description, 4-129
preprogramming stop mode, 3-11
store_elc function, 4-130
store_steps_rev function, 4-131

T

target position. *See* load_target_pos function.
technical support, C-1 to C-2
telephone and fax support numbers, C-2
trapezoidal trajectory profile (figure), 3-8
with acceleration factor, 3-11
trigger buffer functions
begin_prestore, 3-15, 4-8 to 4-9
end_prestore, 3-15, 4-19
purpose and use, 3-16 to 3-17
trig_buff_delim function, 3-15, 3-17, 4-132
trigger_io function, 3-15, 3-17, 4-133
trigger buffer size in function packets (table), 4-9

V

ValueMotion boards
call board support (table), B-6 to B-8
function attribute summary (table), B-1 to B-5
installing, 1-1
ValueMotion constants include file, 4-3 to 4-4
ValueMotion DOS library, 2-3
ValueMotion software
features, 1-1
language support, 1-2
requirements for getting started, 1-1
software programming choices, 1-2
ValueMotion software overview, 3-1 to 3-19. *See also* function reference.
24-bit digital I/O, 3-18
breakpoint functions, 3-17 to 3-18
communications buffers, 3-14 to 3-16
board-level communication function buffer, 3-15
I/O port trigger buffers, 3-15 to 3-16
return data buffer, 3-15
error conditions, 3-19
function types
board-level functions, 3-1
per-axis specific functions, 3-1
initialization, 3-1 to 3-7
absolute position mode, 3-4

find_home function, 3-7
find_index function, 3-7
inverting steps and direction signals (note), 3-2
I/O port configuration, 3-5 to 3-6
limit and home switch configuration, 3-3
PID filter parameters, 3-6 to 3-7
position modes, 3-3 to 3-5
recommended procedure, 3-1 to 3-3
relative position mode, 3-4
required steps (table), 3-2
step output mode and polarity, 3-6
velocity mode, 3-5
motion trajectory, 3-8 to 3-14
closed-loop functions, 3-12 to 3-13
motion interactions with limit and home switches, 3-13 to 3-14
overview, 3-8 to 3-9
parameters, 3-9 to 3-11
acceleration and acceleration factor, 3-10 to 3-11
base velocity, 3-11
stop mode selection, 3-11 to 3-12
decelerated stop, 3-11 to 3-12
halt stop, 3-12
kill stop, 3-12
stop on following error, 3-12
trigger buffer functions, 3-16 to 3-17
ValueMotion Windows libraries, 2-1 to 2-3
32-bit Windows application, 2-2
dynamic link libraries (DLLs), 2-1
pcMotion32.dll, 2-1
pcmotion.dll, 2-1
function prototypes, 2-2
import libraries, 2-2
LabWindows/CVI applications, 2-3
Windows 3.1 application, 2-2
variable data types
arrays, 4-3
primary type names and ranges (table), 4-2 to 4-3
velocity functions
load_rpm, 3-9, 4-58 to 4-59
load_vel, 3-2, 3-9, 4-70 to 4-72
load_vel_change, 3-9, 4-73 to 4-74
read_rpm, 4-98 to 4-99
read_steps_vel, 4-100 to 4-101
set_base_vel, 3-9, 3-11, 4-106
velocity mode, 3-5
Visual Basic for Windows
primary type names and ranges (table), 4-2 to 4-3
programming considerations, 4-3 to 4-4
ValueMotion constants include file, 4-3 to 4-4

W

Windows applications, building
32-bit (Windows 95/98/NT), 2-2
buffer allocation, 2-2
parameter passing, 2-2
LabWindows/CVI applications, 2-3
Windows 3.1, 2-2
Windows libraries. *See* ValueMotion Windows libraries.

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP