Using Help

<u>Conventions</u> <u>Navigating Help</u> <u>Searching Help</u> <u>Printing Help File Topics</u>

Conventions

This help file uses the following formatting and typographical conventions:

< > Angle brackets that contain numbers separated by an ellipsis represent a range of values associated with a bit or signal name—for example, AO <3..0>. [] Square brackets enclose optional items—for example, [response]. The » symbol leads you through nested menu items and » dialog box options to a final action. The sequence File»Page Setup»Options directs you to pull down the File menu, select the Page Setup item, and select Options from the last dialog box. P This icon denotes a tip, which alerts you to advisory information. N This icon denotes a note, which alerts you to important information. This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash. bold Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names, emphasis, or an introduction to a key concept. dark red Text in this color denotes a caution. Underlined text in this color denotes a link to a help topic, green help file, or Web address. italic Italic text denotes variables, emphasis, cross-references, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply. monospace Text in this font denotes text or characters that you enter from the keyboard, sections of code, programming

examples, and syntax examples. This font is also used for

the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.

monospace Bold text in this font denotes the messages and responses
 bold that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.

monospace Italic text in this font denotes text that is a placeholder for a *italic* word or value that you must supply.

Navigating Help (Windows Only)

To navigate this help file, use the **Contents**, **Index**, and **Search** tabs to the left of this window or use the following toolbar buttons located above the tabs:

- Hide—Hides the navigation pane from view.
- Locate—Locates the currently displayed topic in the Contents tab, allowing you to view related topics.
- **Back**—Displays the previously viewed topic.
- **Forward**—Displays the topic you viewed before clicking the **Back** button.
- **Options**—Displays a list of commands and viewing options for the help file.

Searching Help (Windows Only)

Use the **Search** tab to the left of this window to locate content in this help file. If you want to search for words in a certain order, such as "related documentation," add quotation marks around the search words as shown in the example. Searching for terms on the **Search** tab allows you to quickly locate specific information and information in topics that are not included on the **Contents** tab.

Wildcards

You also can search using asterisk (*) or question mark (?) wildcards. Use the asterisk wildcard to return topics that contain a certain string. For example, a search for "prog*" lists topics that contain the words "program," "programmatically," "progress," and so on.

Use the question mark wildcard as a substitute for a single character in a search term. For example, "?ext" lists topics that contain the words "next," "text," and so on.



Note Wildcard searching will not work on Simplified Chinese, Traditional Chinese, Japanese, and Korean systems.

Nested Expressions

Use nested expressions to combine searches to further refine a search. You can use Boolean expressions and wildcards in a nested expression. For example, "example AND (program OR VI)" lists topics that contain "example program" or "example VI." You cannot nest expressions more than five levels.

Boolean Expressions

Click the **•** button to add Boolean expressions to a search. The following Boolean operators are available:

- **AND** (default)—Returns topics that contain both search terms. You do not need to specify this operator unless you are using nested expressions.
- **OR**—Returns topics that contain either the first or second term.
- **NOT**—Returns topics that contain the first term without the second term.
- **NEAR**—Returns topics that contain both terms within eight words of each other.

Search Options

Use the following checkboxes on the **Search** tab to customize a search:

- Search previous results—Narrows the results from a search that returned too many topics. You must remove the checkmark from this checkbox to search all topics.
- Match similar words—Broadens a search to return topics that contain words similar to the search terms. For example, a search for "program" lists topics that include the words "programs," "programming," and so on.
- Search titles only—Searches only in the titles of topics.

Printing Help File Topics (Windows Only)

Complete the following steps to print an entire book from the **Contents** tab:

- 1. Right-click the book.
- 2. Select **Print** from the shortcut menu to display the **Print Topics** dialog box.
- 3. Select the **Print the selected heading and all subtopics** option.
 - Note Select Print the selected topic if you want to print the single topic you have selected in the **Contents** tab.
- 4. Click the **OK** button.

Printing PDF Documents

This help file may contain links to PDF documents. To print PDF documents, click the print button located on the Adobe Acrobat Viewer toolbar.

Axes

An axis consists of a trajectory generator, PID or stepper control block, and some sort of output resource, either a digital-to-analog converter (DAC) output or a stepper pulse generator output. Servo axes must also have some sort of feedback resource, either an encoder or ADC channel. Closed-loop stepper axes also require a feedback resource, and can use either encoder or ADC inputs. Open-loop stepper axes do not require feedback for correct operation. The following table lists the resource IDs and constants for axes:

Resource Name	Resource ID	Constant
Axis Control	0 (0x00)	NIMC_AXIS_CTRL
Axis 1	1 (0x01)	NIMC_AXIS1
Axis 2	2 (0x02)	NIMC_AXIS2
Axis 3	3 (0x03)	NIMC_AXIS3
Axis 4	4 (0x04)	NIMC_AXIS4
Axis 5	5 (0x05)	NIMC_AXIS5
Axis 6	6 (0x06)	NIMC_AXIS6
Axis 7	7 (0x07)	NIMC_AXIS7
Axis 8	8 (0x08)	NIMC_AXIS8
Axis 9	9 (0x09)	NIMC_AXIS9
Axis 10	10 (0x0A)	NIMC_AXIS10
Axis 11	11 (0x0B)	NIMC_AXIS11
Axis 12	12 (0x0C)	NIMC_AXIS12
Axis 13	13 (0x0D)	NIMC_AXIS13
Axis 14	14 (0x0E)	NIMC_AXIS14
Axis 15	15 (0x0F)	NIMC_AXIS15
Axis 16	177 (0xB1)	NIMC_AXIS16
Axis 17	178 (0xB2)	NIMC_AXIS17
Axis 18	179 (0xB3)	NIMC_AXIS18
Axis 19	180 (0xB4)	NIMC_AXIS19
Axis 20	181 (0xB5)	NIMC_AXIS20

Axis 21	182 (0xB6)	NIMC_AXIS21
Axis 22	183 (0xB7)	NIMC_AXIS22
Axis 23	184 (0xB8)	NIMC_AXIS23
Axis 24	185 (0xB9)	NIMC_AXIS24
Axis 25	186 (0xBA)	NIMC_AXIS25
Axis 26	187 (0xBB)	NIMC_AXIS26
Axis 27	188 (0xBC)	NIMC_AXIS27
Axis 28	189 (0xBD)	NIMC_AXIS28
Axis 29	190 (0xBE)	NIMC_AXIS29
Axis 30	191 (0xBF)	NIMC_AXIS30

Functions that can operate on multiple axes simultaneously, such as <u>Read Blend Status</u> and <u>Start Motion</u>, can take the axis control (NIMC_AXIS_CTRL) as their resource parameter.

Use the appropriate constant value defined in the MotnCnst.h or motncnst.bas file. For example, to use axis 9 with the <u>Configure Axis</u> <u>Resources</u> function, pass NIMC_AXIS9 for the axis parameter:

status = flex_config_axis(u8 boardID, NIMC_AXIS9, u8 primaryFeedback, u8
secondaryFeedback,

u8 primaryOutput, u8 secondaryOutput);

Vector Spaces

Vector spaces are logical, multidimensional groups of axes. They can be either one-dimensional, two-dimensional with x and y axes, or threedimensional with x, y, and z axes. The total number of vector spaces NI-Motion supports in your motion control system can be calculated using the following formula:

total vector spaces = [number of activated axes/2].

Vector spaces facilitate 2D and 3D interpolated moves: linear, circular, helical, and spherical. You can send a vector space to many NI-Motion functions to define vector position, vector velocity, vector acceleration, and so on.

Vector spaces are started, stopped, and controlled as if they were a single axis, greatly simplifying the control of coordinated vector axes. All axes in a vector space start and stop at the same time, completing the vector motion profiles programmed. The following table lists the resource IDs and constants for vector space control.

Resource Name	Resource ID	Constant
Vector Space Control	0x10	NIMC_VECTOR_SPACE_CTRL
Vector Space 1	0x11	NIMC_VECTOR_SPACE1
Vector Space 2	0x12	NIMC_VECTOR_SPACE2
Vector Space 3	0x13	NIMC_VECTOR_SPACE3
Vector Space 4	0x14	NIMC_VECTOR_SPACE4
Vector Space 5	0x15	NIMC_VECTOR_SPACE5
Vector Space 6	0x16	NIMC_VECTOR_SPACE6
Vector Space 7	0x17	NIMC_VECTOR_SPACE7
Vector Space 8	0x18	NIMC_VECTOR_SPACE8
Vector Space 9	0x19	NIMC_VECTOR_SPACE9
Vector Space 10	0x1A	NIMC_VECTOR_SPACE10
Vector Space 11	0x1B	NIMC_VECTOR_SPACE11
Vector Space 12	0x1C	NIMC_VECTOR_SPACE12
Vector Space 13	0x1D	NIMC_VECTOR_SPACE13
Vector Space 14	0x1E	NIMC_VECTOR_SPACE14

Vector Space 15	0x1F	NIMC_VECTOR_SPACE15

Functions that can operate on multiple vector spaces simultaneously (for example, <u>Start Motion</u>) can take the vector space control (NIMC_VECTOR_SPACE_CTRL) as their resource parameter.

Vector spaces are configured by mapping axes to the vector space with the <u>Configure Vector Space</u> function. Vector spaces are logical, not physical, and do not require motion resources other than those used by the axes themselves.

Motion Resources

There are four types of motion resources on the NI-Motion controller: <u>encoders</u>, <u>ADC channels</u>, <u>DAC outputs</u>, and <u>stepper outputs</u>. In general, functions relating to motion resources (for example, <u>Read DAC</u> and <u>Read</u> <u>Steps Generated</u>) can be sent to the resource or the axis the resource is mapped to.



Note After they are mapped to an axis, all features and functions of a motion resource are available as part of the axis. It is not necessary to remember or use the resource number directly when accessing these features as part of the axis. Resources are referenced by axis number after they are assigned to that axis.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using these resources with a 73xx controller:

- 7330 controllers do not support DAC outputs or secondary ADCs.
- 7340 controllers do not support secondary ADCs.
- 7350 controllers support a secondary feedback device on each axis.
- 7390 controllers do not support ADC channels, DAC outputs, secondary ADCs, or secondary encoders.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using these resources with the NI SoftMotion Controller:

- The NI SoftMotion Controller supports a secondary feedback device on axes one through fifteen.
 - Note Secondary feedback devices are not available on axes sixteen through thirty.

- Secondary Encoders

Use secondary encoders in a dual-loop feedback system to provide a more accurate velocity feedback estimation.

Resource Name	Resource ID	Constant
Secondary Encoder 1	0X71	NIMC_SECONDARY_ENCODER1
Secondary Encoder 2	0X72	NIMC_SECONDARY_ENCODER2
•	•	
•	•	
•	•	
Secondary Encoder 15	0X7F	NIMC_SECONDARY_ENCODER15

- Secondary ADCs

Resource Name	Resource ID	Constant
Secondary ADC 1	0X91	NIMC_SECONDARY_ADC1
Secondary ADC 2	0X92	NIMC_SECONDARY_ADC2
•	•	
•	•	•
•	•	

Secondary ADC	0X9F	NIMC_SECOND	ARY_ADC15
15			

Encoders

Encoder resources are primarily used for position feedback on servo and closed-loop stepper axes. When encoder resources are not mapped to an axis for use as axis feedback, you can use them for any number of other functions including position or velocity monitoring, as digital potentiometer encoder inputs, or as master encoders for master/slave and gearing applications. The following table lists the resource IDs and constants for encoders:

Resource Name	Resource ID	Constant
Encoder Control	0x20	NIMC_ENCODER_CTRL
Encoder 1	0x21	NIMC_ENCODER1
Encoder 2	0x22	NIMC_ENCODER2
Encoder 3	0x23	NIMC_ENCODER3
Encoder 4	0x24	NIMC_ENCODER4
Encoder 5	0x25	NIMC_ENCODER5
Encoder 6	0x26	NIMC_ENCODER6
Encoder 7	0x27	NIMC_ENCODER7
Encoder 8	0x28	NIMC_ENCODER8
Encoder 9	0x29	NIMC_ENCODER9
Encoder 10	0x2A	NIMC_ENCODER10
Encoder 11	0x2B	NIMC_ENCODER11
Encoder 12	0x2C	NIMC_ENCODER12
Encoder 13	0x2D	NIMC_ENCODER13
Encoder 14	0x2E	NIMC_ENCODER14
Encoder 15	0x2F	NIMC_ENCODER15
Encoder 16	0xC1	NIMC_ENCODER16
Encoder 17	0xC2	NIMC_ENCODER17
Encoder 18	0xC3	NIMC_ENCODER18
Encoder 19	0xC4	NIMC_ENCODER19
Encoder 20	0xC5	NIMC_ENCODER20

Encoder 21	0xC6	NIMC_ENCODER21
Encoder 22	0xC7	NIMC_ENCODER22
Encoder 23	0xC8	NIMC_ENCODER23
Encoder 24	0xC9	NIMC_ENCODER24
Encoder 25	0xCA	NIMC_ENCODER25
Encoder 26	0xCB	NIMC_ENCODER26
Encoder 27	0xCC	NIMC_ENCODER27
Encoder 28	0xCD	NIMC_ENCODER28
Encoder 29	0xCE	NIMC_ENCODER29
Encoder 30	0xCF	NIMC_ENCODER30

Functions that can operate on multiple encoders simultaneously (for example, <u>Read High-Speed Capture Status</u>) can take the encoder control (NIMC_ENCODER_CTRL) as their resource parameter.

All encoders feature high-speed capture inputs and breakpoint outputs. These features are implemented in the encoder processor FPGA and are fully functional when an encoder is used as an independent resource or as feedback for an axis.

ADC Channels

You can use ADC channels as analog feedback for axes or as generalpurpose analog inputs to measure sensors or potentiometers.

All ADC channels are multiplexed and automatically scanned to keep the converted ADC register values current. The following table lists the resource IDs and constants for ADCs:

Resource Name	Resource ID	Constant
ADC 1	0x51	NIMC_ADC1
ADC 2	0x52	NIMC_ADC2
ADC 3	0x53	NIMC_ADC3
ADC 4	0x54	NIMC_ADC4
ADC 5	0x55	NIMC_ADC5
ADC 6	0x56	NIMC_ADC6
ADC 7	0x57	NIMC_ADC7
ADC 8	0x58	NIMC_ADC8
ADC 9	0x59	NIMC_ADC9
ADC 10	0x5A	NIMC_ADC10
ADC 11	0x5B	NIMC_ADC11
ADC 12	0x5C	NIMC_ADC12
ADC 13	0x5D	NIMC_ADC13
ADC 14	0x5E	NIMC_ADC14
ADC 15	0x5F	NIMC_ADC15
ADC 16	0xF1	NIMC_ADC16
ADC 17	0xF2	NIMC_ADC17
ADC 18	0xF3	NIMC_ADC18
ADC 19	0xF4	NIMC_ADC19
ADC 20	0xF5	NIMC_ADC20
ADC 21	0xF6	NIMC_ADC21
ADC 22	0xF7	NIMC_ADC22
ADC 23	0xF8	NIMC_ADC23

ADC 24	0xF9	NIMC_ADC24
ADC 25	0xFA	NIMC_ADC25
ADC 26	0xFB	NIMC_ADC26
ADC 27	0xFC	NIMC_ADC27
ADC 28	0xFD	NIMC_ADC28
ADC 29	0xFE	NIMC_ADC29
ADC 30	0xFF	NIMC_ADC30

ADC channels do not typically provide the same level of feedback performance as encoders, but have the advantage of providing absolute rather than incremental feedback.

DAC Outputs

DAC resources are typically mapped to servo axes and generate the analog control outputs from the PID loops. DAC resources that are not used by axes are available for non-axis specific applications. You can directly control an unmapped DAC as a general-purpose analog output.

The DAC outputs offer 16-bit resolution and the industry-standard ± 10 V range. Refer to the specifications of the motion controller documentation for complete DAC output specifications. The following table lists the resource IDs and constants for DACs:

Resource Name	Resource ID	Constant
DAC 1	0x31	NIMC_DAC1
DAC 2	0x32	NIMC_DAC2
DAC 3	0x33	NIMC_DAC3
DAC 4	0x34	NIMC_DAC4
DAC 5	0x35	NIMC_DAC5
DAC 6	0x36	NIMC_DAC6
DAC 7	0x37	NIMC_DAC7
DAC 8	0x38	NIMC_DAC8
DAC 9	0x39	NIMC_DAC9
DAC 10	0x3A	NIMC_DAC10
DAC 11	0x3B	NIMC_DAC11
DAC 12	0x3C	NIMC_DAC12
DAC 13	0x3D	NIMC_DAC13
DAC 14	0x3E	NIMC_DAC14
DAC 15	0x3F	NIMC_DAC15
DAC 16	0xD1	NIMC_DAC16
DAC 17	0xD2	NIMC_DAC17
DAC 18	0xD3	NIMC_DAC18
DAC 19	0xD4	NIMC_DAC19
DAC 20	0xD5	NIMC_DAC20

DAC 21	0xD6	NIMC_DAC21
DAC 22	0xD7	NIMC_DAC22
DAC 23	0xD8	NIMC_DAC23
DAC 24	0xD9	NIMC_DAC24
DAC 25	0xDA	NIMC_DAC25
DAC 26	0xDB	NIMC_DAC26
DAC 27	0xDC	NIMC_DAC27
DAC 28	0xDD	NIMC_DAC28
DAC 29	0xDE	NIMC_DAC29
DAC 30	0xDF	NIMC_DAC30

Stepper Outputs

Stepper output resources generate the step pulses required for stepper axis control. They operate like the DAC output in a servo axis.

NI-Motion supports the two industry-standard stepper output configurations: Step and Direction, or CW/CCW pulses. Refer to <u>Configure Stepper Output</u> for more information about these output configurations. The following table lists the resource IDs and constants for stepper outputs:

Resource Name	Resource ID	Constant
Stepper Output 1	0x41	NIMC_STEP_OUTPUT1
Stepper Output 2	0x42	NIMC_STEP_OUTPUT2
Stepper Output 3	0x43	NIMC_STEP_OUTPUT3
Stepper Output 4	0x44	NIMC_STEP_OUTPUT4
Stepper Output 5	0x45	NIMC_STEP_OUTPUT5
Stepper Output 6	0x46	NIMC_STEP_OUTPUT6
Stepper Output 7	0x47	NIMC_STEP_OUTPUT7
Stepper Output 8	0x48	NIMC_STEP_OUTPUT8
Stepper Output 9	0x49	NIMC_STEP_OUTPUT9
Stepper Output 10	0x4A	NIMC_STEP_OUTPUT10
Stepper Output 11	0x4B	NIMC_STEP_OUTPUT11
Stepper Output 12	0x4C	NIMC_STEP_OUTPUT12
Stepper Output 13	0x4D	NIMC_STEP_OUTPUT13
Stepper Output 14	0x4E	NIMC_STEP_OUTPUT14
Stepper Output 15	0x4F	NIMC_STEP_OUTPUT15
Stepper Output 16	0xE1	NIMC_STEP_OUTPUT16
Stepper Output 17	0xE2	NIMC_STEP_OUTPUT17
Stepper Output 18	0xE3	NIMC_STEP_OUTPUT18
Stepper Output 19	0xE4	NIMC_STEP_OUTPUT19
Stepper Output 20	0xE5	NIMC_STEP_OUTPUT20
Stepper Output 21	0xE6	NIMC_STEP_OUTPUT21

Stepper Output 22	0xE7	NIMC_STEP_OUTPUT22
Stepper Output 23	0xE8	NIMC_STEP_OUTPUT23
Stepper Output 24	0xE9	NIMC_STEP_OUTPUT24
Stepper Output 25	0xEA	NIMC_STEP_OUTPUT25
Stepper Output 26	0xEB	NIMC_STEP_OUTPUT26
Stepper Output 27	0xEC	NIMC_STEP_OUTPUT27
Stepper Output 28	0xED	NIMC_STEP_OUTPUT28
Stepper Output 29	0xEE	NIMC_STEP_OUTPUT29
Stepper Output 30	0xEF	NIMC_STEP_OUTPUT30

General-Purpose I/O Ports

You can use the general-purpose I/O port structure for programmable logic controller (PLC) functions or for simple point I/O. Refer to <u>Analog &</u> <u>Digital I/O</u> for information about how to configure the direction and polarity of the I/O ports and bits, set and reset individual bits, and read the logical port status.

These I/O ports are also hardware resources, but because they are never mapped to axes, they are not considered motion resources. Refer to 73xx Controller General-Purpose I/O Port IDs and NI SoftMotion Controller General-Purpose I/O Port IDs for I/O port resource IDs and constants.



Notes

- 1. The RTSI port is not supported by the NI SoftMotion Controller.
- 2. The resource ID for the RTSI port (NIMC_RTSI_PORT) is 0x09.

Available I/O Resources

The following table lists the I/O resources by controller type.

Device	Digital I/O Lines
NI 7330	four 8-bit ports, RTSI I/O port
NI 7340	four 8-bit ports, RTSI I/O port
NI 7350	eight 8-bit ports, RTSI I/O port
NI PCI-7390	four 2-bit input ports, four 2-bit output ports, RTSI I/O port
NI SoftMotion Controller for Xenus (Copley Controls CANopen drive)	seven digital input lines, three digital output lines per axis
NI SoftMotion Controller for Accelnet (Copley Controls CANopen drive)	seven digital input lines, two digital input lines per axis

73xx Controller General-Purpose I/O Port IDs and Constants

7330, 7340, and 7350 Controller I/O Port Resource IDs and Constants

The following table lists the resource IDs and constants for I/O ports on 7330, 7340, and 7350 controllers.



Note The I/O line direction is user configurable using the <u>Set I/O</u> <u>Port Direction</u> function, so these controllers do not have separate port IDs for input and output ports.

Resource Name	Resource ID	Constant
I/O Port 1	0x01	NIMC_IO_PORT1
I/O Port 2	0x02	NIMC_IO_PORT2
I/O Port 3	0x03	NIMC_IO_PORT3
I/O Port 4	0x04	NIMC_IO_PORT4
I/O Port 5	0x05	NIMC_IO_PORT5
I/O Port 6	0x06	NIMC_IO_PORT6
I/O Port 7	0x07	NIMC_IO_PORT7
I/O Port 8	0x08	NIMC_IO_PORT8
RTSI Port	0x09	NIMC_RTSI_PORT

PCI-7390 Controller I/O Port Resource IDs

The following table lists the resource IDs for I/O ports on the PCI-7390 controller:

	Digi	tal Input Ports		
Resource Name	Resource ID	Constant	Resource Name	Reso II
Input Port 1	0x01	NIMC_DIGITAL_INPUT_PORT1	Output Port 1	0x81
Input Port 2	0x02	NIMC_DIGITAL_INPUT_PORT2	Output Port 2	0x82
Input Port 3	0x03	NIMC_DIGITAL_INPUT_PORT3	Output Port 3	0x83
Input Port 4	0x04	NIMC_DIGITAL_INPUT_PORT4	Output Port 4	0x84

NI SoftMotion Controller General-Purpose I/O Port IDs and Constants

The following table lists the resource IDs and constants for I/O ports on the NI SoftMotion Controller:

	Digi	ital Input Ports		
Resource Name	Resource ID	Constant	Resource Name	Res
Input Port 1	0x01	NIMC_DIGITAL_INPUT_PORT1	Output Port 1	0x81
Input Port 2	0x02	NIMC_DIGITAL_INPUT_PORT2	Output Port 2	0x82
Input Port 3	0x03	NIMC_DIGITAL_INPUT_PORT3	Output Port 3	0x83
Input Port 4	0x04	NIMC_DIGITAL_INPUT_PORT4	Output Port 4	0x84
Input Port 5	0x05	NIMC_DIGITAL_INPUT_PORT5	Output Port 5	0x8
Input Port 6	0x06	NIMC_DIGITAL_INPUT_PORT6	Output Port 6	0x8(
Input Port 7	0x07	NIMC_DIGITAL_INPUT_PORT7	Output Port 7	0x87
Input Port 8	0x08	NIMC_DIGITAL_INPUT_PORT8	Output Port 8	0x8{
Input Port 9	0x09	NIMC_DIGITAL_INPUT_PORT9	Output Port 9	0x8§
Input Port 10	0x0A	NIMC_DIGITAL_INPUT_PORT10	Output Port 10	0x8/
Input Port 11	0x0B	NIMC_DIGITAL_INPUT_PORT11	Output Port 11	0x8[
Input Port 12	0x0C	NIMC_DIGITAL_INPUT_PORT12	Output Port 12	0x8(
Input Port	0x0D	NIMC_DIGITAL_INPUT_PORT13	Output	18x0

13			Port 13	
Input Port 14	0x0E	NIMC_DIGITAL_INPUT_PORT14	Output Port 14	0x8I
Input Port 15	0x0F	NIMC_DIGITAL_INPUT_PORT15	Output Port 15	0x8F
Input Port 16	0x10	NIMC_DIGITAL_INPUT_PORT16	Output Port 16	0x9(
Input Port 17	0x11	NIMC_DIGITAL_INPUT_PORT17	Output Port 17	0x91
Input Port 18	0x12	NIMC_DIGITAL_INPUT_PORT18	Output Port 18	0x92
Input Port 19	0x13	NIMC_DIGITAL_INPUT_PORT19	Output Port 19	0x93
Input Port 20	0x14	NIMC_DIGITAL_INPUT_PORT20	Output Port 20	0x94
Input Port 21	0x15	NIMC_DIGITAL_INPUT_PORT21	Output Port 21	0x9t
Input Port 22	0x16	NIMC_DIGITAL_INPUT_PORT22	Output Port 22	0x9(
Input Port 23	0x17	NIMC_DIGITAL_INPUT_PORT23	Output Port 23	0x97
Input Port 24	0x18	NIMC_DIGITAL_INPUT_PORT24	Output Port 24	0x9{
Input Port 25	0x19	NIMC_DIGITAL_INPUT_PORT25	Output Port 25	0x9§
Input Port 26	0x1A	NIMC_DIGITAL_INPUT_PORT26	Output Port 26	0x9/
Input Port 27	0x1B	NIMC_DIGITAL_INPUT_PORT27	Output Port 27	0x9[
Input Port 28	0x1C	NIMC_DIGITAL_INPUT_PORT28	Output Port 28	0x9(
Input Port 29	0x1D	NIMC_DIGITAL_INPUT_PORT29	Output Port 29	0x9[

Input Port	0x1E	NIMC_DIGITAL_INPUT_PORT30	Output	0x9E
30			Port 30	

Trajectory Parameters

All trajectory parameters for servo and closed-loop stepper axes are expressed in terms of quadrature encoder counts. Parameters for openloop stepper axes are expressed in steps. For servo axes, the encoder resolution in counts per revolution determines the ultimate positional resolution of the axis.

For stepper axes, the number of steps per revolution depends upon the type of stepper driver and motor being used. For example, a stepper motor with 1.8°/step (200 steps/revolution) used in conjunction with a 10x microstep driver would have an effective resolution of 2,000 steps per revolution. Resolution on closed-loop stepper axes is limited to the steps per revolution or encoder counts per revolution, whichever is coarser.

There are two other factors that affect the way trajectory parameters are loaded to the NI motion controller versus how they are used by the trajectory generators: floating-point versus fixed-point parameter representation, and time base.

You can load some trajectory parameters as either floating-point or fixedpoint values. The internal representation on the NI motion controller is always fixed-point, however. This fact is important when working with onboard variables, input, and return vectors. It also has a small effect on parameter range and resolution, as shown in the example in the <u>Velocity</u> in <u>RPM</u> topic.

The second factor is the time base. Velocity and acceleration values are loaded in counts/s, RPM, RPS/sec, steps/s, and so on—all functions of seconds or minutes. But the trajectory generators update target position at the Trajectory Update Rate, which is programmable with the Enable Axes function. This means that the range for these parameters depends on the update rate selected, as shown in the example in the Velocity in RPM topic.

RPM values stored in onboard variables are in double-precision IEEE format (f64). Refer to <u>Onboard Variables and Input and Return Vectors</u>, for information about the number of variables required to hold an RPM value.

NI 73xx Velocity in RPM

NI 73xx Velocity in Counts/s or Steps/s

NI 73xx Acceleration in Counts/s² NI 73xx Acceleration in RPS/s NI 73xx Velocity Override in Percent NI 73xx Arc Angles in Degrees
ValueMotion to NI-Motion Conversion

To aid in converting a ValueMotion application to NI-Motion, the following table lists each function in the ValueMotion API and gives the nearest NI-Motion function. Some ValueMotion functions are related to two NI-Motion functions. In these cases, the first function is the closest match to the ValueMotion function, but the second NI-Motion function, in combination with the first, is required to completely mimic the functionality of the original ValueMotion function.

Refer to the individual functions in both the ValueMotion and NI-Motion function online help for detailed information about the functional and syntactic differences.

Equivalent NI-Motion Function	Descriptive Function Name
flex_acquire_trajectory_data	Acquire T
flex_begin_store	Begin Pro
flex_communicate	Commun
flex_enable_breakpoint	Enable B Output
flex_configure_breakpoint	Configure
flex_wait_on_event	Wait on E
flex_enable_axis_limit	Enable A
<u>flex_enable_home_inputs</u>	Enable H
flex_jump_on_event	Jump on
flex_end_store	End Prog
flex_find_reference	Find Refe
flex_find_reference	Find Refe
Not Supported	—
flex_flush_rdb	Flush Ret
flex_get_motion_board_info	Get Motic Informatic
flex_get_motion_board_info	Get Motic Informatic
e <u>flex_get_motion_board_nam</u>	e Get Motic
	Equivalent NI-Motion Function flex_acquire_trajectory_data flex_begin_store flex_communicate flex_enable_breakpoint flex_enable_breakpoint flex_enable_breakpoint flex_enable_axis_limit flex_enable_axis_limit flex_enable_home_inputs flex_jump_on_event flex_end_store flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference flex_find_reference

and about move complete statue. Check

n to NI-Motion Conversion

ng a ValueMotion application to NI-Motion, the ts each function in the ValueMotion API and gives the n function. Some ValueMotion functions are related to nctions. In these cases, the first function is the closest ieMotion function, but the second NI-Motion function, in the first, is required to completely mimic the e original ValueMotion function.

idual functions in both the ValueMotion and NI-Motion slp for detailed information about the functional and ces.

Ec NI Fi	quivalent -Motion unction	Descri Functi Name	iptive on
	flex_acquire_trajectory_data		Acquire Trajectory Data
	flex_begin_store		Begin Program Storage
	flex_communicate		Communicate
	flex_enable_breakpoint		Enable Breakpoint Output
	flex_configure_breakpoint		Configure Breakpoint
	flex_wait_on_event		Wait on Event
	flex_enable_axis_limit		Enable Axis Limits
	flex_enable_home_inputs		Enable Home Inputs
	flex_jump_on_event		Jump on Event
	flex_end_store		End Program Storage
	flex_find_reference		Find Reference
	flex_find_reference		Find Reference
	Not Supported		_
	<u>flex_flush_rdb</u>		Flush Return Data Buffer
	flex_get_motion_board_info		Get Motion Board Information
rd_info	flex_get_motion_board_info		Get Motion Board Information
rd_name	flex_get_motion_board_name	<u>9</u>	Get Motion Board Name
	flow aboald move consulate	ototuc	Chaoly Mayo Complete

Function Execution Times

For selected function execution times, select your controller from the following list:

- <u>NI 7330</u>
- <u>NI 7334</u>
- <u>NI 7340</u>
- <u>NI 7342</u>
- <u>NI 7344</u>
- <u>NI 7350</u>
- <u>NI 7390</u>

Communication between the Host Computer and the NI Motion Controller

The host computer communicates with the NI motion controller through a number of I/O port addresses on the ISA or PCI bus.

At the controller's base address is the primary bidirectional data transfer port. This port supports FIFO data passing in both send and readback directions. The NI motion controller has both a command buffer for incoming commands and a Return Data Buffer (RDB) for return data.

At offsets from the controller's base address are two read-only status registers. The flow of communications between the host and the NI motion controller is controlled by handshaking bits in the Communications Status Register (CSR). The MCS register provides instantaneous motion status of all axes.

Board Identification Parameter

Packets, Handshaking, and FIFO Buffers

Return Data Buffer

Board Identification Parameter

The first parameter to every NI-Motion function is the board identification parameter.

For C/C++ users, all functions in flexmotn.h have BOARD as the first parameter. BOARD is defined as boardID (u8)—the board identification number assigned by Measurement & Automation Explorer (MAX).

For Visual Basic users, all functions in flexmotn.bas have boardID (Integer) as the first parameter, which is the board identification number assigned by Measurement & Automation Explorer.

Make sure you pass the correct board identification parameter for the programming language you are using. To use multiple NI-Motion devices in one application, pass the appropriate board identification parameter to each function.

Packets, Handshaking, and FIFO Buffers

This topic briefly describes how commands and data are passed between the host computer and the NI motion controller. This information is provided for reference purposes. The NI-Motion software provides drivers, DLLs, and C function libraries that handle the host-to-controller communications for you.

Data passed to or from the NI motion controller is handled in a packet format. A packet consists of a packet identifier word, command and data content, and a packet terminator word. This approach to communications enhances the integrity of data communications, speeds the processing of the transferred command and data, and organizes operation into powerful, high-level functions.

Each word in a packet is sent over the bus after checking the Ready-to-Receive (RTR) handshaking bit in the CSR. Refer to the <u>Read</u> <u>Communication Status</u> function for the status cluster and more information about the status reported in the CSR.

Command and data packets are checked for packet format errors as the controller receives them. If the controller detects a packet error, it immediately reports the error by setting an error bit in the CSR. After the packet is received without error, the command and data is stored in a FIFO buffer.

This FIFO can hold up to 16 commands. The NI-Motion RTOS processes commands when it is not busy with higher priority tasks. In the unlikely event that the FIFO fills up before any commands can be processed, the host is held off with a Not-Ready-to-Receive condition.

Each command is processed and a determination is made either to execute the command immediately, or store it away in a program to be executed later. Commands are also checked for data and modal (sequence) errors at this time. Modal errors are flagged by setting the Error Message bit in the CSR. This modal error is functionally different from the packet communication error previously described. Refer to Errors and Error Handling for more information.

Return Data Buffer (73xx Controllers Only)

Data or status requested by the host is buffered in the Return Data Buffer (RDB) in FIFO mode. The FIFO is 16 commands deep.

When data exists in the RDB, the Ready to Send bit in the Communication Status Register (CSR) is set.

You can use the RDB either as a temporary buffer that holds a single return data packet, or as a small FIFO buffer. When the requested data is available in the RDB, use <u>Read Return Data Buffer</u> to retrieve it. You also can request multiple data items and leave them in the buffer for retrieval at a later time. If the RDB becomes full and cannot accommodate requested return data, NI-Motion generates an error and sets the Error Message bit in the CSR.

Errors and Error Handling

To minimize the possibility of erroneous system operation, functions, packets, and data are checked for errors at multiple levels within the NI-Motion software and within the firmware that resides on the NI motion controller.

In a perfect system, errors should not be generated. However, during application development and debugging, errors are unfortunately quite common. NI-Motion offers an extensive error handling structure and utility functions to allow you to quickly diagnose any error-generating situation. Refer to the Error & Utility Functions for information about specific functions and Error Codes for a listing of errors returned by NI-Motion.

Error Codes

The following table lists all NI–Motion error codes, symbolic names, and a brief description of the associated error conditions.

Refer to the <u>Error & Utility Functions</u> for detailed descriptions of the functions for error handling. Refer to <u>Errors and Error Handling</u> for more information about errors and error handling techniques.

Error Code	Symbolic Name	Description
0	NIMC_noError	No error.
-70001	NIMC_readyToReceiveTimeoutError	Ready to Receive Timeout. controller not ready receive commanc the specil timeout p This error occur if th controller processin previous commanc this error persists, (when the controller not be bu contact N
-70002	NIMC_currentPacketError	Either this function is supported type of cc or the cor received incomplet command and cann

avacuta t

Codes

wing table lists all NI–Motion error codes, symbolic names, and escription of the associated error conditions.

the Error & Utility Functions for detailed descriptions of the s for error handling. Refer to Errors and Error Handling for more on about errors and error handling techniques.

ode	Symbolic Name	Description
	NIMC_noError	No error.
	NIMC_readyToReceiveTimeoutError	Ready to Receive Timeout. The controller is still not ready to receive commands after the specified timeout period. This error may occur if the controller is busy processing previous commands. If this error persists, even when the controller should not be busy, contact National Instruments.
	NIMC_currentPacketError	Either this function is not supported by this type of controller, or the controller received an incomplete command packet and cannot

avacuta tha

Error Handling Techniques

Be sure to constantly watch for error conditions. Always check the function return status for a non-zero error code, and react to the error as appropriate. These non-modal errors are easily handled like any other function error in C/C++ or other programming languages. The NI-Motion software includes a utility function, <u>Get Error Description</u>, which you can use to create error description strings for display.

Modal error handling is a bit more involved. Because these errors are rare but can occur at any time, design the application to check for modal errors at various intervals. This is done by calling the <u>Read</u> <u>Communication Status</u> function and checking the Error Message bit. How often to check for modal errors depends upon the application, but you can use the following list as a guideline:

- 1. Check for modal errors at the end of each major subroutine or functional block.
- 2. Check for modal errors at the end of an initialization procedure. Also, check after each axis initialization. Always check, after executing a Find Reference function, to make sure the sequence completed successfully.
- 3. Include a modal error check in every status polling loop. Most applications include a polling loop to display motion status, position, velocity, and so on. This way you are assured of never missing a modal error.
- 4. You can always add a modal error check after each NI-Motion API call, but that is inefficient and unnecessary. Remember that a correctly written program does not generate errors.
- 5. During debugging, you can run an independent application to check for modal errors. The MAX interactive panels always check modal errors for you when MAX is running.

Refer to the example programs, included with the NI-Motion software, to see how error handling is implemented in practice.

Error Message Stack

Modal errors generate an error message containing the command ID, resource ID, and error code that is pushed on the Error Message Stack on the NI motion controller and flagged in the Error Message (Err Msg) bit of the Communications Status Register (CSR). You can return a modal error message to the host by executing the <u>Read Error Message</u> function.

The Error Message Stack is organized as a last-in-first-out (LIFO) buffer so that the most recent error is available immediately. Older errors can be read with additional calls to the Read Error Message function and are returned in the inverse order to which they were generated. When the stack is empty, the Error Message (Err Msg) bit in the CSR is reset.

The Error Stack can hold up to 10 errors. When the stack is full, which is an unlikely event, additional error messages are thrown away.

Fatal Hardware and Communications Errors

There are a few errors that, if detected, indicate a severe or fatal error condition. These include but are not limited to NIMC_boardFailureError, NIMC_watchdogTimeoutError, NIMC_FPGAProgramError, NIMC_DSPInitializationError, NIMC_IOInitializationError, and NIMC_readyToReceiveTimeoutError.

Refer to <u>Error Codes</u> for a complete list of error codes and possible causes.

Fatal errors are unlikely, but if they occur, try to clear them by resetting or power cycling. If this procedure does not clear the problem, refer to <u>Technical Support and Professional Services</u>.

Modal and Non-Modal Errors

NI-Motion can detect two types of errorsmodal and non-modal. Nonmodal errors are errors detected at the time of function execution. This includes communications failures, parameter value errors, and bad Board ID error.

Each NI-Motion function returns a status that indicates if the function executed successfully. A non-zero return status indicates that the function failed to execute and the status value returned is the non-modal error code.

Modal errors, on the other hand, are errors that are not detected at the time of function execution. Because functions can be buffered in the onboard FIFO, it is not possible to detect all potential errors at the time of function execution. Furthermore, some functions can be legal at one time and illegal at another, depending on the state or mode of the NI motion controller. All errors of these types are classified as modal errors. This modal error structure also correctly detects errors generated by incorrectly sequenced functions in onboard programs.

Function Types and Parameters

In addition to the API functional organization, NI-Motion functions can be categorized by common format, execution, and parameter characteristics.

Bitmapped versus Per-Resource Functions

Single and Double-Buffered Parameters

Input and Return Vectors

Onboard Variables

Bitmapped versus Per-Resource Functions

There are two basic types of NI-Motion functions—those that operate on one resource at a time, and those that operate on multiple axes, vector spaces, I/O bits, and so on simultaneously.

Per-resource functions typically send numeric values to, or read numeric values from, the selected axis or resource. They operate identically on each axis or member in the resource family.

In contrast, functions that operate on multiple bits send and return bitmaps, where each bit (axis, vector space, I/O bit, and so on) is represented by one bit in the bitmap.

Some functions set and reset bits in the bitmap using the Must On/Must Off (MOMO) protocol. This tri-state protocol allows you to set/reset one or more bits without affecting the other bits in the bitmap. Refer to any of the MOMO function descriptions for complete information about this protocol.

Bitmapped functions act on all bits simultaneously. Do not use these functions incrementally because each execution completely reconfigures all bits in the bitmap.

Note Bitmapped functions can only be used for simultaneous action involving axes one through fifteen.

Reflective Memory Functions

Some of the <u>Advanced Read</u> functions use reflective memory to return status and data information to the controller. Unlike the <u>trajectory control</u> functions that also return this information, these specialized functions read status and data information from the register and send it back in a much higher response time.

These functions should *not* be used for event checking. Only use the information returned by these functions for display purposes. This is because the tradeoff for significantly faster response time is that these functions are not serialized with other operations. Functions that return data information are read from a register on the controller and refreshed every 5 ms. Functions that return status information are refreshed as soon as possible. This configuration makes the data and status information quickly accessible—in about 40 μ s as opposed to > 2 ms with the trajectory control functions. However, this also means that you can read status information sooner than the function has time to execute. For example, a call to Start Motion followed immediately by a call to the **TnimcStatusMoveCompleteStatus** attribute of the Read Axis Status function may return a True value because the Start Motion function may not have started before the move complete status information is returned.

Note Functions using reflective memory are not supported by the NI SoftMotion Controller.

The following table lists the functions using reflective memory, which controller has reflective memory support, and which attributes use reflective memory, if applicable:

Function	7330	7340	7350	7390
Read All Axis Data			Y	Y
Read All Axis Status			Y	Y
Read Axis Data (all attributes)			Y	Y
Read Axis Status (all attributes)		—	Y	Y
Read Capture Compare Data				
Captured Position			N	N

Position Capture Occurred			Y	Y
Position Compare Occurred		—	Y	Y
Read Coordinate Data (all attributes)			Y	Y
Read Coordinate Status (all attributes)			Y	Y
Read Coordinate Position (all attributes)	—	—	Y	Y
Read Digital I/O Data				
Output Active State	N	N	N	N
Input Active State	Ν	N	N	N
Output Active	N	N	Y	Y
Input Active	N	N	Y	Y
Configure As Input	Ν	Ν	Ν	
Read Encoder Data (all attributes)			Y	Y
Read Motion I/O Data				
Forward Limit Active				
Reverse Limit Active	Ν	Ν	Y	Y
Forward Software Limit Active	Ν	Ν	Y	Y
Reverse Software Limit Active	Ν	Ν	Y	Y
Home Input Active	Ν	Ν	Y	Y
Inhibit-In Active	Ν	Ν	Y	Y
In-Position Active	Ν	Ν	Ν	Ν
Drive Ready Active	Ν	Ν	Ν	Ν
Inhibit-Out Active	Ν	Ν	Ν	Ν
	Ν	Ν	Ν	Ν

Considerations when Using Read Functions

Read functions return data from the NI motion controller. There are two types of Read functions. The first takes a return vector and returns the data to a general-purpose variable in onboard memory, or the Return Data Buffer (RDB). This type of function is typically used in conjunction with onboard programming. The second type returns the data by reference into a variable in the application. The functions that return data by reference have a suffix _rtn. This is the more commonly used type of function. Refer to Input and Return Vectors for more information about return vectors.

Example

The return vector version of the <u>Read Position</u> function has the following function prototype:

status = flex_read_pos (boardID, axis, returnVector)

When you use this function, the data retrieved by the controller is placed into the general-purpose variable indicated by the **returnVector** parameter. If the **returnVector** is 0xFF (for Visual Basic users, &HFF), the return data is placed in the Return Data Buffer (RDB) for later retrieval.

The _rtn version of the <u>Read Position</u> function has the following function prototype:

status = flex_read_pos_rtn (boardID, axis, position)

where **boardID** and **axis** are inputs and **position** is an output. When you use this function, the NI-Motion software places the data retrieved from the controller into the **position** variable that you referenced when calling the function. For C/C++ users, **position** is a pointer to an i32. For Visual Basic users, **position** is of type long and the pass-by-reference behavior is made clear to the compiler by the function prototype in the Flexmotn.bas header file.

Single and Double-Buffered Parameters

Almost all NI-Motion parameters are either single-buffered or doublebuffered on the controller. Single-buffered parameters take effect immediately upon function execution and remain in effect until they are overwritten with another call to the function that loaded or set them. It is not necessary to constantly reload single-buffered parameters each time you deal with an axis, vector space, or resource. The obvious exception to this is action commands like <u>Start Motion</u>, <u>Stop Motion</u>, and so on, which must be called each time.

Most trajectory control parameters are double-buffered. You can load these parameters on the fly without affecting the move in process. They do not take effect until the next Start or Blend Motion function is executed. Like single-buffered parameters, the controller retains the values so they do not have to be loaded before each move unless you want to change their values.

Load Breakpoint Position and Load Breakpoint Modulus are the only nontrajectory parameters that are also double-buffered. They have no effect until you execute a subsequent Enable Breakpoint Output function.

Input and Return Vectors

Many functions that load values and virtually all readback functions support vectoring. Load functions, for example Load Target Position, take an input vector that specifies the source of the data, either immediate (within the function call), from a general-purpose onboard variable, or from an indirect variable. Read functions, for example Read Position, take a return vector that specifies the destination for the returned data, either the host computer, an onboard variable, or an indirect variable.

The ability to use variables in motion control functions is one of the powerful features of the NI-Motion onboard programming environment. You can read data from a resource into a variable, scale or perform some other calculation on the value, and then load the new value as a trajectory or other motion parameter. All <u>data operations functions</u> take data from variables and return the result through a return vector, typically to another variable.

 $\overline{\mathbb{N}}$

Note Data returned to the host by a return vector of 0xFF is actually left in the Return Data Buffer (RDB). Read data from the RDB using <u>Read Return Data Buffer</u>.

In addition to specifying a variable directly, you can use indirect variables to reference a variable indirectly, much like a pointer in C. Refer to <u>Onboard Variables</u> for more information.

Input and return vectors are very useful when writing onboard programs but have little or no use in programs running on the host computer. For this reason, the default value for input vector is immediate (0xFF) and the API includes a second _rtn version for all Read functions. This version automatically retrieves the data from the RDB after requesting it and returns it by reference to the output parameter.

 $\overline{\mathbb{N}}$

Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no return vector is required.

The return vector must be 0xFF for 7330 and 7390 controllers, because they do not support onboard programming.

Refer to <u>Communication between the Host Computer and the NI-Motion</u> <u>Controller</u> for information about the RDB.

Onboard Variables

NI-Motion supports 120 general-purpose variables (0x01 through 0x78) for use in onboard programs. Variables are 32 bits wide and can hold either signed (i32) or unsigned (u32) values. Variables can be referenced in input and return vector parameters. <u>Data Operations</u> functions use variables exclusively for input operands and the output result.

In general, most functions have a single data parameter that fits into a single 32-bit variable. If the function uses only a 16-bit data value, it is in the least significant portion (right-most) of the 32-bit variable. However, some functions with input or return vectors have more than one data parameter. As a general rule, each parameter, regardless of size, requires its own variable. For these functions, the vector points to the first variable in a sequential group of variables. Parameters are then associated with variables in sequential order.

A few advanced functions handle variables differently from the previous description. Refer to <u>Using Inputs and Return Vectors with Onboard</u> <u>Variables</u> for more information about variables and vectors.

Indirect Variables

Indirect variables provide functionality similar to pointers in a programming language such as C. When you use an indirect variable (0x81 through 0xF8) as an input vector or return vector, the NI-Motion software looks at the contents of the indirect variable and uses that value as the input or return vector. Indirect variables 0x81 through 0xF8 correspond to variables 0x01 through 0x78. So when you specify an input vector of 0x81, the NI motion controller uses the contents of variable 0x01 as the input vector. This can be very useful in looping structures in onboard programs, as well as changing the input values to functions dynamically.

Example

Suppose the values of onboard variables 1-10 are as follows:

0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A

0 7 0 0 0 0 1000 0 0

In this example, reading the indirect variable 0x82 returns the value 1000.

Note You can save the entire set of onboard general-purpose



variables to Flash ROM with the <u>Save Default Parameters</u> functions.

Variable Array

Variable arrays are used when reading or writing a buffer of data to or from onboard memory using the <u>Read Buffer</u> or <u>Write Buffer</u> functions.

In memory, the buffer occupies n + 2 variables, where n is the number of buffer elements read. The first variable contains the number of values read from the buffer. The second variable contains the error code of the read operation. If this value is zero, then no error occurred. The next n elements are the values read from the array. The read buffer accounts for the two additional values. However, when you are writing a buffer using a variable array, the second parameter is the regeneration mode, not an error code.

Example

Suppose the onboard variable contains the following information:

0x1 0x2 0x3 0x4 0x5 0x6 0x7 0x8 ...

5 0 0 0 0 0 0 0 ...

Further, suppose that there is a buffer containing the following elements:

12345

If you were to read the first five elements of the buffer to onboard memory location 0x81, the onboard memory would look like the following:

0x1 0x2 0x3 0x4 0x5 0x6 0x7 0x8 0x9 0xA 0xB 0xC ...

5 0 0 0 5 0 1 2 3 4 5 0 ...

0x5 contains the number of elements to read. 0x6 displays the errors that occurred. In this case, the location contains a 0, meaning no errors occurred. 0x7 through 0xB are the five elements that were read.

Considerations when Using Functions with Input Vectors

NI-Motion functions that take an input vector allow the data for the function to be loaded from different sources. The inputVector argument in a function tells the NI motion controller to take the function data either from the host computer or from an onboard general-purpose variable. An input vector of 0xFF (for Visual Basic users, &HFF) causes the NI motion controller to get the data from the host computer, in other words, from the data parameters in the function call. Refer to Input and Return Vectors for more information about input vectors.

Using Inputs and Return Vectors with Onboard Variables

The following table provides additional information about how to use input and return vectors in conjunction with onboard variables. The table contains functions with more than one data parameter that require multiple variables when using vectors. The Maximum Variable Number column lists the highest variable number that a vector can point to and still have room for all the data without exceeding the total variable space. This table also contains notes that highlight a few special cases where the number of parameters does not equal the number of variables.

Function Name	Vector Type	Number of Parameters	Number (Variables Required	of Maximum Variable Number	Comments
Load All PII Parameters	D Input	8	8	0x71	
<u>Load</u> <u>Velocity</u> <u>Filter</u> Parameter	Input	2	2	0x77	
<u>Load</u> <u>Velocity in</u> <u>RPM</u>	Input	1	2 (<u>Note 1</u>)	0x75	National Instruments recommends using Load Velocity for onboard variables.
<u>Load</u> <u>Accel/Dece</u> in RPS/sec	Input <u>I</u>	1	2 (<u>Note 1</u>)	0x75	
<u>Load Vecto</u> <u>Space</u> Position	<u>r</u> Input	3	3	0x76	
<u>Read</u> <u>Velocity in</u> <u>RPM</u>	Return	1	2 (<u>Note 1</u>)	0x75	National Instruments recommends

NI-Motion Windows Libraries

This section contains general information about building NI-Motion applications, describes the nature of the NI-Motion files used in building NI-Motion applications, provides information specific to each language environment as well as general syntax information that applies to all languages, and explains the basics of building 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 the development tool reference manual for details on creating applications that call DLLs.

The NI-Motion DLL, FlexMotion32.dll, is used by NI-Motion applications under all versions of Windows.

If you are programming in C or C++, you must link in the appropriate import library so that you can call the NI-Motion DLL. In Microsoft Windows Vista/XP/2000, the import libraries are different for Microsoft and Borland C/C++. The import libraries contain information about the NI-Motion DLL-exported functions.

NI-Motion is packaged with function prototype files for different Windows development tools. For C/C++ development, the NI-Motion header file, flexmotn.h, is provided. For Visual Basic development, a BAS module, flexmotn.bas, is provided. If you are not using any of these development tools, you may need to create your own function prototype file based on the files provided with the NI-Motion software.

Variable Data Types

Every function description has a parameter table that lists the data types for each parameter. Refer to <u>Primary Types</u>, <u>Arrays</u>, <u>Structures and Other</u> <u>User-Defined Data Types</u>, and <u>Function Return Status</u>, for notation descriptions used in those parameter tables and throughout the documentation for variable data types.

Primary Types

The following table shows the primary type names and their ranges:

Туре	Description	Range	C/C++	Visual BASIC	Pas
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%)	Sma
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. Refer to the i16 description.	Wor
i32	32-bit signed integer	-2,147,483,648 to 2,147,483,647	Long	Long (for example, count&)	Lon(
u32	32-bit unsigned integer	0 to 4,294,967,295	Unsigned long	Not supported by BASIC. For functions that require	Carc 32-b oper syst Refe

				unsigned long integers, use the signed long integer type instead. Refer to the i32 description.	the i desc
f32	32-bit single- precision floating point	-3.402823 × 10 ³⁸ to 3.402823 × 10 ³⁸	Float	Single (for example, num!)	Sing
f64	64-bit double- precision floating point	-1.797683134862315 × 10 ³⁸ to 1.797683134862315 × 10 ³⁸	Double	Double (for example, voltage Number)	Dou

Arrays

When a primary type is inside square brackets, such as, [i16], an array of the type named is required for that parameter. Typically, arrays are passed by reference, not value. Refer to <u>Language-Specific</u> <u>Considerations</u> for information about passing and returning arrays of data.

Structures and Other User-Defined Data Types

NI-Motion software uses data structures to send and receive groups of parameters to and from the controller. Typically, data structures are passed by reference, not value. Refer to <u>Language-Specific</u> <u>Considerations</u> for information about passing and returning structures of data.

Function Return Status

Every NI-Motion function is of the following form:

status = function_name (parameter 1, parameter 2, parameter n)

Each function returns a value in the status variable that indicates the success or failure of the function. A returned status of NIMC_noError (0) indicates that the function was sent to the NI motion controller successfully. A non-zero status indicates that the function was not executed because of an error. Refer to Error Codes for a complete description of errors and possible causes.

In addition to errors returned in the status variable, modal errors can occur when the controller executes the function. These modal errors have to be explicitly read from the Error Stack on the NI motion controller. Refer to <u>Errors and Error Handling</u> for more information about modal errors and the Error Stack.

For C/C++ users, the header file, flexmotn.h provides the NI-Motion function prototypes. All the functions in flexmotn.h have FLEXFUNC as the return status. FLEXFUNC defines the return data type status. In Windows, FLEXFUNC also defines the calling convention of the NI-Motion dynamic link library.

FLEXFUNC defines the return status of the NI-Motion functions to be i32. In addition, FLEXFUNC defines the calling convention as standard (stdcall), which is the same as that used to call Win32 API functions.

For Visual Basic users, flexmotn.bas provides the NI-Motion function prototypes. All NI-Motion functions in Visual Basic return an i32.

Language-Specific Considerations

Apart from the data type differences, there are a few languagedependent considerations you need to be aware of when you use the NI-Motion API. Refer to the appropriate section that applies to the programming language.

Note Be sure to include the NI-Motion function prototypes by including the appropriate NI-Motion header file in the source code. Refer to the <u>NI-Motion Windows Libraries</u> for the header file appropriate to the operating system and programming environment.

Visual Basic for Windows

u8 Data Type Not Supported

Because Visual Basic does not support the u8 data type, all the NI-Motion functions that take parameters that are of type u8 can be passed integers (%). The NI-Motion DLL ignores the upper byte of the integer passed.

For example, the Enable Axis function, which has the data type for **boardID** and **PIDrate** as u8, can be called as shown in the following example.

Example

Dim status as long Dim boardID as integer Dim PIDrate as integer Dim axisMap as integer boardID = 1 The board identification number PIDrate = NIMC_PID_RATE_250 250 microsecond update rate axisMap = &H1E Enable axes 1 through 4 status = flex_enable_axes (boardID, 0, PIDrate, axisMap)

Data Returned by Reference

The NI-Motion functions that return data do so in variables passed into the function by reference.

Example

Dim status as long Dim boardID as integer Dim position as long Dim axis as integer boardID = 1 axis = NIMC_AXIS1 status = flex_read_pos_rtn (boardID, axis, position)

The position for axis one is returned in the position variable.

Data Returned in Arrays

While passing an array to a NI-Motion function in Visual Basic, you need to pass the first element of the array by reference.

Example

You would pass in returnData (0) as the parameter, where returnData (0 to MAX) is an array of MAX longs.

Dim status as long Dim boardID as integer Const MAX = 12 Dim returnData (0 to MAX) as long boardID = 1 status = flex_read_trajectory_data_rtn(boardID, returnData&(0))

Trajectory data is returned in the returnData array and can be accessed by incrementing through the array.

User-Defined Data Types

Two user-defined data types are used by the NI-Motion functions under Visual Basic—the registry information data type registryRecord, and the PID parameters data type PIDVals.

The registry information data type registryRecord is used by the <u>Read</u> <u>Object Registry</u> function to get the information about the object registry on the NI motion controller. Refer to <u>Onboard Programming Functions</u> for more information about the object registry.

Type registryRecord

device As IntegerThe object number type As IntegerThe type of object pStart As LongThe address where the object is stored size As LongSize of the object

End Type

The PID parameters data type PIDVals is used by the <u>Load All PID</u> <u>Parameters</u> function to load the PID and PIVFF parameters for an axis. Refer to <u>Axis & Resource Configuration Functions</u> for more information about PID and PIVFF parameters.

Type PIDVals

kp As IntegerProportional Gain ki As IntegerIntegral Gain ilim As IntegerIntegration Limit kd As IntegerDerivative Gain td As IntegerDerivative Sample Period kv As IntegerVelocity Gain aff As IntegerAcceleration Feedforward vff As IntegerVelocity Feedforward

End Type

Example

While using user-defined data types, pass the data types by reference to the function.

Dim boardID as integer Dim axis as integer Dim pidvalues As PIDVals Dim inputVector as integer pidvalues.kp = 100 pidvalues.ki = 0 pidvalues.ilim = 1000 pidvalues.kd = 1000 pidvalues.td = 2pidvalues.kv = 0 pidvalues.aff = 0pidvalues.vff = 0 boardID =1 axis = NIMC_AXIS1 inputVector = & HFF status = flex_load_loop_params(boardID, axis, pidvalues, inputVector)
C/C++ for Windows

Data Returned by Reference

The NI-Motion functions that return data do so in variables whose address is passed into the function.

Example

To read position on an axis, you have to pass the address of the position variable.

i32 status; i32 position; u8 boardID = 1; u8 axis = NIMC_AXIS1; status = flex_read_pos_rtn (boardID, axis, &position);

The data position for axis one is returned in the position variable.

Data Returned in Arrays

While passing an array to a NI-Motion function, you need to pass the address of the beginning of the array.

Example

You would pass returnData as the parameter where returnData is an array of size MAX i32s.

```
#define MAX 12
i32 status;
u8 boardID;
i32 returnData [MAX];
boardID = 1;
status = flex_read_trajectory_data_rtn(boardID, returnData);
```

Trajectory data is returned in the returnData array and can be accessed by incrementing through the array.

NI-Motion Data Structures

Two data structures are used by the NI-Motion functions—the registry information data structure REGISTRY, and the PID parameters data structure PID.

The registry information data structure REGISTRY is used by the <u>Read</u> <u>Object Registry</u> function to get the information about the object registry on the NI motion controller. Refer to <u>Onboard Programming Functions</u> for more information about the object registry.

typedef struct {

u16 device; // The object number u16 type; // The type of object u32 pstart; // The address where the object is stored u32 size; // Size of the object

} REGISTRY;

The PID parameters data structure PID is used by the <u>Read Object</u> <u>Registry</u> function to load the PID and PIVFF parameter for an axis. Refer to <u>Axis & Resource Configuration Functions</u> for more information on PID and PIVFF parameters.

typedef struct{

u16 kp; //Proportional Gain u16 ki; //Integral Gain u16 ilim; //Integration Limit u16 kd; //Derivative Gain u16 td; //Derivative Sample Period u16 kv; //Velocity Gain u16 aff; //Acceleration Feedforward u16 vff; //Velocity Feedforward

} PID;

Example

While using the data structures, pass the address of the structure in the function.

```
i32 status;
u8 boardID=1;
u8 axis=NIMC_AXIS1;
u8 inputVector=0xFF;
PID PIDValues;
PIDValues.kp = 100;
PIDValues.ki = 0;
PIDValues.ilim = 1000;
```

```
PIDValues.kd = 1000;
PIDValues.td = 2;
PIDValues.kv = 0;
PIDValues.aff = 0;
PIDValues.vff = 0;
status = flex_load_pid_parameters(boardID, axis, &PIDValues, inputVector);
```

Creating 32-Bit Applications

<u>Creating a 32-Bit LabWindows/CVI Application</u> <u>Creating a 32-Bit</u> <u>Microsoft or Borland C/C++ Application</u> <u>Creating a 32-Bit Visual Basic Application</u>

Creating a 32-Bit LabWindows/CVI Application

The NI-Motion header file for LabWindows/CVI programmers is FlexMotn.h. FlexMotn.h automatically includes other required files such as MotnErr.h, which is the NI-Motion error file for errors returned by the NI-Motion functions, and MotnCnst.h, the NI-Motion constants file for the constants used while making calls to the NI-Motion functions. All header files are installed in the NI-Motion\Include directory.

To create an application using LabWindows/CVI, link to the appropriate NI-Motion import library. If you set the default compiler compatibility mode to Microsoft Visual C++, link to FlexMS32.lib, which is installed in the NI-Motion\lib\Microsoft directory. If you set the default compiler compatibility mode to Borland C++, link to FlexBC32.lib library, which is installed in the NI-Motion\lib\Borland directory.

Example programs using these import libraries and the Flexmotion32.dll are in installed in the NI-Motion\Examples directory.

Creating a 32-Bit Microsoft or Borland C/C++ Application

The NI-Motion header file for C/C++ programmers is FlexMotn.h. FlexMotn.h automatically includes other required files such as MotnErr.h, the NI-Motion error file for errors returned by the NI-Motion functions, and MotnCnst.h, the NI-Motion constants file for the constants used while making calls to the NI-Motion functions. All header files are installed in the NI-Motion\Include directory.

Microsoft C/C++ programmers must link to the FlexMS32.lib import library, which is installed in the NI-Motion\lib\Microsoft directory. Borland C/C++ programmers must link to the FlexBC32.lib import library, which is installed in the NI-Motion\lib\Borland directory. The Borland import library is created in Borland version 5.0.

C\C++ example programs using these import libraries and the Flexmotion32.dll are installed in the NI-Motion\Examples directory.

Creating a 32-Bit Visual Basic Application

The NI-Motion function prototype file for Visual Basic programmers is the FlexMotn.bas module. In addition, motnerr.bas, the NI-Motion error file for errors returned by the NI-Motion functions, and motncnst.bas, the NI-Motion constants file for the constants used while making calls to the NI-Motion functions, must also be included in Visual Basic projects when making NI-Motion applications. All the Visual Basic modules are installed into the NI-Motion\Include directory.

Visual Basic example programs are installed in the NI-Motion\Examples directory.

About the NI-Motion Software

NI-Motion software provides a comprehensive API you use to control the NI motion controllers. NI-Motion software combined with the NI motion controllers provide functionality and power for integrated motion systems for use in laboratory, test, and production environments.

For programming ease, NI-Motion software is enhanced by a toolbox of drivers, LabVIEW VI libraries, and Windows dynamic link libraries (DLLs) that implement the entire NI-Motion API.

The NI-Motion software package includes Measurement & Automation Explorer (MAX), which associates physical bus addresses with board IDs, which are used in programs to distinguish between controllers. MAX also verifies that the NI-Motion controller is installed correctly and is communicating with the host computer.

The motion configuration environment under MAX allows you to configure initialization settings that are used to initialize the NI motion controller. MAX also provides an easy interactive environment for testing and troubleshooting the motion system. Refer to the MAX online help for more information.

For application development, the NI-Motion software package includes example programs to help get you up and running quickly.

A separate NI-Motion software package includes a complete motion C library, with examples, for use with C or C++ development environments.

Initialization

The configuration panels in Measurement & Automation Explorer (MAX) allow you to set a large number of NI motion controller attributes such as motor type, limit switch active state, position mode, and maximum velocity. Most of the attributes are one-time configuration settings that does not change. By setting these attributes one time in MAX, your programming is much simpler, and you can concentrate on the motion-specific part of the application.

When you call the Initialize Controller function, all the attributes stored in MAX are retrieved and sent to the motion controller. After the controller has been initialized, you do not need to initialize each time you run the application. National Instruments recommends you call Initialize Controller one time after restarting the computer or resetting the motion hardware. The motors are temporarily disabled (killed) while this initialization process takes place.

In most cases, you call Initialize Controller with the Settings Name terminal unwired, which causes the current initialization settings in MAX to be used. You also can initialize the controller with alternate initialization settings by passing in the name of the alternate setting as specified in MAX.

Initialize Controller

flex_initialize_controller

Device Compatibility

Initialize Controller

Usage

status = flex_initialize_controller(u8 boardID, i8* settingsName);

Purpose

Initializes the specified controller according to the settings as configured in Measurement & Automation Explorer.

Parameters

Name	Туре	Description	
boardID	u8	assigned by Measurement & Automation Explorer (MAX)	
settingsName	[i8]	name of configuration settings	

Parameter Discussion

settingsName is the name of the configuration settings, such as *Default* 7330 Settings, as specified in MAX. To use the current configuration settings, pass in NULL for the **settingsName** parameter. **settingsName** must be a null terminated string.

Using This Function

This function allows you to initialize the controller to a known state based on the configuration settings stored in the MAX database. This function pulls values from the configuration database and calls dozens of NI-Motion functions, configuring the controller for you in one simple step.

If the Initialize Controller function returns an error, you can call the <u>Get</u> <u>Last Error</u> function to determine the specific NI-Motion function and resource generating the error. You can then call the <u>Get Error Description</u> function, which formats a string for you with the function name, resource ID and error description.



Note If you are running a NI-Motion startup application on ETS (PharLap) systems, the first function in your application must be either the Clear Power Up Status or Initialize Controller function. Calling any other function first may result in error –70012 (NIMC_badBoardIDError) because the driver may not have loaded before a NI-Motion function is called.

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

Axis & Resource Configuration

These functions give you access to some of the most powerful features of NI-Motion. They allow you to map encoder, ADC, and DAC resources to various axes, configure axes for servo or stepper control, and combine axes into 2D and 3D vector spaces for advanced motion control applications.

These axis and resource configuration functions are typically executed one time during system initialization. You can call most of them at any time to reconfigure the motion control system on-the-fly. All of the Axis & Resource Configuration functions have default values that provide good starting points for many motion control applications.

Configure Axis Resources Configure Move Complete Criteria Configure Stepper Output Configure Vector Space Enable Axis Load Advanced Control Parameter Load All PID Parameters Load Commutation Parameter Load Counts/Steps per Revolution Load Single PID Parameter Load Velocity Filter Parameter Set Stepper Loop Mode flex_config_axis

Device Compatibility

Configure Axis Resources

Usage

status = flex_config_axis(u8 boardID, u8 axis, u8 primaryFeedback, u8 secondaryFeedback, u8 primaryOutput, u8 secondaryOutput);

Purpose

Configures an axis by defining its feedback and output resources.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to configure
primaryFeedback	k u8 primary encoder or ADC feedback re	
secondaryFeedback	u8	secondary encoder
primaryOutput	u8	primary DAC or step output resource
secondaryOutput	u8	secondary DAC

Parameter Discussion

axis is the axis to configure. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

primaryFeedback is the number for the primary feedback resource being mapped to the axis. The primary feedback resource is used for position feedback, derivative (Kd) damping, and velocity feedback (Kv) if a secondary resource is not configured. Enter 0 (zero) to configure no primary feedback resource.

secondaryFeedback is the number for an optional secondary feedback resource being mapped to the axis. If a secondary feedback resource is mapped, it is used for velocity feedback (Kv). Enter 0 (zero) to configure no secondary feedback resource. Refer to the *Velocity Feedback Gain* section of Load Single PID Parameter for more information about Kv.

primaryOutput is the number for the primary output resource being mapped to the axis. This is the main command output. Enter 0 (zero) to configure no primary output resource.

secondaryOutput is the number for an optional secondary output resource being mapped to the axis. This is an optional command output. Enter 0 (zero) to configure no secondary output resource.

Using This Function

This function defines the feedback and output devices for an axis. You can configure up to two feedback resources and two output resources for each axis. This flexible mapping of resources to axes allows for advanced servo and stepper configurations such as: independent velocity and position feedback devices (dual-loop control), dual DAC outputs with different offsets, and changing feedback on the fly.

The various feedback and output resources on the NI motion controller have different interface, performance, and functionality characteristics. This function allows you to define the axis and tailor its performance.

Resource Type	7330	7340	7350	7390
Primary Feedback	Encoder/ADC	Encoder/ADC	Encoder/ADC	Encoder
Secondary Feedback	Encoder	Encoder	Encoder/ADC	—
Primary Output	Stepper	Stepper/DAC	Stepper/DAC	Stepper
Secondary Output		DAC	DAC	

The following table lists valid resources:

Tip Refer to Motion Resources for more information about resource IDs.

This function must be called for each axis that is used by an application prior to enabling the axis. The factory default mapping of resources to axes is as follows:

Axis	Prim	ary Feedback	Cocordomy	Primary Output		
	Resource ID	Constant	Feedback	Resource ID	Constant	
1	0x21 (Enc 1)	NIMC_ENCODER1	0	0x31 (DAC 1)	NIMC_DAC	
2	0x22 (Enc 2)	NIMC_ENCODER2	0	0x32 (DAC 2)	NIMC_DAC	
3	0x23 (Enc 3)	NIMC_ENCODER3	0	0x33 (DAC 3)	NIMC_DAC	
4	0x24 (Enc	NIMC_ENCODER4	0	0x34	NIMC_DAC	

	4)			(DAC 4)	
5	0x25 (Enc 5)	NIMC_ENCODER5	0	0x35 (DAC 5)	NIMC_DAC
6	0x26 (Enc 6)	NIMC_ENCODER6	0	0x36 (DAC 6)	NIMC_DAC
7	0x27 (Enc 7)	NIMC_ENCODER7	0	0x37 (DAC 7)	NIMC_DAC
8	0x28 (Enc 8)	NIMC_ENCODER8	0	0x38 (DAC 8)	NIMC_DAC
9	0x29 (Enc 9)	NIMC_ENCODER9	0	0x39 (DAC 9)	NIMC_DAC
10	0x3A (Enc 10)	NIMC_ENCODER10	0	0x4A (DAC 10)	NIMC_DAC
11	0x3B (Enc 11)	NIMC_ENCODER11	0	0x4B (DAC 11)	NIMC_DAC
12	0x3C (Enc 12)	NIMC_ENCODER12	0	0x4C (DAC 12)	NIMC_DAC
13	0x3D (Enc 13)	NIMC_ENCODER13	0	0x4D (DAC 13)	NIMC_DAC
14	0x3E (Enc 14)	NIMC_ENCODER14	0	0x4E (DAC 14)	NIMC_DAC
15	0x3F (Enc 15)	NIMC_ENCODER15	0	0x4F (DAC 15)	NIMC_DAC
16	0xC1 (Enc 16)	NIMC_ENCODER16	0	0xD1 (DAC 16)	NIMC_DAC
17	0xC2 (Enc 17)	NIMC_ENCODER17	0	0xD2 (DAC 17)	NIMC_DAC
18	0xC3 (Enc 18)	NIMC_ENCODER18	0	0xD3 (DAC 18)	NIMC_DAC
19	0xC4 (Enc 19)	NIMC_ENCODER19	0	0xD4 (DAC 19)	NIMC_DAC
20	0xC5 (Enc 20)	NIMC_ENCODER20	0	0xD5 (DAC 20)	NIMC_DAC

21	0xC6 (Enc 21)	NIMC_ENCODER21	0	0xD6 (DAC 21)	NIMC_DAC
22	0xC7 (Enc 22)	NIMC_ENCODER22	0	0xD7 (DAC 22)	NIMC_DAC
23	0xC8 (Enc 23)	NIMC_ENCODER23	0	0xD8 (DAC 23)	NIMC_DAC
24	0xC9 (Enc 24)	NIMC_ENCODER24	0	0xD9 (DAC 24)	NIMC_DAC
25	0xCA (Enc 25)	NIMC_ENCODER25	0	0xDA (DAC 25)	NIMC_DAC
26	0xCB (Enc 26)	NIMC_ENCODER26	0	0xDB (DAC 26)	NIMC_DAC
27	0xCC (Enc 27)	NIMC_ENCODER27	0	0xDC (DAC 27)	NIMC_DAC
28	0xCD (Enc 28)	NIMC_ENCODER28	0	0xDD (DAC 28)	NIMC_DAC
29	0xCE (Enc 29)	NIMC_ENCODER29	0	0xDE (DAC 29)	NIMC_DAC
30	0xCF (Enc 30)	NIMC_ENCODER30	0	0xDF (DAC 30)	NIMC_DAC

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

- If you are using an NI 73*xx* motion controller, you cannot configure an axis when any axes are enabled. You must first disable all axes using the <u>Enable Axis</u> function.
- To change feedback on the fly (NI 7350 controllers only), call this function when the axis is not moving (move complete).

Note The feedback resources used for switching feedback on the fly must meet the following conditions:

- The feedback device must be enabled.
- The feedback device being switched to must not be used by another axis.

Refer to <u>Switching Feedback on the Fly</u> in the *NI-Motion Help* for detailed information about configuring your system to switch between two feedback devices.

Example 1

To change axis 3 to use the fourth encoder channel and the first DAC output, call the Configure Axis Resources function with the following configuration:

Axis	PrimaryFeedback	SecondaryFeedback	Primary Output	Secondary
3	NIMC_ENCODER4	0	NIMC_DAC1	0

To avoid potential contention for output resources, this function always honors the configuration of the last time it is called. In this example, both axis 1 (by default) and axis 3 want to use DAC 1. Similarly, both axis 4 (by default) and axis 3 want to use encoder 4. To avoid contention, DAC 1 is assigned to axis 3 and removed from axis 1, and encoder 4 is assigned to axis 3 and removed from axis 4, resulting in the following configuration:

Axis	PrimaryFeedback	SecondaryFeedback	Primary Output	Secondary

1	NIMC_ENCODER1	0	0	0
4	0	0	NIMC_DAC4	0

You must now call this function again to configure axis 4 with a different feedback resource and axis 1 with a different output resource.

Example 2

To configure axis 2 for dual-loop control you can use the following parameters:

Axis	PrimaryFeedback	SecondaryFeedback	Primary Output	Secondary
2	NIMC_ADC1	NIMC_ENCODER2	NIMC_DAC2	0

In this example, an ADC channel is used for the primary position feedback (kp, ki, kd) while an encoder is used for the secondary velocity feedback. This application typically uses velocity feedback (Kv) from the encoder for stability. Refer to <u>Load Single PID Parameter</u> for information about setting Kv.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- When you are using the NI SoftMotion Controller, you cannot map an axis resource to another axis. For example, encoder 3 (NIMC_ENCODER3) cannot be mapped to axis 1.
- To enable dual-loop feedback on the NI SoftMotion Controller, your motion control system must include a second encoder on the axis. Otherwise, an error is returned. To configure the secondary encoder for axis 1, set **secondaryFeedback** to NIMC_SECONDARY_ENCODER1. To enable dual-loop feedback on axis 1, set **primaryFeedback** to NIMC_ENCODER1 and set the **secondaryFeedback** to NIMC_SECONDARY_ENCODER1.
- The NI SoftMotion Controller supports secondary feedback devices on axes 1 to 15 only.

- Secondary Encoders

Use secondary encoders in a dual-loop feedback system to provide a more accurate velocity feedback estimation.

Resource Name	Resource ID	Constant
Secondary Encoder 1	0X71	NIMC_SECONDARY_ENCODER1
Secondary Encoder 2	0X72	NIMC_SECONDARY_ENCODER2
	•	•
	•	-
	•	•
Secondary Encoder 15	0X7F	NIMC_SECONDARY_ENCODER15

- Secondary ADCs

Resource Name	Resource ID	Constant
Secondary ADC 1	0X91	NIMC_SECONDARY_ADC1
Secondary ADC 2	0X92	NIMC_SECONDARY_ADC2
•	•	
•	•	•
•		•
Secondary ADC 15	0X9F	NIMC_SECONDARY_ADC15

flex_config_mc_criteria

Device Compatibility

Configure Move Complete Criteria

Usage

status = flex_config_mc_criteria(u8 boardID, u8 axis, u16 criteria, u16 deadBand, u8 delay, u8 minPulse);

Purpose

Configures the criteria for the Move Complete status to be True.

Parameters

Name	Туре	Description							
boardID	u8	assigned by Measurement & Automation Explorer (MAX)							
axis	u8	axis to configure							
criteria	u16	conditions that must be met for the MC status to be True							
deadBand	u16	tolerance area around target position							
delay	u8	settling time delay, in ms							
minPulse	u8	minimum time the MC status must stay true, in ms							

Parameter Discussion

axis is the axis to configure. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

criteria is the bitmap that defines the criteria for the move complete status to be True:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D(
0	0	0	0	0	0	0	0	0	0	0	ln Pos	Delay	Stop	MOff	P(
											F U 3				

D0 Profile Complete (PC):

1 = Profile must be complete (default)

0 = N/A (cannot reset)

D1 Motor Off (MOff):

1 = Motor is off or killed

0 = Motor off status not considered (default)

D2 Stop:

- 1 = Run/Stop must be stopped
- 0 = Run/Stop not considered (default)

D3 Delay:

1 = Move complete only after delay

0= Move complete not delayed (default)

D4 In-Position (In Pos):

1 = Must be within deadband of target position

0 = Ignore in-position status (default)

The effect of the criteria parameters can be summarized with the following equation:

```
Move Complete = (Profile Complete [OR Motor Off])
[AND (Run/ Stop == Stop)]
```

[AND (Delay == Done)] [AND (| position - target position | < deadband)] where [...] = optional criteria.

deadband is the tolerance around the target position, and has a range of 0 (default) to 32,767. If selected, the move is only considered complete when | position - target position | < **deadband**.

delay is a programmable settling time delay in ms. You can program it from 0 (default) to 255 ms.

minPulse is the minimum time, in ms, that the move complete status must stay true. This parameter allows you to enforce a minimum pulse width on the move complete status even if the axis is started again. This is the amount of time you have to read the move complete status before it is reset. The range is 0 (default) to 255 ms.

Using This Function

This function defines the conditions for reporting a move complete. When a move is complete on an axis, the corresponding bit in the Move Complete Status (MCS) register is set. Refer to the <u>Read Move Complete</u> <u>Status</u> function for information about reading the MCS register.

This function allows a great deal of control over when and how a move is considered complete. The **criteria** bitmap contains five bits to set the conditions used to determine the Move Complete status. The first two, Profile Complete and Motor Off, are logically OR'd to provide the basis for Move Complete. The Profile Complete bit is always set and cannot be disabled. When the axis trajectory generator completes its profile, this condition is satisfied. If the Motor Off bit is set, any condition that causes the axis to turn its motor off (a kill or following error trip) satisfies this basic requirement for Move Complete. In other words, either Profile Complete OR Motor Off must be True for Move Complete to be True.

The next three criteria, Run/Stop, Delay, and In Position, are optional conditions that are logically ANDed to further qualify the Move Complete status. If the Run/Stop bit is set, the axis must also be logically stopped for the move to be considered complete. Refer to the Load Run/Stop Threshold function for information about the Run/Stop status.

If the **delay** bit is set, the axis must wait a user-defined delay after the other criteria are met before the move is considered complete. The user-defined **delay** parameter is typically used to wait the mechanical settling time so that a move is not considered complete until vibrations in the mechanical system have damped out. It also can be used to compensate for PID pull-in time due to the integral term. This pull-in is typically at velocities below the Run/Stop threshold.

Finally, if the In-Position bit is set, the axis checks its final stopping position versus its target position and only sets the Move Complete status if the absolute value of the difference is less than the in-position deadband.

A non-zero value for **minPulse** guarantees the status stays in the True state for at least this minimum time, even if another move starts immediately. You can use this feature to make sure the host does not miss a Move Complete status when it polls the Move Complete Status register.

Note You can use the Delay parameter to guarantee a minimum time for the False state. The status transitions from Complete to Not Complete at the start of a move and stays in the Not Complete state for at least this delay time even in the case of a zero distance move.

Tip You can adjust move complete criteria settings in MAX on the **Trajectory Settings**.

This function is typically called for each axis prior to using the axis for position control. After the criteria are set, they remain in effect until changed. You can execute this function at any time.

When an axis starts, its corresponding bit in the Move Complete Status register is reset to zero. When the move completes, the bit is set to one. You can check the status of an axis or axes at any time by polling the MCS register. Onboard programs can use this status to automatically sequence moves with the <u>Wait on Event</u> function.



Tip Use <u>Write Trajectory Data</u> for more move complete criteria options.

Configure Stepper Output

Usage

i32 status = flex_configure_stepper_output (u8 boardID, u8 axisOrStepperOutput, u16 outputMode, u16 polarity, u16 driveMode);

Purpose

Configures the drive mode, output mode, and polarity of a stepper output.
Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrStepperOutput	u8	axis or stepper output to configure
outputMode	u16	specifies the output mode
polarity	u16	sets the polarity for the stepper output
driveMode	u16	specifies the drive mode

Parameter Discussion

axisOrStepperOutput is the axis or stepper output to configure. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or

NIMC_STEP_OUTPUT1 through NIMC_STEP_OUTPUT30. On motion controllers that support fewer than thirty axes, configuring non-existent axes or stepper outputs returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Stepper</u> <u>Outputs</u> for axis and stepper output resource IDs. When sent to a stepper axis, this function configures the mapped stepper output. Alternatively, you can execute this function directly on the stepper output resource.

outputMode specifies the mode as Step and Direction or Clockwise/Counterclockwise.

Constant	Value	Descriptio
NIMC_CLOCKWISE_COUNTERCLOCKWISE	0	Configures the mode Clockwise/Counter
NIMC_STEP_AND_DIRECTION	1	Configures the mode and Direction

polarity is the stepper output configured as active low or active high. When configured as active low, the output is active when there is a low signal on the output pin. Conversely, active high means that the output is active when there is a high signal on the output pin.

Constant	Value	Description
NIMC_ACTIVE_HIGH	0	Logical True or On
NIMC_ACTIVE_LOW	1	Logical False or Off

driveMode specifies either **Open Collector** or **Totem Pole** mode. **driveMode** is available only on the 7350 and is not configurable on 7330/40/90 motion controllers.

Constant	Value	Description
NIMC_OPEN_COLLECTOR	0	Configures output to be Open Collector
NIMC_TOTEM_POLE	1	Configures output to be Totem Pole

Using This Function

Use this function to configure a stepper output to correctly interface with a stepper driver. NI-Motion supports the two industry standards for stepper control outputs. The most popular mode is **Step and Direction**, where one output produces the step pulses and the other output produces a direction signal.

In **Clockwise/Counterclockwise** (**CW/CCW**) mode, the first output produces pulses when moving forward, or **CW**, while the second output produces pulses when moving reverse, or **CCW**.

In either mode, you can set the active polarity with the polarity bit to active low or active high. For example, in **Step and Direction** mode, the polarity bits determine if a high direction output is forward or reverse. It also determines the resting states of outputs when they are not pulsing.

The Configure Stepper Output function is typically called for each stepper axis prior to using the axis for position control. After the modes and polarity are set, they remain in effect until changed. You can execute this function at any time.

For the 7350 controller, you can set the stepper output **driveMode** to **Open Collector** or **Totem Pole**. On the 7330/40, stepper output is **Open Collector** and cannot be changed. On the 7390, stepper output is isolated so **driveMode** does not apply and cannot be changed. For more information about **Open Collector** and **Totem Pole** drive modes, refer to the documentation for your 7350 device. flex_config_vect_spc

Device Compatibility

Configure Vector Space

Usage

status = flex_config_vect_spc(u8 boardID, u8 vectorSpace, u8 xaxis, u8 yaxis, u8 zaxis);

Purpose

Defines the axes that are associated with a vector space.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
vectorSpace	u8	vector space to configure
xaxis	u8	physical axis to act as the logical x axis
yaxis	u8	physical axis to act as the logical y axis
zaxis	u8	physical axis to act as the logical z axis

Parameter Discussion

vectorSpace is the vector space to configure. On motion controllers that support fewer than thirty axes, configuring non-existent vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Vector</u> <u>Spaces</u> for vector space resource IDs.



Note For NI 7330/40/90 motion controllers, valid values are NIMC_VECTOR_SPACE1 (default), NIMC_VECTOR_SPACE2, and NIMC_VECTOR_SPACE3.



Note For the NI 7350 motion controllers, valid values are NIMC_VECTOR_SPACE1 (default) through NIMC_VECTOR_SPACE4.



Note For the NI SoftMotion Controller, valid values are NIMC_VECTOR_SPACE1 (default) through NIMC_VECTOR_SPACE15.

xaxis is the physical axis (NIMC_AXIS1 through NIMC_AXIS30) to act as the logical x axis. The default is NIMC_NOAXIS (none).

yaxis is the physical axis (NIMC_AXIS1 through NIMC_AXIS30) to act as the logical y axis. The default is NIMC_NOAXIS (none).

zaxis is the physical axis (NIMC_AXIS1 through NIMC_AXIS30) to act as the logical z axis. The default is NIMC_NOAXIS (none).

Using This Function

The Configure Vector Space function is used to group axes into a vector space. A vector space defines an x and y (2D) or x, y, and z (3D) coordinate space. You can map any physical axis to the logical x, y, and z axes to coordinate the motion of multiple axes. The total number of vector spaces NI-Motion supports in your motion control system can be calculated using the following formula:

total vector spaces = [number of activated axes/2].

After it is configured, you can use the Vector Space number in all functions that support vector spaces. Vector spaces are used in 2D and 3D linear interpolation with vector position, vector velocity, vector acceleration and deceleration, and vector operation mode. They are also used in circular, helical and spherical arc moves. You can start, blend, and stop vector spaces just like axes. You can even synchronously start multiple vector spaces for multi-vector space coordination.

Many status and data readback functions also operate on vector spaces. You can read vector position, vector velocity, vector blend status, and so on, or you can read per-axis values and status for the axes within the vector space.

While vector spaces can be comprised of three axes, it is possible to define two-axis or even one-axis vector spaces. These vector spaces function properly for all functions that do not require a greater axis count.

You can use other complex motion control functions with vector spaces, including electronically gearing an independent axis to a master axis contained within a vector space definition.



Note Axes cannot belong to two vector spaces at the same time. To move an axis from one vector space to another, you must demap the axis from the first vector space and then map it to the second vector space.



Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

Example

Vector space 2 (NIMC_VECTOR_SPACE2) is configured with axis 3 as the x axis, axis 1 as the y axis, and axis 2 as the z axis. The resulting 3D

vector space is shown in the following figure:

Z Axis = Axis 2



flex_enable_axis

Device Compatibility

Enable Axis

Usage

status = flex_enable_axis(u8 boardID, u8 reserved, u8 PIDrate, u16 axisMap);

Purpose

Enables the operating axes and defines the PID and trajectory update rate.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
reserved	u8	unused input
PIDrate	u8	PID update rate
axisMap	u16	bitmap of axes to enable

Parameter Discussion

reserved is an unused input. The input value is ignored.

PIDrate is the PID control loop and trajectory generator update rate, for NI 73*xx* motion controllers. For stepper axes, this parameter also determines how often the step generator is updated. The range for this parameter is 0 to 7, with a default of 3 (250 μ s).

Update Rate = (PIDrate + 1) x 62.5 μ s

PIDrate	Update Rate
0	62.5 μs
1	125 µs
2	188 µs
3	250 µs (default)
4	312 µs
5	375 µs
6	438 µs
7	500 µs

axisMap is the bitmap of enabled axes.

D15 .	•••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis . 15		Axis 10	Axis 9	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	0

D1 through D15:

1 = Axis enabled

0 = Axis disabled (default)

Using This Function



Caution Illegally configured axes cannot be enabled and attempting to do so generates an error. For example, an attempt to enable a servo axis that does not at least have its Primary Feedback device mapped generates an error.

You can set the update rate slower than the maximum. This is useful in many applications to scale the effective range of the PID control loop parameters and/or to improve stability. Refer to the Load Single PID Parameter function for more information about the PID parameters affected by the update rate.

You cannot change the PID rate when axes are enabled. To change the PID rate from the default (250 μ s) while not changing the number of axes enabled, you must disable all axes and then re-enable them with the new PID rate.



Caution If an axis is configured for onboard commutation, the axis attempts to initialize the current phase. Depending on the phase initialization type, this may cause the axis to move. If you disable the axis, the current phase information and the phase initialization occur when you enable the axis.



Note To use a slower PID rate, set the NIMC_PID_RATE_MULTIPLIER attribute in the Load Advanced Control Parameter function.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- The ADC channel scan rate is affected by the number of channels enabled. This could limit the effective update rate (for axes with analog feedback). Refer to the <u>Enable ADCs</u> function for more information.
- This function is used to enable the specific axes required for the application and set the servo (and stepper) update or sample rate. An axis is enabled for operation by a True in the corresponding location in the bitmap. Only enabled axes are updated and there is a direct correspondence between the number of enabled axes and the fastest update rate allowed.

Number of Enabled Axes (7330/40/90)	Fastest Update Rate	Number of Enabled Axes (7350)	Fastest Update Rate
1	62 µs	1–2	62 µs
2	125 µs	3–4	125 µs
3	188 µs	5–6	188 µs
4	250 µs	7–8	250 µs

- The fastest update rate is achievable only when all axes are single-feedback servo axes and no extra encoders are enabled.
 - **Caution** Update rates that are too fast for the number of axes, stepper outputs, and/or encoders enabled generates an error and the previous setting remains in effect. Refer to Errors and Error Handling for information about errors and error handling.
- The Enable Axis function automatically enables the feedback devices mapped to the enabled axes. It is not necessary to

explicitly enable the encoders or ADC channels before enabling the axes. Axes must be killed before they are disabled or the error NIMC_wrongModeError is generated. Refer to the <u>Enable Encoders</u> and <u>Enable ADCs</u> functions for more information about enabling and disabling these resources when you are using them not mapped to an axis.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- Use the <u>Load Axis Configuration Parameter</u> function to enable axes sixteen through thirty.
- It is not necessary to kill, or deactivate, an axis before disabling it.
- When you are using this function with the NI SoftMotion Controller, you can enable axes, but you cannot change the PID rates. To configure the axes on these drives, use the configuration utility that is included with the Copley Controls drive.
- The NI SoftMotion Controller host loop update rates depend on the number of axes in the system as follows:
 - 10 ms for a motion control system that uses 1 to 8 axes
 - 20 ms for a motion control system that uses 9 to 15 axes
 - 30 ms for a motion control system that uses 16 to 23 axes
 - 40 ms for a motion control system that uses 24 to 30 axes

flex_load_advanced_control_parameter <u>Compatib</u>

Load Advanced Control Parameter

Usage

i32 status = flex_load_advanced_control_parameter (u8 boardID, u8 axis, u16 parameterType, u32 value, u8 inputVector);

Purpose

Sets an advanced control loop parameter for a given axis.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to configure
parameterType	u16	parameter to configure
value	u32	value to load for the specified parameter
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the axis to configure. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

parameterType is one of the following control loop parameters to set:

Constant	Value	Description
NIMC_STATIC_FRICTION_MODE	0x0	Enables or disables static friction compensation.
NIMC_STATIC_FRICTION_MAX_DEADZONE	0x3	Specifies the maximum deadzone window for static friction compensation.
NIMC_STATIC_FRICTION_MIN_DEADZONE	0x4	Specifies the minimum deadzone window for static friction compensation.
NIMC_STATIC_FRICTION_ITERM_OFFSET_FWD	0x5	Specifies the voltage necessary to overcome static friction when beginning a move in the forward direction.
NIMC_STATIC_FRICTION_ITERM_OFFSET_REV	0x6	Specifies the

		voltage necessary to overcome static friction when beginning a move in the reverse direction.
NIMC_PID_RATE_MULTIPLIER	0x7	Specifies a multiplier to slow the PID Rate .
NIMC_NOTCH_FILTER_FREQUENCY	0x8	Specifies the frequency of the signal that you want to remove.
NIMC_NOTCH_FILTER_BANDWIDTH	0x9	Specifies the bandwidth of the filter.
NIMC_NOTCH_FILTER_ENABLE	0xA	Enables or disables the notch filter.
NIMC_LOWPASS_FILTER_CUTOFF_FREQUENCY	0xB	Specifies a cutoff frequency to remove output filters.
NIMC_LOWPASS_FILTER_ENABLE	0xC	Enables or disables the lowpass filter.
NIMC_SECONDARY_PID_MODE	0xD	Configures the controller to use the second set of PID

parameters based on the condition specified in
value.

value is dependent upon the specified control loop parameterType.

inputVector contains the parameter and value that are the source of the data for this function.

Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return</u> <u>Vectors</u> for more detailed information.

Using This Function

Use this function to configure a single control loop parameter for a given axis.

Enable **Static Friction Mode** to configure static friction (stiction) compensation on the motion system. Stiction compensation adjusts for increased power needed to overcome static friction when beginning a move from a standstill. Set **Static Friction Mode** to 1 to enable, or 0 to disable, static friction compensation.

The **Max** and **Min Deadzones** are the windows around the target position that determine its sensitivity to external forces that try to move it from its position. Inside the maximum deadzone window, an axis can be moved without response. As soon as the movement causes the axis to exceed the maximum deadzone, the control loop responds by correcting the position back to within the window defined by minimum deadzone. The range is 1 to 32,767.

Forward and **Reverse Offset Voltages** are the voltages necessary to overcome static friction when beginning a move. The voltages can be the same for both directions, but can be different under some circumstances. For example, if an axis moves against gravity in one direction and with gravity in the other direction, the offset voltage necessary to overcome static friction against gravity is higher than that of the direction that moves with gravity.

The **PID Rate Multiplier** specifies a multiplier to slow the **PID Rate**. The default is 1 (no change), and the range is 1-10. For example, with a 250µs **PID Rate** and a **PID Rate Multiplier** of 4, the effective control loop rate is shown in the following equation:

```
effective rate = PID Rate × PID Rate Multiplier = 250\mu s \times 4 = 1 ms
```

Notch filters remove a narrow band of frequencies around a given point to filter output signals that could cause system instability. Set **Notch Filter Enable** to 1 to enable, or 0 to disable, the filter. Use **Notch Filter Frequency** to specify the frequency of the signal that you want to remove. The range for the frequency is 1 Hz to **PID Rate** (in Hz)/2. Use **Notch Filter Bandwidth** to specify the bandwidth of the filter.

For example, a 250µs **PID Rate** would have a maximum frequency of 2000 Hz as shown in the following equation:

```
max frequency = 4000 Hz/2 = 2000 Hz or
```

max frequency = $(250 \ \mu s/2)^{-1} = 2000 \ Hz$

Lowpass filters remove output signals above a specific cutoff frequency. The range for the frequency is 1 Hz to **PID Rate** (in Hz)/2. Set **Lowpass Filter Enable** to 1 to enable, or 0 to disable, the lowpass filter.

Second Set of PID Parameters is used to configure the controller to transition from the primary set of PID parameters to the secondary set based on the current condition specified in **value**. This transition occurs in the control loop, so there is no latency between transition time. The following table includes valid conditions this transition can occur under.

Name	Value	Constant	Description
Disabled	0	NIMC_PID_DISABLED	Always uses the primary set of PID parameters.
Switch Feedback	1	NIMC_PID_CHANGE_IN_FEEDBACK	Uses the secondary set of PID parameters when feedback changes.
Acceleration	2	NIMC_PID_ACCELERATING	Uses the second set of PID parameters when the move is accelerating toward the target velocity.
Velocity	3	NIMC_PID_MOVING	Uses the second set of PID

			parameters when the motor is moving. Otherwise uses the primary set of PID parameters.
Direction	4	NIMC_PID_MOVING_REVERSE	Uses the second set of PID parameters if the motor is moving in the reverse direction. Otherwise, uses the primary set of PID parameters.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx controller:

- Static Friction and PID Rate Multiplier parameters can be set only on 7340/50 motion controllers.
- Notch Filter and Lowpass Filter parameters can be set only on 7350 motion controllers.
- The NIMC_SECONDARY_PID_MODE attribute is valid only on 7350 controllers.
 - Only the **Disabled** and **Switch Feedback** options are valid.
 - If the **Switch Feedback** option is selected, the control loop toggles between sets of PID parameters each time the feedback is changed. For example, if you are using encoder 1, the control loop uses the primary set of PID parameters. If you switch to ADC 1, the control loop uses the secondary set of PID parameters. If you switch back to encoder 1, the control loop switches back to the primary set of PID parameters.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- If the **Switch Feedback** option is selected, the control loop uses the primary set of PID parameters when you use a primary resource and uses the secondary set of PID parameters when you use an auxiliary resource. For example, if you use encoder 1, the control loop uses the primary set of PID parameters. If you use auxiliary encoder 1, the control loop uses the secondary set of PID parameters.
- NIMC_SECONDARY_PID_MODE is not valid for the NI SoftMotion Controller for CANopen.

flex_load_commutation_parameter

Device Compatibility

Load Commutation Parameter

Usage

i32 status = flex_load_commutation_parameter (u8 boardID, u8 axis, u16 attribute, f64 value);

Purpose

Sets a commutation parameter for a given axis.

Parameters

Name	Туре	Description		
boardID	u8	assigned by Measurement & Automation Explorer (MAX)		
axis	u8	axis to configure		
attribute	u16	attribute to configure		
value	f64	value to load for the specified attribute		

Parameter Discussion

axis is the axis to configure. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

attribute is one of the following commutation attributes to set:

Constant	Value	Description
NIMC_COMM_INITIALIZATION_TYPE	0	Loads the Initialization Type for sinusoidal commutation.
NIMC_COMM_FIND_ZERO_VOLT	1	Loads Find Zero Voltage for sinusoidal commutation.
NIMC_COMM_FIND_ZERO_TIME	2	Loads Find Zero Time for sinusoidal commutation.
NIMC_COMM_DIRECT_SET_PHASE	3	Loads Direct Set Phase for sinusoidal commutation.
NIMC_COMM_ELECTRICAL_CYCLE_COUNTS	4	Loads the Electrical Cycle Counts for sinusoidal commutation.
NIMC_HALL_SENSOR_TYPE	5	Loads the Hall Effect Sensor type for sinusoidal commutation.

NIMC_COMM_MODE	6	Loads the Commutation Mode for sinusoidal
		commutation.

value is dependent upon the specified commutation **attribute**. Select the appropriate **value** for the corresponding **attribute**:

Attribute	Value
NIMC_COMM_INITIALIZATION_TYPE (0)	0 NIMC_HALL_SENS 1 NIMC_SHAKE_AN 2 NIMC_DIRECT_SE
NIMC_COMM_FIND_ZERO_VOLT (1)	0 to 10 volts
NIMC_COMM_FIND_ZERO_TIME (2)	0 to 10,000 ms
NIMC_COMM_DIRECT_SET_PHASE (3)	0 to 359°
NIMC_COMM_ELECTRICAL_CYCLE_COUNTS (4)	2 to 268,534,000 cou
NIMC_HALL_SENSOR_TYPE (5)	0 NIMC_HALL_SENS 1 NIMC_HALL_SENS
NIMC_COMM_MODE (6)	0 NIMC_EXTERNAL 1 NIMC_ONBOARD_
Using This Function

Use the Load Commutation Parameter function to configure a commutation parameter for a given axis.



Note Only 7350 controllers support the Load Commutation Parameter function.

Initialization Type specifies how the controller initializes the commutation phase. Select one of the following values:

Value	Description
0	Hall Sensor—Initializes the phase based on the Hall sensor's input.
1	Shake and Wake —Initializes the phase by driving the system to the zero degree commutation phase.
2	Direct Set —Sets the commutation phase directly with the phase angle specified.

Caution Phase initialization is performed when the axes are enabled. If you use **Shake and Wake**, the system can move during the initialization process.

Find Zero Voltage specifies the voltage to use with **Shake and Wake** initialization. The range is 0 to 10 volts. **Find Zero Time** specifies the duration of **Shake and Wake** initialization in milliseconds. The range is 0 to 10,000 ms.

Direct Set Phase specifies phase angle to set with **Direct Set** initialization. For **Direct Set**, the controller sets the current position as the specified phase angle. National Instruments recommends this initialization method only for custom systems with a known initial phase angle. The range for the phase angle is 0 to 359°.

Electrical Cycle Counts specifies the encoder counts for the commutation cycle. This information is usually listed in the motor or stage documentation. Some motor manufacturers present this information as the number of poles. In such cases, convert number of poles to counts per electrical cycle with the following equation. The valid data range is 2 to 268,435,000 counts.

encoder counts per electrical cycle = $2 \times \frac{\text{encoder counts per revolution}}{\text{number of poles}}$

Caution Counts per electrical cycle must be set correctly to avoid overheating and damaging the motor.

Hall Sensor Type specifies the type of Hall Effect sensor to use. There are two base types of Hall Effect sensor inputs. If you are using **Type 1**, the pattern matches the following graph:



If you are using Type 2, the pattern matches the following graph:



Refer to your motor documentation for information about the type of Hall Effect sensor to use. Refer to the documentation for your 7350 device for information about configuring Hall Effect sensors. Select one of the following values:

Value	Description
0	Туре 1
1	Туре 2

Commutation Mode configures the axis to use onboard commutation or to be commutated externally. Select one of the following values:

Value	Description	
0	External Commutation	
1	Onboard Commutation	

Call the Load Commutation Parameter function before enabling the axis. If the axis is enabled, the commutation parameters does not take effect until you call <u>Enable Axis</u> again.

Device Compatibility

Load Counts/Steps per Revolution

Usage

status = flex_load_counts_steps_rev(u8 boardID, u8 axis, u16 unitType, u32 countsOrSteps);

Purpose

Loads the quadrature counts or steps per revolution for an axis.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to control
unitType	u16	type of information, counts or steps, to load
countsOrSteps	u32	quadrature counts or steps per revolution

Parameter Discussion

axis is the axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

unitType is the type of information, counts or steps, to load. The following are valid values:

unitType Constant	unitType Value	Action
NIMC_COUNTS	0	Use to define counts/revolution for a quadrature encoder.
NIMC_STEPS	1	Use to define steps/revolution for a stepper motor.

countsOrSteps is either the quadrature counts per revolution for the encoder mapped to the axis or the number of steps (full, half, or microstep) per motor revolution.

Using This Function

Use this function to load any feedback value (counts or volts) per unit of measure or to load the number of steps per motor revolution. For encoders, this is typically in units of quadrature counts per motor revolution, but can be counts per inch, per cm, or per any unit of measure. For analog feedback, it can be in units of scaled voltage. Steps can be full steps, half steps, or microsteps depending upon how you have the external stepper driver and motor configured.

This parameter must be correctly loaded before you call the Load Velocity in RPM, Load Accel/Decel in RPS/sec, Load Velocity Threshold in RPM, Read Velocity in RPM, Load Move Constraints, or Find **Reference** functions.

The Find Reference function searches for the encoder index for one revolution as defined by this function. Therefore, another useful unit of measure is counts per index period. Linear encoders often have indexes every inch or every centimeter.

Closed-loop stepper functionality relies on the ratio of counts to steps and not on the absolute values of counts or steps per revolution. For closedloop operation, any unit of measure (UOM) that allows you to enter both counts per UOM and steps per UOM that are within their valid ranges work. Notice that this function must be called twice: one time to set up the stepper motor steps/rev, and again to set up the guadrature encoder counts/rev.



Caution For closed-loop stepper controllers, steps per revolution/counts per revolution must be in the range of 1/32,767 <steps/counts < 32,767.

This function loads a scale factor that affects subsequently loaded and read back values of velocity and acceleration.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx controller:

- **countsOrSteps** is interpreted as number of encoder counts per revolution for encoder feedback or number of ADC codes (LSBs) per revolution for analog feedback.
- The range for the **countsOrSteps** parameter is 2 to 2²⁸–1 with a default value of 2,000.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

• The range for the **countsOrSteps** parameter is 1 to 2³¹–1 with a default value of 1.



- **Note** This parameter must be correctly loaded before you load target positions or write contour position information to the controller or camming table.
- This function determines the unit used when loading velocity. You must set the countsOrSteps value to 1 to use this function to load velocity values in counts/sec and load acceleration values in counts/sec². If you set countsOrSteps to any value other than 1, you must use the Load Move Constraints function to load velocity and acceleration in counts.

flex_load_pid_parameters

Device Compatibility

Load All PID Parameters

Usage

status = flex_load_pid_parameters(u8 boardID, u8 axis, PID* PIDValues, u8
inputVector);

Purpose

Loads all 8 PID control loop parameters for an axis.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to configure
PIDValues	PID*	data structure containing all 8 PID parameters
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the axis for which to load PID parameters. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

PIDValues data structure contains all eight PID parameters in the following structure:

struct {

u16 kp;// Proportional Gain

u16 ki;// Integral Gain

u16 il;// Integration Limit

u16 kd;// Derivative Gain

u16 td;// Derivative Sample Period

u16 kv;// Velocity Feedback Gain

u16 aff;// Acceleration Feedforward Gain

u16 vff;// Velocity Feedforward Gain

} PID;

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function allows you to set all eight PID parameters at the same time for a given axis. You can call this function at any time. However, it is typically used during initialization to configure and tune an axis. NI-Motion also offers a <u>Load Single PID Parameter</u> function, which you can use to change an individual value on-the-fly without having to worry about the other unchanged values.

Refer to the motion controller documentation for an overview of the PID control loop on the NI motion controller and to the <u>Load Single PID</u> <u>Parameter</u> function for descriptions of the individual PID parameters and their ranges.



Note If you are doing onboard programming and are using **inputVector** to get the data this function needs, note that this function reads the variables starting at the memory address pointed to by inputVector in the following order: Kp, Ki, IL, Kd, Td, Kv, Aff, Vff.

Loading a Second Set of PID Parameters

To load a second set of PID parameters, use the Load All PID Parameters and Load Single PID Parameter functions to load the second set of PID values. Instead of using regular axis constants, such as axis 1 = NIMC_AXIS1, axis 2 = NIMC_AXIS2, and so on, use the following alternate constants: axis 1 = NIMC_SECOND_PID1, axis 2 = NIMC_SECOND_PID2, up to NIMC_SECOND_PID30.

Use the <u>Load Advanced Control Parameter</u> function to enable the new set of PID parameters based on the condition specified in the **value** parameter.

Refer to the *Remarks* section of the <u>Load Advanced Control Parameter</u> topic for more information about how to change feedback while the axis is enabled.

flex_load_single_pid_parameter

Device Compatibility

Load Single PID Parameter

Usage

status = flex_load_single_pid_parameter(u8 boardID, u8 axis, u16
parameterType, u16 PIDValue, u8 inputVector);

Purpose

Loads a single PID control loop parameter for an axis.

Parameters

Name	Туре	Description	
boardID	u8	assigned by Measurement & Automation Explorer (MAX)	
axis	u8	axis to configure	
parameterType	u16	selects PID parameter to load	
PIDValue	u16	PID value to load	
inputVector	u8	source of the data for this function	

Parameter Discussion

axis is the axis for which to load PID parameters. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

parameterType Constant	parameterType Value
NIMC_KP	0
NIMC_KI	1
NIMC_IL	2
NIMC_KD	3
NIMC_TD	4
NIMC_KV	5
NIMC_AFF	6
NIMC_VFF	7

parameterType is the selector for the PID parameter to load.

PIDValue is the value to load for the selected PID parameter.

PID Parameter	Abbreviation	Data Ranges	Default
Proportional Gain	Кр	0 to 32,767	100
Integral Gain	Ki	0 to 32,767	0
Integration Limit	Ilim	0 to 32,767	1,000
Derivative Gain	Kd	0 to 32,767	1,000
Derivative Sample Period	Td	0 to 63	2
Velocity Feedback Gain	Kv	0 to 32,767	0
Acceleration Feedforward Gain	Aff	0 to 32,767	0
Velocity Feedforward Gain	Vff	0 to 32,767	0

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function allows you to change an individual PID value on-the-fly without having to worry about the other unchanged PID values. Refer to the NI motion controller documentation for an overview of the enhanced PID control loop on the NI motion controller.

Example

To load a Kp of 1,000 to an axis 2, call the Load Single PID Parameter function with the following parameters:

- **axis** = NIMC_AXIS2
- parameterType = NIMC_KP
- **PIDValue** = 1,000
- **inputVector** = 0xFF (Immediate)

Loading a Second Set of PID Parameters

To load a second set of PID parameters, use the Load All PID <u>Parameters</u> and Load Single PID Parameter functions to load the second set of PID values. Instead of using regular axis constants, such as axis 1 = NIMC_AXIS1, axis 2 = NIMC_AXIS2, and so on, use the following alternate constants: axis 1 = NIMC_SECOND_PID1, axis 2 = NIMC_SECOND_PID2, up to NIMC_SECOND_PID30.

Use the <u>Load Advanced Control Parameter</u> function to enable the new set of PID parameters based on the condition specified in the **value** parameter.

Refer to the *Remarks* section of the <u>Load Advanced Control Parameter</u> topic for more information about how to change feedback while the axis is enabled.

PID Loop Parameters

Refer to the links below for specific information about the components of the PID loop:

- Proportional Gain
- Integral Gain
- Integration Limit
- Derivative Gain

- Derivative Sample Period
- Velocity Feedback Gain
- Acceleration Feedforward
- <u>Velocity Feedforward</u>

Set Stepper Loop Mode

Usage

status = flex_set_stepper_loop_mode(u8 boardID, u8 axis, u16 loopMode);

Purpose

Sets a stepper axis to operate in either open-loop, closed-loop, or P-Command mode.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer
axis	u8	axis to control
loopMode	u16	mode of the stepper axis

Parameter Discussion

axis is the axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

loopMode sets the mode for the stepper axis.

- 0 = open-loop (default)
- 1 = closed-loop
- 2 = P-Command

Using This Function

You can operate stepper axes in both open and closed-loop modes. In open-loop mode, the stepper axis controls the trajectory profile and generates steps but has no feedback from the motor or actuator to determine if the profile is followed correctly.

In closed-loop mode, the feedback position is constantly compared to the number of steps generated to see if the stepper motor is moving correctly. When the trajectory profile is complete, missing steps (if any) are made up with a pull-in move. If, at any time during the move, the difference between the instantaneous commanded position and the feedback position exceeds the programmed following error limit, the axis is killed and motion stops.

You use P-Command mode for systems in which a servo drive receives step/dir (CW/CCW) from the motion controller. The drive closes the PID loop, and the motion controller generates the trajectory. Similar to closed-loop mode, in P-Command mode, the axis is killed when the programmed following error limit is exceeded.

Caution For proper closed-loop and P-Command operation, the correct values for steps/rev and counts/rev must be loaded with the Load Counts/Steps per Revolution function. Incorrect counts to steps ratio can result in failure to reach the desired target position and erroneous closed-loop stepper operation.

To operate in closed-loop or P-Command mode, a stepper axis must have a primary feedback resource, such as an encoder, mapped to it prior to enabling the axis. Refer to the <u>Configure Axis Resources</u> function for more information about feedback resources. You can operate an axis with a primary feedback resource in open-loop, closed-loop, or P-Command mode and you can switch the mode at any time. You can still read the position of the mapped feedback resource even when the axis is in open-loop mode. flex_load_velocity_filter_parameter

Device Compatibility

Load Velocity Filter Parameter

Usage

status = flex_load_velocity_filter_parameter(u8 boardID, u8 axis, u16 filterDistance, u16 filterTime, u8 inputVector);

Purpose

Loads the velocity filter parameters and sets the velocity threshold, above which an axis is considered running.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to control
filterDistance	u16	number of position changes necessary for a velocity to be reported
filterTime	u16	filter time constant in ms
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

filterDistance is the number of position changes in counts/steps necessary for a velocity to be reported. The range for this parameter is from 0 to 1000 counts/steps. A value of 0 turns off the velocity filter.

filterTime is the maximum velocity update period in ms. The range for this parameter is 1 to 2500 ms.

inputVector contains the distance and interval parameters that are the source of the data for this function.

Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return</u> <u>Vectors</u> for more detailed information.

Using This Function

This function configures the internal filter for velocity reporting. The reported velocity can oscillate due to feedback quantization error, noise, and jittery systems, rendering the data useless. You can apply a filter over multiple data samples to calculate an average velocity that reflects the actual system behavior. This filter is represented by **filterDistance**. The larger the **filterDistance**, the more position changes accumulate to perform velocity calculation. This is usually better for a noisy system.

A large **filterDistance** can introduce a longer delay in velocity reporting as the position changes accumulate. To limit this delay and sluggish data reading, specify a minimum update interval using the **filterTime** parameter. The axis velocity is updated within this update interval if the **filterDistance** requirement is not met.

- Note Velocity quantization noise is a measurement-only phenomenon and does not affect the ability of the motion controller to accurately control velocity and position during a move.
- Note This function expects the parameters to be loaded in the following order if onboard variables are used: filterDistance, filterTime.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

• For the **inputVector** parameter, the NI SoftMotion Controller supports only the immediate return vector (0XFF).

Trajectory Control

This functions group contains detailed descriptions of functions used to set up and control motion trajectories on the NI motion controller. It includes functions to load double-buffered trajectory parameters, read back instantaneous velocity, position and trajectory status, as well as functions to configure blending, gearing, and other advanced trajectory features.

Double buffered parameters for axes and vector spaces include acceleration, deceleration, velocity, s-curve, operation mode, target position, and circular, helical, and spherical arc parameters. You can send these parameters to the controller at any time, but they do not take effect until you execute the next <u>Start Motion</u> or <u>Blend Motion</u> function. This double buffering allows you to set up moves ahead of time, synchronizing them with a single Start Motion or Blend Motion call.

Other trajectory functions allow you to configure the operation of trajectory generators and set status thresholds. These parameters include following error, blend factor, gear master, ratio and enable, position modulus, velocity threshold, torque limit, torque offset, and software limit positions. Unlike double-buffered parameters, if you change these parameters on the fly, they take effect immediately. Also in this category are functions to reset position to zero or another value and to force a velocity override.

During a move, you can read the instantaneous values of position, velocity, following error, and DAC output (torque). There are also functions to read the following trajectory status: move complete, profile complete, blend complete, motor off, following error trip, velocity threshold, and DAC limit status. These trajectory values and status are used for move sequencing, system coordination, and overall monitoring purposes.

Finally, NI-Motion offers a set of functions to acquire time-sampled position and velocity data into a large onboard buffer and then later read it out for analysis and display. These functions implement a digital oscilloscope that is useful during system set up, PID tuning, and general motion with data acquisition synchronization.

Load Target Position

Load Vector Space Position
Set Operation Mode Load Following Error **Reset Position** Load Velocity in RPM Load Velocity Load Accel/Decel in RPS/sec Load Acceleration/Deceleration **Read Position Read Vector Space Position** Read Velocity in RPM Read Velocity **Read Following Error** Read per Axis Status **Read Blend Status** Read Move Complete Status **Read Trajectory Status Check Blend Complete Status Check Move Complete Status** Wait for Blend Complete Wait for Move Complete

flex_check_blend_complete_status

Device Compatibility

Check Blend Complete Status

Usage

status = flex_check_blend_complete_status(u8 boardID, u8 axisOrVectorSpace, u16 axisOrVSMap, u16* blendComplete);

Purpose

Checks the blend complete status for an axis, vector space, group of axes, or group of vector spaces.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space selector
axisOrVSMap	u16	bitmap of axes or vector spaces to check
blendComplete	u16*	the blend complete status

Parameter Discussion

axisOrVectorSpace can select an axis (NIMC_AXIS1 through NIMC_AXIS15), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously checking multiple axes or vector spaces, the **axisOrVSMap** parameter indicates which axes or vector spaces to check. On motion controllers that support fewer than fifteen axes, checking non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

axisOrVSMap is the bitmap of axes or vector spaces to check. It is only required when multiple axes or vector spaces are selected with the **axisOrVectorSpace** parameter. Otherwise, this parameter is ignored. NI-Motion ignores additional axes and vector spaces if you select non-existent axes.

When checking multiple axes (**axisOrVectorSpace** = NIMC_AXIS_CTRL):

D15	••••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis		Axis	0									
15		10	9	8	1	6	5	4	3	2	L	

For D1 through D15:

1 = Blend must be complete on specified axis

0 = Blend can be either complete or not complete on specified axis (do not care)

When checking multiple vector spaces (**axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL):

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS 15	 VS 10	VS 9	VS 8	VS 7	VS 6	VS 5	VS 4	VS 3	VS 2	VS 1	0

For D1 through D15:

1 = Blend must be complete on specified vector space

0 = Blend can be either complete or not complete on specified vector space (do not care)

To check for blend complete on a single axis or vector space, set the

axisOrVectorSpace selector to the specified axis or vector space. The **axisOrVSMap** bitmap is ignored. To check for blend complete on multiple axes, the **axisOrVectorSpace** selector is set to NIMC_AXIS_CTRL and the **axisOrVSMap** bitmap defines the axes to check. They must all be blend complete for the **blendComplete** output to be true. Similarly, to check for blend complete on multiple vector spaces, the **axisOrVectorSpace** selector is set to NIMC_VECTOR_SPACE_CTRL and the **axisOrVSMap** bitmap defines the vector spaces to check.

blendComplete indicates if the blend is complete on the axes or vector spaces specified.

- 1 = Blend complete
- 0 = Blend not complete

Using This Function

This function extends the functionality of <u>Read Blend Status</u> such that you can define a combination of axes or coordinate (vector) spaces to check as a group. This function also can check the blend complete status for a single axis or coordinate space, much like Read Blend Status. Instead of decoding the output of the Read Blend Status function yourself, this function does that for you by comparing the axes or vector spaces specified in the **axisOrVectorSpace** and **axisOrVSMap** input parameters with the blend complete status for the appropriate axes or vector spaces. The output is a single true/false value indicating if the specified blend or blends are complete.

This function does not check for following error status or axis off status. Checking for faults during moves is recommended. To do this, use <u>Read</u> <u>per Axis Status</u>, which returns following error and axis off status in addition to the move complete status. Refer to the <u>NI-Motion Help</u> for more information and example code.

For more information about blend complete status, refer to the <u>Read</u> <u>Blend Status</u> function.

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following includes considerations you must make when you are using this function with the NI SoftMotion Controller:

To check the blend complete status on axes sixteen through thirty, use the <u>Read per Axis Status</u> function.

flex_check_move_complete_status

Device Compatibility

Check Move Complete Status

Usage

status = flex_check_move_complete_status(u8 boardID, u8 axisOrVectorSpace, u16 axisOrVSMap, u16* moveComplete);

Purpose

Checks the move complete status for an axis, vector space, group of axes, or group of vector spaces.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space selector
axisOrVSMap	u16	bitmap of axes or vector spaces to check
moveComplete	u16*	the move complete status

Parameter Discussion

axisOrVectorSpace can select an axis (NIMC_AXIS1 through NIMC_AXIS15), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously checking multiple axes or vector spaces, the **axisOrVSMap** parameter indicates which axes or vector spaces to check. On motion controllers that support fewer than fifteen axes, checking non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

axisOrVSMap is the bitmap of axes or vector spaces to check. It is only required when multiple axes or vector spaces are selected with the **axisOrVectorSpace** parameter. Otherwise, this parameter is ignored. NI-Motion ignores additional axes and vector spaces if you select non-existent axes.

When checking multiple axes (**axisOrVectorSpace** = NIMC_AXIS_CTRL):

D15	••••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis 15		Axis 10	Axis 9	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	0

For D1 through D15:

1 = Move must be complete on specified axis

0 = Move can be either complete or not complete on specified axis (do not care)

When checking multiple vector spaces (**axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL):

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS 15	 VS 10	VS 9	VS 8	VS 7	VS 6	VS 5	VS 4	VS 3	VS 2	VS 1	0

For D1 through D15:

1 = Move must be complete on specified vector space

0 = Move can be either complete or not complete on specified vector space (do not care)

To check for move complete on a single axis or vector space, set the

axisOrVectorSpace selector to the specified axis or vector space. The **axisOrVSMap** bitmap is ignored.

To check for move complete on multiple axes, the **axisOrVectorSpace** selector is set to NIMC_AXIS_CTRL and the **axisOrVSMap** bitmap defines the axes to check. They must all be move complete for the **moveComplete** output to be true. Similarly, to check for move complete on multiple vector spaces, the **axisOrVectorSpace** selector is set to NIMC_VECTOR_SPACE_CTRL and the **axisOrVSMap** bitmap defines the vector spaces to check.

moveComplete indicates if the move is complete on the axes or vector spaces specified.

- 1 = Move complete
- 0 = Move not complete

Using This Function

This function extends the functionality of <u>Read Trajectory Status</u> such that you can define a combination of axes or coordinate (vector) spaces to check as a group. This function also can check the move complete status for a single axis or coordinate space, much like Read Move Complete Status. Instead of decoding the output of the Read Trajectory Status function yourself, this function does that for you by comparing the axes or vector spaces specified in the **axisOrVectorSpace** and **axisOrVSMap** input parameters with the move complete status for the appropriate axes or vector spaces. The output is a single true/false value indicating if the specified move or moves are complete.

This function does not check for following error status or axis off status. Checking for faults during moves is recommended. To do this, use <u>Read</u> <u>per Axis Status</u>, which returns following error and axis off status in addition to the move complete status. Refer to the <u>NI-Motion Help</u> for more information and example code.

Refer to the <u>Read Trajectory Status</u> and <u>Configure Move Complete</u> <u>Criteria</u> functions for more information about move complete status.

 $\overline{\mathbb{N}}$

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following includes considerations you must make when you are using this function with the NI SoftMotion Controller:

To check the move complete status on axes sixteen through thirty, use the <u>Read per Axis Status</u> function.

flex_load_acceleration

Device Compatibility

Load Acceleration/Deceleration

Usage

status = flex_load_acceleration(u8 boardID, u8 axisOrVectorSpace, u16 accelerationType, u32 acceleration, u8 inputVector);

Purpose

Loads the maximum acceleration and/or deceleration value for an axis or vector space.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to control
accelerationType	u16	selector for acceleration, deceleration, or both
acceleration	u32	acceleration value in counts/s ² or steps/s ²
inputVector	u8	source of the data for this function

Parameter Discussion

axisOrVectorSpace is the axis or vector space to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

accelerationType is the selector for loading acceleration, deceleration or both acceleration and deceleration (default).

accelerationType Constant	accelerationType Value
NIMC_BOTH	0 (default)
NIMC_ACCELERATION	1
NIMC_DECELERATION	2

acceleration is the acceleration (and/or deceleration) value in counts/s² (servo axes) or steps/s² (stepper axes).

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function specifies the maximum rate of acceleration and/or deceleration for individual axes or vector spaces. When executed on a vector space, the value controls the vector acceleration (deceleration) along the vector move path.

You can use this function to load separate limits for acceleration and deceleration or to set them both to the same value with one call. These parameters are double-buffered so you can load them on the fly without affecting the move in process, and they take effect on the next <u>Start</u> Motion or Blend Motion call. After they are loaded, these parameters remain in effect for all subsequent motion profiles until re-loaded by this function. You do not need to load acceleration before each move unless you want to change the acceleration and/or deceleration value.

Acceleration defines how quickly the axis or axes come up to speed and is typically limited to avoid excessive stress on the motor, mechanical system, and/or load. A separate, slower deceleration is useful in applications where coming to a gentle stop is important.



Note Use <u>Load Move Constraints</u> to load the acceleration in less than 1 count/s².

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

 Acceleration and deceleration values are converted to an internal 16.16 fixed-point format in units of counts/s² before being used by the trajectory generator. You can calculate the minimum acceleration increment with the following formula:

minimum acceleration/deceleration = Amin × $(\frac{1}{T_5})^2$

where: Amin is 1/65,536 counts/sample² or steps/sample² and Ts is the sample period in seconds per sample.

• For a typical servo axis with 2,000 counts per revolution operating at the 250 ms update rate, calculate the minimum acceleration/deceleration increment using the following equation:

```
\left(\frac{1}{65,536}\right) \times \left(\frac{4,000^2}{2,000}\right) = 0.122070 counts/s
```

• You can calculate the maximum acceleration/deceleration using the following equation:

maximum acceleration/deceleration = Amax $\times (\frac{1}{T_{5}})^{2}$

where: *Amax* is 32 counts/sample and *T*s is the sample period in seconds per sample, and is constrained according to the following equations:

```
acceleration \leq 256 × deceleration deceleration \leq 65536 × acceleration
```

 If you are doing onboard programming and are using inputVector to get the data this function needs, notice that this function reads the variables starting at the memory address pointed to by inputVector in the following order: accelerationType, acceleration.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are

using this function with the NI SoftMotion Controller:

- Use <u>Load Counts/Steps per Revolution</u> to set **countsOrSteps** value to 1. Otherwise, NI-Motion returns an error.
- The range of acceleration/deceleration values is not limited by the fixed 16.16 calculation, and the NI SoftMotion Controller maintains full 64-bit precision.
- For the **inputVector** parameter, the NI SoftMotion Controller supports only the immediate return vector (0XFF).

flex_load_target_pos

Device Compatibility

Load Target Position

Usage

status = flex_load_target_pos(u8 boardID, u8 axis, i32 targetPosition, u8
inputVector);

Purpose

Loads the target position for the next axis move.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to control
targetPosition	i32	target position in counts or steps
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the axis to control. Valid values are NIMC AXIS1 through NIMC AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC badResourceIDOrAxisError). Refer to Axes for axis resource IDs.

targetPosition is the specified target position for the next axis move in counts (servo axes) or steps (stepper axes). Target positions can be anywhere within the 32-bit position range, $-(2^{31})$ to $+(2^{31}-1)$. The default value for target position is zero (0).



Caution Any single move is limited to $\pm (2^{31}-1)$ counts or steps. An error is generated if you exceed this limit by loading a target position too far from the current axis position.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to Input and Return Vectors for more detailed information.

Using This Function

This function loads a target position to the axis specified. Target positions can be for single axis moves, multi-axis coordinated moves, or vector space moves. Position values indicate the specified end location and direction of motion (target position).

Target position is double-buffered so you can load it on the fly without affecting the move in process, and it take effect on the next <u>Start</u> or <u>Blend</u> <u>Motion</u> function. When the target position is loaded, it is interpreted as either an absolute target position, a relative target position, a target position relative to the last captured position or with the effect of a position modulus, depending on the mode set with the <u>Set Operation</u> <u>Mode</u> function.

This means that if you set the operation mode to NIMC_RELATIVE_TO_CAPTURE, you must load the target position after the capture has occurred because the position is evaluated on load.

After you execute the start or blend, the axis or axes follows the programmed trajectory and end up at the absolute, relative, or modulo target position.



Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

flex_load_velocity

Device Compatibility

Load Velocity

Usage

status = flex_load_velocity(u8 boardID, u8 axisOrVectorSpace, i32 velocity, u8
inputVector);

Purpose

Loads the maximum velocity for an axis or vector space.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to control
velocity	i32	velocity in counts/s or steps/s
inputVector	u8	source of the data for this function

Parameter Discussion

axisOrVectorSpace is the axis or vector space to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

velocity is the target or maximum slewing velocity in counts/s (servo axes) or steps/s (stepper axes).



Note It is possible to load a velocity slower than 1 count or step per second by using the <u>Load Velocity in RPM</u> or <u>Load Move</u> <u>Constraints</u> functions.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.
Using This Function

This function specifies the maximum trajectory velocity for individual axes or vector spaces. When executed on a vector space, the value controls the vector velocity along the vector move path. For velocity control applications, the sign of the loaded velocity specifies the move direction.

Velocity is a double-buffered parameter so you can load it on the fly without affecting the move in process, and it take effect on the next <u>Start</u> <u>Motion</u> or <u>Blend Motion</u> call. After it is loaded, this parameters remains in effect for all subsequent motion profiles until re-loaded by this function. You do not need to load velocity before each move unless you want to change the velocity.



Note Use <u>Load Move Constraints</u> to load a velocity of less than 1 count/sec.

 $\overline{\mathbb{N}}$

Note The velocity loaded with this function is the maximum move velocity. Actual velocity attainable is determined by many factors including PID tuning, length of move, acceleration and deceleration values, and physical constraints of the amplifier/motor/mechanical system.

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

 Velocity values in counts/s or steps/s are also converted to the internal 16.16 fixed-point format in units of counts or steps per sample (update) period before being used by the trajectory generator. Although the motion controller can control velocity to 1/65,536 of a count or step per sample, it is impossible to load a value that small with the Load Velocity function, as shown in the following formula:

Velocity in counts or step/s = Vmin × $(\frac{1}{T_s})$

where:

Vmin is 1/65,536 counts/sample or steps/sample, and *Ts* is the sample period in seconds per sample.

 $Ts = 625 \times 10^{-6}$

 $\left(\frac{1}{65,536}\right) \times 16,000 = 0.244$ counts or steps/s

This function takes an integer input with a minimum value of 1 count/s or step/s. You cannot load fractional values. If you need to load a velocity slower than one count or step per second, use the Load Velocity in RPM function.

You can calculate the maximum velocity with the following equation:

maximum velocity = Vmax

where:

Vmax is 20 MHz for servos,

8 MHz for steppers on a 7350 controller,

4 MHz for steppers on a 7330/40/90 controller,

and is constrained by acceleration/deceleration according to the following equation:

velocity \leq (65536 × deceleration) – acceleration

where *velocity* is in counts/sample and *acceleration* and *deceleration* are in counts/sample².

- NI 7350—For servo axes, the velocity range is from ±1 to ±20,000,000 counts/s. For stepper axes it is ±1 to ±8,000,000 steps/s. The upper range limits are the physical limitations of the encoder inputs and stepper generator outputs.
- NI 7340—For servo axes, the velocity range is from ±1 to ±20,000,000 counts/s. For stepper axes it is ±1 to ±4,000,000 steps/s. The upper range limits are the physical limitations of the encoder inputs and stepper generator outputs.
- **NI 7330/7390**—The **velocity** range is ±1 to ±4,000,000 steps/s. The upper range limits are the physical limitations of the encoder inputs and stepper generator outputs.
- NI-Motion firmware uses a 16.16 fixed-point format internally to represent velocity in counts/sample period or steps/sample period and acceleration/deceleration in counts/sample period² or steps/sample period². If you change counts or steps per revolution or change the update rate using <u>Enable Axis</u>, you must reload the velocity and acceleration/deceleration.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- The range of velocity values is not limited by the fixed 16.16 calculation, and full 64-bit precision is maintained.
- Use <u>Load Counts/Steps per Revolution</u> to set the **countsOrSteps** value to 1. Otherwise, NI-Motion returns an error.

flex_load_vs_pos

Device Compatibility

Load Vector Space Position

Usage

status = flex_load_vs_pos(u8 boardID, u8 vectorSpace, i32 xPosition, i32 yPosition, i32 zPosition, u8 inputVector);

Purpose

Loads the axis target positions for the next vector space move.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
vectorSpace	u8	vector space to control
xPosition	i32	x axis target position in counts or steps
yPosition	i32	y axis target position in counts or steps
zPosition	i32	z axis target position in counts or steps
inputVector	u8	source of the data for this function

Parameter Discussion

vectorSpace is the vector space to control. Valid values are NIMC VECTOR SPACE1 through NIMC VECTOR SPACE15. On motion controllers that support fewer than thirty axes, configuring non-existent vector spaces returns error 70006 (NIMC badResourceIDOrAxisError). Refer to Vector Spaces for vector space resource IDs.

xPosition, **yPosition**, and **zPosition** are the specified axis target positions for the next vector space move in counts (servo axes) or steps (stepper axes). Target positions can be anywhere within the 32-bit position range, $-(2^{31})$ to $+(2^{31}-1)$. The default value for position is zero (0).



Caution Any single move is limited to $\pm (2^{31}-1)$ counts or steps on an axis. An error is generated if you exceed this limit by loading target position too far from the current axis positions.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to Input and Return Vectors for more detailed information.

Using This Function

This function loads up to three axis target positions for the vector space specified. This function is identical to calling the <u>Load Target Position</u> function up to three times, one time per each axis in the vector space. Position values indicate the specified end location and direction of motion (target position).

Target positions are double-buffered so you can load them on the fly without affecting the move in process, and they take effect on the next <u>Start Motion</u> or <u>Blend Motion</u> function. When the target positions are loaded, they are interpreted as either absolute target positions, relative target positions, target positions relative to the last captured positions or with the effect of a position modulus, depending on the mode set with the <u>Set Operation Mode</u> function.

This means that if you set the operation mode to NIMC_RELATIVE_TO_CAPTURE, you must load the target position after the capture has occurred because the position is evaluated on load.

After you execute the start or blend, the axes in the vector space follows the programmed trajectory and end up at the absolute, relative, or modulo target positions.

 $\overline{\mathbb{N}}$

Note If you are doing onboard programming and are using **inputVector** to get the data this function needs, note that this function reads the variables starting at the memory address pointed to by **inputVector** in the following order: **xPosition**, **yPosition**, **zPosition**.



Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

flex_read_axis_status

Device Compatibility

Read per Axis Status

Usage

status = flex_read_axis_status(u8 boardID, u8 axis, u8 returnVector);

Purpose

Reads the motion status on a per-axis basis.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to read
returnVector	u8	destination for the return data

Parameter Discussion

axis is the axis to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns the trajectory and motion I/O status for the specified axis. It also returns the success or failure status of the most recent Find <u>Reference</u> sequences.

This function returns any one of the values shown in the bitmap in the **Example** section.



Note You also can read individual item status in a multi-axis format with NI-Motion functions like <u>Read Axis Limit Status</u>, <u>Read Trajectory Status</u>, and so on.

Example

Read per Axis Status is called on axis 4 and the function returns axisStatus = 0xBE02.

The returned value 0xBE02 corresponds to the following bitmap:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2
MC	BC	Dir	HSC	IF	HF	ΒP	VT	Reserved	S/W	Home	Limit	FE	MC
									Limit				
1	0	1	1	1	1	1	0	0	0	0	0	0	0

The following list describes each of the bitmap elements:

D0 Run/Stop Status (R/S):

1 = Axis running

0 = Axis stopped

D1 Profile Complete (PC):

1= Profile complete

0 = Profile generation in process

D2 Motor Off (Moff):

- 1 = Motor off (killed)
- 0 = Motor on

D3 Following Error (FE):

1 = Axis exceeded the programmed following error limit

0 = Axis following error is below the programmed following error limit



Note If you are running a contoured move or slave axis move with a stepper axis, it is possible that the contour velocity or the slave axis geared velocity exceeds the maximum step rate of the controller. In this case, the controller kills the axis and sets the following error status to true. Refer to the *Specifications* section of your controller user manual for the maximum step rate of your controller.

D4 Limit Input (Limit):

1 = Forward or reverse limit input active

- 0 = Neither limit active
- D5 Home Input (Home):
 - 1 = Home input active
 - 0 = Home input not active
- D6 Software Limit (S/W Limit):
 - 1 = Forward or reverse software limit reached
 - 0 = Neither software limit reached
- D7 Reserved
- D8 Velocity Threshold (VT):
 - 1 = Velocity above threshold
 - 0 = Velocity below threshold
- D9 Breakpoint (BP):
 - 1 = Breakpoint occurred
 - 0 = Breakpoint pending or disabled
- D10 Home Found (HF):
 - 1 = Home found during last Find Home
 - 0 = Find Home sequence in process or home not found

D11 Index Found (IF):

- 1 = Encoder Index found during last Find Index
- 0 = Find Index sequence in process or index not found
- D12 High Speed Capture (HSC):
 - 1 = High speed capture occurred
 - 0 = High speed capture pending
- D13 Direction (Dir):
 - 0 = Reverse occurred
 - 1 = Forward

D14 Blend Complete (BC):

1 = Blend is complete occurred

0 = Blend is pending

D15 Move Complete (MC):

1 = Move is complete occurred

0 = Move is not complete

Device Compatibility

Read per Axis Status Return

Usage

status = flex_read_axis_status_rtn(u8 boardID, u8 axis, u16* axisStatus);

Purpose

Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to read
axisStatus	u16*	bitmap of per-axis status

Parameter Discussion

axis is the axis to read. Valid values are NIMC_AXIS1 through

NIMC_AXIS30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

axisStatus is a bitmap of motion status for the axis.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1
MC	BC	Dir	HSC	IF	HF	BP	VT	WD	S/W Limit	Home	Limit	FE	MOff	P(

D0 Run/Stop Status (R/S):

- 1 = Axis running
- 0 = Axis stopped

D1 Profile Complete (PC):

- 1= Profile complete
- 0 = Profile generation in process
- D2 Motor Off (Moff):
 - 1 = Motor off (killed)
 - 0 = Motor on
- D3 Following Error (FE):
 - 1 = Axis exceeded the programmed following error limit
 - 0 = Axis following error is below the programmed following error limit
- Note If you are running a contoured move or slave axis move with a stepper axis, it is possible that the contour velocity or the slave axis geared velocity exceeds the maximum step rate of the controller. In this case, the controller kills the axis and sets the following error status to true. Refer to the *Specifications* section of your controller user manual for the maximum step rate of your controller.
- D4 Limit Input (Limit):
 - 1 = Forward or reverse limit input active
 - 0 = Neither limit active
- D5 Home Input (Home):

1 = Home input active

- 0 = Home input not active
- D6 Software Limit (S/W Limit):
 - 1 = Forward or reverse software limit reached
 - 0 = Neither software limit reached
- D7 Communication Watchdog (WD):
 - 1 = Watchdog occurred.
 - 0 = Watchdog did not occur.
- D8 Velocity Threshold (VT):
 - 1 = Velocity above threshold
 - 0 = Velocity below threshold
- D9 Breakpoint (BP):
 - 1 = Breakpoint occurred
 - 0 = Breakpoint pending or disabled
- D10 Home Found (HF):
 - 1 = Home found during last Find Home
 - 0 = Find Home sequence in process or home not found
- D11 Index Found (IF):
 - 1 = Encoder Index found during last Find Index
 - 0 = Find Index sequence in process or index not found
- D12 High Speed Capture (HSC):
 - 1 = High speed capture occurred
 - 0 = High speed capture pending



Note If configured as a digital input, the high-speed capture line state does not affect this bit.

D13 Direction (Dir):

- 1 = Reverse
- 0 = Forward

D14 Blend Complete (BC):

- 1 = Blend complete
- 0 = Blend pending

D15 Move Complete (MC):

- 1 = Move complete
- 0 = Move not complete

Using This Function

This function returns the trajectory and motion I/O status for the specified axis. The valid status information this function returns is shown in the bitmap in **axisStatus**.

This function also returns the success or failure status of the most recent <u>Find Reference</u> sequences.



Note This function returns undefined values for axes that are currently performing find reference moves.

Note You also can read individual item status in a multi-axis format with NI-Motion functions like <u>Read Axis Limit Status</u>, <u>Read Trajectory Status</u>, and so on.

Example

Read per Axis Status is called on axis 4 and the function returns **axisStatus** = 0xBE02.

The returned value 0xBE02 corresponds to the following bitmap:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1
MC	BC	Dir	HSC	IF	HF	ΒP	VT	WD	S/W	Home	Limit	FE	MOff	PC
									Limit					
1	0	1	1	1	1	1	0	0	0	0	0	0	0	1

Refer to the <u>bitmap</u> descriptions above for information about what each true (1) and false (0) value means in this example bitmap.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

• In the **axisStatus** bitmap, D7 Communication Watchdog (WD) is not supported by 73xx controllers.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- When you are using the NI SoftMotion Controller with a device that supports multiple high-speed captures and breakpoints, the status for primary and secondary inputs is concatenated by a logical OR operation to populate **axisStatus**.
- In the **axisStatus** bitmap, D7 Communication Watchdog (WD) is used for a communication status watchdog on the NI SoftMotion Controller.

flex_read_pos

Device Compatibility

Read Position

Usage

status = flex_read_pos(u8 boardID, u8 axis, u8 returnVector);

Purpose

Reads the position of an axis.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to read
returnVector	u8	destination for the return data

Parameter Discussion

axis is the axis to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns the instantaneous position of the specified axis. For servo axes, it returns the primary feedback position in counts. For openloop stepper axes, it returns the number of steps generated. For closedloop stepper axes, it converts the primary feedback position from counts to steps and then returns the value in steps. Closed-loop stepper axes require you to correctly load values of steps per revolution and counts per revolution to function correctly.



Note For closed-loop axes, this function always returns the position of the primary feedback resource. Refer to the <u>Read</u> <u>Vector Space Position</u> function for an easy way to read up to three axis positions for a vector space in one call.



Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

flex_read_pos_rtn

Device Compatibility

Read Position Return

Usage

status = flex_read_pos_rtn (u8 boardID, u8 axis, i32* position);

Purpose

Reads the position of an axis.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to read
position	i32*	axis position in counts (servo) or steps (stepper)

Parameter Discussion

axis is the axis to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

position is the axis position in quadrature counts (for servo axes) or steps (for stepper axes).
Using This Function

This function returns the instantaneous position of the specified axis. For servo axes, it returns the primary feedback position in counts. For openloop stepper axes, it returns the number of steps generated. For closedloop stepper axes, it converts the primary feedback position from counts to steps and then returns the value in steps. Closed-loop stepper axes require you to correctly load values of steps per revolution and counts per revolution to function correctly.



Note For closed-loop axes, this function always returns the position of the primary feedback resource. Refer to the <u>Read</u>
 <u>Vector Space Position</u> function for an easy way to read up to three axis positions for a vector space in one call.

flex_read_rpm

Device Compatibility

Read Velocity in RPM

Usage

status = flex_read_rpm(u8 boardID, u8 axisOrVectorSpace, u8 returnVector);

Purpose

Reads the filtered velocity of an axis or vector space in RPM.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to read
returnVector	u8	destination for the return data

Parameter Discussion

axisOrVectorSpace is the axis or vector space to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, reading non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector</u> <u>Spaces</u> for axis and vector space resource IDs.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns the axis or vector space filtered velocity in RPM. The filtered velocity is measured based on time elapsed between position changes. To avoid quantization error and jitter in the mechanical system, a filtering algorithm in <u>Load Velocity Filter Parameter</u> is performed with parameters configured.

For vector spaces, this function returns vector velocity, the root-meansquare of the filtered velocities of the individual axes that make up the vector space.



Note This function requires previously loaded values of either counts per revolution (for servo axes) or steps per revolution (for stepper axes) to operate correctly.

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

flex_read_rpm_rtn

Device Compatibility

Read Velocity in RPM Return

Usage

status = flex_read_rpm_rtn(u8 boardID, u8 axisOrVectorSpace, f64* RPM);

Purpose

Reads the filtered velocity of an axis or vector space in RPM.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to read
RPM	f64*	filtered velocity in RPM

Parameter Discussion

axisOrVectorSpace is the axis or vector space to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, reading non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector</u> <u>Spaces</u> for axis and vector space resource IDs.

RPM is the filtered velocity value in RPM expressed as a doubleprecision floating point number. For vector spaces, **RPM** is the filtered vector velocity for the vector move. The sign of **RPM** indicates direction of motion.

Using This Function

This function returns the axis or vector space filtered velocity in RPM. The filtered velocity is measured based on time elapsed between position changes. To avoid quantization error and jitter in the mechanical system, a filtering algorithm in <u>Load Velocity Filter Parameter</u> is performed with parameters configured.

For vector spaces, this function returns vector velocity, the root-meansquare of the filtered velocities of the individual axes that make up the vector space.



Note This function requires previously loaded values of either counts per revolution (for servo axes) or steps per revolution (for stepper axes) to operate correctly.

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

flex_read_velocity

Device Compatibility

Read Velocity

Usage

status = flex_read_velocity(u8 boardID, u8 axisOrVectorSpace, u8
returnVector);

Purpose

Reads the filtered velocity of an axis or vector space.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to read
returnVector	u8	destination for the return data

Parameter Discussion

axisOrVectorSpace is the axis or vector space to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, reading non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector</u> <u>Spaces</u> for axis and vector space resource IDs.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns the axis or vector space filtered velocity. To reduce the effects of noise and jitter on the filter calculation, the data this function returns is averaged based on parameters set in the Load Velocity Filter Parameter function.

For vector spaces, this function returns vector velocity, the root-meansquare of the filtered velocities of the individual axes that make up the vector space.



Note You also can read velocity in RPM by calling the <u>Read</u> <u>Velocity in RPM</u> function.

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

flex_read_velocity_rtn

Device Compatibility

Read Velocity Return

Usage

status = flex_read_velocity_rtn(u8 boardID, u8 axisOrVectorSpace, i32*
velocity);

Purpose

Reads the filtered velocity of an axis or vector space.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to read
velocity	i32*	axis or vector space filtered velocity in counts/s (servo) or steps/s (stepper)

Parameter Discussion

axisOrVectorSpace is the axis or vector space to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, reading non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector</u> <u>Spaces</u> for axis and vector space resource IDs.

velocity is filtered velocity in counts/s (for servo axes) or steps/s (for stepper axes). The sign of velocity indicates direction of motion. For vector spaces, velocity is the filtered vector velocity for the vector move.

Using This Function

This function returns the axis or vector space filtered velocity. To reduce the effects of noise and jitter on the filter calculation, the data this function returns is averaged based on parameters set in the Load Velocity Filter Parameter function.

For vector spaces, this function returns vector velocity, the root-meansquare of the filtered velocities of the individual axes that make up the vector space.



Note You also can read velocity in RPM by calling the <u>Read</u> <u>Velocity in RPM</u> function.

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

flex_read_vs_pos

Device Compatibility

Read Vector Space Position

Usage

status = flex_read_vs_pos(u8 boardID, u8 vectorSpace, u8 returnVector);

Purpose

Reads the position of all axes in a vector space.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
vectorSpace	u8	vector space to read
returnVector	u8	destination for the return data

Parameter Discussion

vectorSpace is the vector space to read. Valid values are NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, reading non-existent vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Vector Spaces</u> for vector space resource IDs.

inputVector indicates the source data for this function. Available input vectors include input data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to Input and Return Vectors for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns the instantaneous positions of the axes in the specified vector space. For servo axes, it returns the primary feedback position in counts. For open-loop stepper axes, it returns the number of steps generated. For closed-loop stepper axes, it converts the primary feedback position from counts to steps and then returns the value in steps. Closed-loop stepper axes require correctly loaded values of steps per revolution and counts per revolution to function correctly.



Note For closed-loop axes, this function always returns the position of the primary feedback resource.

flex_read_vs_pos_rtn

Device Compatibility

Read Vector Space Position Return

Usage

status = flex_read_vs_pos_rtn(u8 boardID, u8 vectorSpace, i32* xPosition, i32*
yPosition, i32* zPosition);

Purpose

Reads the position of all axes in a vector space.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
vectorSpace	u8	vector space to read
xPosition	i32*	x axis position in counts or steps
yPosition	i32*	y axis position in counts or steps
zPosition	i32*	z axis position in counts or steps

Parameter Discussion

vectorSpace is the vector space to read. Valid values are NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, reading non-existent vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Vector Spaces</u> for vector space resource IDs.

xPosition, **yPosition**, and **zPosition** are the positions in quadrature counts (for servo axes) or steps (for stepper axes) of the three axes in the vector space. For vector spaces with less than three axes, zero (0) is returned on the unused axes.
Using This Function

This function returns the instantaneous positions of the axes in the specified vector space. For servo axes, it returns the primary feedback position in counts. For open-loop stepper axes, it returns the number of steps generated. For closed-loop stepper axes, it converts the primary feedback position from counts to steps and then returns the value in steps. Closed-loop stepper axes require correctly loaded values of steps per revolution and counts per revolution to function correctly.



Note For closed-loop axes, this function always returns the position of the primary feedback resource.

flex_reset_pos

Device Compatibility

Reset Position

Usage

status = flex_reset_pos(u8 boardID, u8 axis, i32 position1, i32 position2, u8
inputVector);

Purpose

Resets the axis position to the specified position, taking following error into account.



Note If you are using this function with an NI 7350 or NI 7390 device that you are <u>reading positions on</u>, it might take up to 5 ms for the application to display the most recently read position after you call this function.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to control
position1	i32	reset value for axis and primary feedback resource
position2	i32	reset value for secondary feedback resource
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

position1 is the reset value for the axis and its associated primary feedback resource, taking following error into account. You can reset position to any value in the total position range of $-(2^{31})$ to $+(2^{31}-1)$.

position2 is the reset value for the optional secondary feedback resource. You can reset position to any value in the total position range of $-(2^{31})$ to $+(2^{31}-1)$.



Note For stepper closed-loop configurations, where the encoder counts per revolution is greater than the steps per revolution, the range of the position parameters is reduced by the ratio of counts/rev to steps/rev. For example, if the ratio is 4,000 counts per revolution to 2,000 steps per revolution (2:1 ratio), then the range of position parameters is divided by 2.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function resets the axis position and the associated primary feedback position, taking following error into account, and resets the optional secondary feedback position. This means that if you reset the position to 0 and have a following error of 15, the feedback position is -15.

You can reset position to zero or any value in the 32-bit position range. You can reset the secondary feedback to the same value as the axis and primary feedback or you can reset it to a different value. If a secondary feedback resource is not in use, the corresponding reset value is ignored.



Note Non-zero reset values are useful for defining a position reference offset.

Position can be reset at any time. However, it is recommended that you reset position only while the axis is stopped. An axis reset while the axis is moving does not have a repeatable reference position.



Note For axes in p-command mode, ensure that the axis is not moving before calling Reset Position. Calling Reset Position when the axis is moving introduces position error into the system.

Typically, you execute this function one time after the Find Home and Find Index sequences of the Find Reference function have completed successfully to set the zero position for the system and do not call it again until the next power-up.



Note Enabled breakpoints are automatically disabled when you execute a Reset Position or <u>Reset Encoder Position</u> function on the corresponding axis.

An ADC channel used as a primary feedback resource is reset by storing an offset value when this function is executed. In this way, its zero reference is not lost and you can still read the actual ADC value with the <u>Read ADC</u> function.

Resetting position sets the value returned by steps/revolution to the primary reset position.



Note National Instruments recommends disabling gearing before resetting the position of the master axis or the slave axis. When

the position of either the master axis or the slave axis is reset, the reset offset is treated as gearing offset, and the slave axis jumps to the new gear position in one PID period.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- The secondary reset position is ignored.
- When you reset a slave position, gearing is not disabled.
 - Note National Instruments recommends that you do *not* reset the position when an axis is moving or gearing is enabled because doing so might cause the axis to jump and/or cause a following error.

flex_set_op_mode

Device Compatibility

Set Operation Mode

Usage

status = flex_set_op_mode(u8 boardID, u8 axisOrVectorSpace, u16
operationMode);

Purpose

Sets the operation mode for an axis or vector space.

Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to control
operationMode	u16	mode of operation

Parameter Discussion

axisOrVectorSpace is the axis or vector space to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

operationMode selects the type of position or velocity mode for an axis or vector space.

operationMode Constant	operationMode Value
NIMC_ABSOLUTE_POSITION	0
NIMC_RELATIVE_POSITION	1
NIMC_VELOCITY	2
NIMC_RELATIVE_TO_CAPTURE	3
NIMC_MODULUS_POSITION	4
NIMC_ABSOLUTE_CONTOURING	5
NIMC_RELATIVE_CONTOURING	6

Using This Function

This function is used both during initialization and during normal motion control operation to configure the mode of operation for all trajectory commands to the axis or vector space specified.

Position modes are applied to all axes in a coordinate (vector) space. If you later want to operate an axis independently in a different mode from the other axes, call Set Operation Mode again on that axis.



Note All axes in a vector space must have the same operation mode. If the operation modes are different on each axis when a <u>Start Motion</u> or <u>Blend Motion</u> function is executed, an error is generated.

The operation mode must be set or changed before any other trajectory parameters are loaded for the next move. The operation mode affects how the target position and velocity values are interpreted. Trajectory parameters loaded after a mode change are interpreted in the newly selected mode. Trajectory parameters loaded any time before a mode change do not reflect the new mode.



Note Changing operation mode after the trajectory parameters are loaded can result in improper operation.

The five operation modes are described in the following sections:

NIMC_ABSOLUTE_POSITION

In absolute position mode, target positions are interpreted with respect to an origin, reference, or zero position. The origin is typically set at a home switch, end of travel limit switch, or encoder index position. An absolute position move uses the preprogrammed values of acceleration, deceleration, s-curve, and velocity to complete a trajectory profile with an ending position equal to the loaded absolute target position. **Caution** Any single move is limited to between -2^{31} and $2^{31}-1$ counts or steps. An error is generated if you exceed this limit by loading a target position too far from the current position.

The length of an absolute move depends upon the loaded position and the current position when the move is started. If the target position is the same as the current position, no move occurs.

NIMC_RELATIVE_POSITION

In relative position mode while motion is not in progress, loaded target positions are interpreted with respect to the current position at the time the value is loaded. A relative position move uses the preprogrammed values of acceleration, deceleration, s-curve and velocity to complete a trajectory profile with an ending position equal to the sum of the loaded relative target position and the starting position.

If a relative move is started while motion is in progress, the new target position is calculated with respect to the target position of the move already in progress (considered to be the new starting position), as if that move had already completed successfully. Motion continues to the new relative position, independent of the actual position location when the new move is started.

In relative mode, the new target position is calculated and doublebuffered when you execute either the <u>Load Target Position</u> or <u>Load</u> <u>Vector Space Position</u> function. You must reload the relative target position each time before executing a <u>Start Motion</u> or <u>Blend Motion</u> function.

NIMC_VELOCITY

In velocity mode, the axis moves at the loaded velocity until you execute a Stop Motion function, a limit is encountered, or a new velocity is loaded and you execute a Start Motion function. Load target positions have no effect in velocity mode. The direction of motion is determined by the sign of the loaded velocity.

You can update velocity at any time to accomplish velocity profiling. Changes in velocity while motion is in progress uses the preprogrammed acceleration, deceleration, and s-curve values to control the change in velocity. You can reverse direction by changing the sign of the loaded velocity and executing a Start Motion function.



Note Executing a Blend Motion function in velocity mode has no effect because the move in process never normally stops. You must always use the Start Motion function to update velocity in velocity mode.

NIMC_RELATIVE_TO_CAPTURE

The relative-to-capture position mode is very similar to absolute position mode, except that the zero position reference (origin) is the last captured position for the axis or axes. A relative-to-capture position move uses the preprogrammed values of acceleration, deceleration, s-curve and velocity to complete a trajectory profile with an ending position equal to the sum of the loaded target position and the last captured position.

In relative-to-capture mode, the new target position is calculated and double-buffered when you execute either the <u>Load Target Position</u> or <u>Load Vector Space Position</u> function. These functions use existing values in the position capture register(s). You must load the target position after the capture event has occurred and before executing the Start Motion or Blend Motion function.

This mode is typically used in registration applications. Refer to the Gearing section of the *NI-Motion Help* for information about using superimposed moves/registration applications. Also, refer to the Rotating Knife section of the *NI-Motion Help* for example code that includes superimposed moves.

NIMC_MODULUS_POSITION

In modulus position mode, the loaded target position is interpreted within the boundaries of a modulus range and the direction of motion is automatically chosen to generate the shortest trajectory to the target. To load the modulus range execute the <u>Load Position Modulus</u> function.

Modulus position mode is typically used with rotary axes or for other similarly repetitive motion applications.



Note Multiple revolution moves cannot be accomplished by indicating target positions greater than the modulus value. All moves are resolved to one modulus range.

Example

A rotary tool changer has a modulus of 360° , such that 0° , 360° , 720° , and so on, are the same rotary position.

In modulus position mode, the present position and the specified target position are used to calculate the shortest trajectory to the target position. If the present position is 30° and the target position is 290°, there are two possible moves:

 $290-30 = 260^{\circ}$ in the clockwise direction, or $290-360-30 = -100^{\circ}$ in the counterclockwise direction.

Because 100° is the shortest trajectory, the tool changer moves counterclockwise to the target position of 290°.

NIMC_ABSOLUTE_CONTOURING and NIMC_RELATIVE_CONTOURING

The absolute and relative contouring modes allow you to provide the motion controller with a series of points to spline through. These modes allow you to specify arbitrary motion and velocity profiles. Velocity, acceleration, and deceleration are all determined implicitly by the contour points you provide.



Note If you are running a contoured move with a stepper axis, it is possible that the contour velocity exceeds the maximum step rate of the controller. In this case, the controller kills the axis and sets the following error status to true. Refer to the *Specifications* section of your controller user manual for the maximum step rate of your controller.

To use contouring mode, you must also configure a buffer with the <u>Configure Buffer</u> function and fill the buffer with contouring data. Successive points in the buffer are fed to the motion controller automatically at the user-specified time interval in milliseconds. The motion controller then splines through the contour points. Refer to the Configure Buffer function for more information.

All contour moves are relative, meaning that the move starts from the current position of the axis or axes. In absolute contouring mode, positions are interpreted with respect to the starting position of the contouring move. In relative contouring mode, positions are interpreted with respect to the previous position in the contouring data array.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- NI 7330 and NI 7390 controllers do not support absolute or relative contouring.
- NI 73xx controllers do not support NIMC_VELOCITY for vector spaces.

NI SoftMotion Controller Considerations

The following includes considerations you must make when you are using this function with the NI SoftMotion Controller:

The NI SoftMotion Controller does not support the NIMC_MODULUS_POSITION option for **operationMode**.

Wait for Blend Complete

Usage

status = flex_wait_for_blend_complete(u8 boardID, u8 axisOrVectorSpace, u16 axisOrVSMap, u32 timeout, i32 pollInterval, u16* blendComplete);

Purpose

Waits up to the specified period of time for a blend to complete on an axis, vector space, group of axes, or group of vector spaces.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description				
boardID	u8	assigned by Measurement & Automation Explorer (MAX)				
axisOrVectorSpace	u8	axis or vector space selector				
axisOrVSMap	u16	bitmap of axes or vector spaces to check				
timeout	u32	timeout in milliseconds				
pollInterval	i32	polling interval in milliseconds				
blendComplete	u16*	the blend complete status				

Parameter Discussion

axisOrVectorSpace can select an axis (NIMC_AXIS1 through NIMC_AXIS15), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously waiting for a blend to complete on multiple axes or vector spaces, the **axisOrVSMap** parameter indicates which axes or vector spaces to wait for. On motion controllers that support fewer than fifteen axes, waiting for non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

axisOrVSMap is the bitmap of axes or vector spaces to wait for. It is only required when multiple axes or vector spaces are selected with the **axisOrVectorSpace** parameter. Otherwise, this parameter is ignored. NI-Motion ignores additional axes and vector spaces if you select non-existent axes.

When waiting for multiple axes (**axisOrVectorSpace** = NIMC_AXIS_CTRL):

D15	•••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis		Axis 10	Axis 9	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	0

For D1 through D15:

1 = Blend must be complete on specified axis

0 = Blend can be either complete or not complete on specified axis (do not care)

When waiting for multiple vector spaces (**axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL):

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS 15	 VS 10	VS 9	VS 8	VS 7	VS 6	VS 5	VS 4	VS 3	VS 2	VS 1	0

For D1 through D15:

1 = Blend must be complete on specified vector space

0 = Blend can be either complete or not complete on specified vector space (do not care)

To wait for blend complete on a single axis or vector space, set the

axisOrVectorSpace selector to the specified axis or vector space. The **axisOrVSMap** parameter is ignored.

To wait for blend complete on multiple axes, the **axisOrVectorSpace** selector is set to NIMC_AXIS_CTRL and the **axisOrVSMap** bitmap defines the axes to wait for. They must all be blend complete, for the **blendComplete** output to be true. Similarly, to wait for blend complete on multiple vector spaces, the **axisOrVectorSpace** selector is set to NIMC_VECTOR_SPACE_CTRL and the **axisOrVSMap** bitmap defines the vector spaces to wait for.

timeout is the amount of time, in milliseconds, to wait for the blend to become complete.

pollInterval is the amount of time, in milliseconds, to wait between successive queries to the controller to determine if the blend is complete.

blendComplete indicates if the blend is complete on the axes or vector spaces specified:

- 1 = Blend complete
- 0 = Blend not complete

Using This Function

This utility function is built on top of the <u>Check Blend Complete Status</u> function, and is provided for your programming convenience. This function compares the axes or vector spaces specified in the **axisOrVectorSpace** and/or **axisOrVSMap** input parameters with the blend complete status for the appropriate axes or vector spaces. It does this repeatedly, with the **pollInterval** time determining the frequency that the controller is queried. When the blend is complete, this function returns with **blendComplete** set to true (1), otherwise, it waits for the amount of time specified by **timeout**. If the function waits longer than **timeout**, it returns NIMC_eventTimeoutError, and **blendComplete** is set to false (0).

The output is a single true/false value indicating if the specified blend or blends are complete.

This function does not check for following error status or axis off status. Checking for faults during moves is recommended. To do this, use <u>Read</u> <u>per Axis Status</u>, which returns following error and axis off status in addition to the move complete status. Refer to the <u>NI-Motion Help</u> for more information and example code.

For more information about blend complete status, refer to the <u>Read</u> <u>Blend Status</u> function.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following includes considerations you must make when you are using this function with the NI SoftMotion Controller:

To wait for blend complete on axes sixteen through thirty, use the <u>Read</u> <u>per Axis Status</u> function.

Wait for Move Complete

Usage

status = flex_wait_for_move_complete(u8 boardID, u8 axisOrVectorSpace, u16 axisOrVSMap, u32 timeout, i32 pollInterval, u16* moveComplete);

Purpose

Waits up to the specified period of time for a move to complete on an axis, vector space, group of axes, or group of vector spaces.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space selector
axisOrVSMap	u16	bitmap of axes or vector spaces to check
timeout	u32	timeout in milliseconds
pollInterval	i32	polling interval in milliseconds
moveComplete	u16*	the move complete status

Parameter Discussion

axisOrVectorSpace can select an axis (NIMC_AXIS1 through NIMC_AXIS15), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously waiting for a move to complete on multiple axes or vector spaces, the **axisOrVSMap** parameter indicates which axes or vector spaces to wait for. On motion controllers that support fewer than fifteen axes, waiting for non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

axisOrVSMap is the bitmap of axes or vector spaces to wait for. It is only required when multiple axes or vector spaces are selected with the **axisOrVectorSpace** parameter. Otherwise, this parameter is ignored. NI-Motion ignores additional axes and vector spaces if you select non-existent axes.

When waiting for multiple axes (**axisOrVectorSpace** = NIMC_AXIS_CTRL):

D15	••••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis 15		Axis 10	Axis 9	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	0

For D1 through D15:

1 = Move must be complete on specified axis

0 = Move can be either complete or not complete on specified axis (do not care)

When waiting for multiple vector spaces (**axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL):

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS 15	 VS 10	VS 9	VS 8	VS 7	VS 6	VS 5	VS 4	VS 3	VS 2	VS 1	0

For D1 through D15:

1 = Move must be complete on specified vector space

0 = Move can be either complete or not complete on specified vector space (do not care)

To wait for move complete on a single axis or vector space, set the

axisOrVectorSpace selector to the specified axis or vector space. The **axisOrVSMap** parameter is ignored.

To wait for move complete on multiple axes, the **axisOrVectorSpace** selector is set to NIMC_AXIS_CTRL and the **axisOrVSMap** bitmap defines the axes to wait for. They must all be move complete, for the **moveComplete** output to be true. Similarly, to wait for move complete on multiple vector spaces, the **axisOrVectorSpace** selector is set to NIMC_VECTOR_SPACE_CTRL and the **axisOrVSMap** bitmap defines the vector spaces to wait for.

timeout is the amount of time, in milliseconds, to wait for the move to become complete.

pollInterval is the amount of time, in milliseconds, to wait between successive queries to the controller to determine if the move is complete.

moveComplete indicates if the move is complete on the axes or vector spaces specified:

- 1 = Move complete
- 0 = Move not complete

Using This Function

This utility function is built on top of the <u>Check Move Complete Status</u> function, and is provided for your programming convenience. This function compares the axes or vector spaces specified in the **axisOrVectorSpace** and/or **axisOrVSMap** input parameters with the move complete status for the appropriate axes or vector spaces. It does this repeatedly, with the **pollInterval** time determining the frequency that the controller is queried. When the move is complete, this function returns with the **moveComplete** parameter set to true (1), otherwise it waits for the amount of time specified by the **timeout** parameter. If the function waits longer than **timeout** it returns NIMC_eventTimeoutError, and **moveComplete** is set to false (0).

The output is a single true/false value indicating if the specified move or moves are complete.

This function does not check for following error status or axis off status. Checking for faults during moves is recommended. To do this, use <u>Read</u> <u>per Axis Status</u>, which returns following error and axis off status in addition to the move complete status. Refer to the <u>NI-Motion Help</u> for more information and example code.

Refer to the <u>Read Trajectory Status</u> and <u>Configure Move Complete</u> <u>Criteria</u> functions for more information about move complete status.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following includes considerations you must make when you are using this function with the NI SoftMotion Controller:

To wait for move complete on axes sixteen through thirty, use the <u>Read</u> <u>per Axis Status</u> function.

Arc

You can use Arc functions to load parameters for circularly interpolated moves. It includes 2D circular arcs, 3D helical arcs, and full 3D spherical arcs functions.

Circular interpolation is an advanced feature of NI-Motion and is primarily used in continuous path applications such as machining, pattern cutting, liquid dispensing, robotics, and so on. For maximum smoothness and accuracy, the NI-Motion DSP implements arcs through a cubic spline algorithm.

Arc functions are always sent to a vector space. Velocity and acceleration parameters loaded by executing those functions on the vector space are used as the vector velocity and vector acceleration for all subsequent arc moves. All arc parameters are double-buffered and take effect upon the next Start Motion or Blend Motion functions execution.



Note Arc radius determines the practical range for vector acceleration and velocity. Unrealizable vector values generate an error and the start or blend does not execute.

You can blend arc moves into linearly interpolated moves and vice versa. You also can load all axes in the vector space in the same blend factor with the <u>Load Blend Factor</u> function.

Arc moves are defined relative to their starting position and as such, are inherently operated in relative position mode. This approach guarantees that the axes are already on the circle in the x'y' plane, and avoids any impossible situations where the end point of the last move and the beginning of the arc move are not coincident. The mode selected with the <u>Set Operation Mode</u> function has no effect on the arc move. It can, however, affect the linearly interpolated vector move you might be blending into or from.

Load Circular Arc

Load Spherical Arc

Load Helical Arc
flex_load_circular_arc

Device Compatibility

Load Circular Arc

Usage

status = flex_load_circular_arc(u8 boardID, u8 vectorSpace, u32 radius, f64 startAngle, f64 travelAngle, u8 inputVector);

Purpose

Loads parameters for making a circular arc move in a 2D or 3D vector space.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
vectorSpace	u8	vector space to control
radius	u32	arc radius in counts or steps
startAngle	f64	starting angle for the arc move in the xy plane in degrees
travelAngle	f64	travel angle for the arc move in the xy plane in degrees
inputVector	u8	source of the data for this function

Parameter Discussion

vectorSpace is the vector space to control. Valid values are NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, configuring non-existent vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Vector Spaces</u> for vector space resource IDs.

radius is the arc radius in counts (servo axes) or steps (stepper axes).

startAngle is the double precision floating point value, in degrees, of the starting angle of the arc.

travelAngle is the double precision floating point value, in degrees, of the angle to traverse. A positive **travelAngle** defines counterclockwise rotation in the xy plane.

Note Internally, the floating point values for **startAngle** and **travelAngle** are represented as scaled, fixed point numbers.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The conversion from floating-point to fixed-point is performed on the host computer, not on the motion controller. To load arc functions from onboard variables, you must use the 16.16 fixed-point representation for all angles.

Using This Function

This function defines an arc in the xy plane of a 2D or 3D vector space.



Note Load the velocity and acceleration/deceleration before loading an arc.

The arc is specified by a radius, starting angle, and travel angle and like all vector space moves, uses the loaded values of vector acceleration and vector velocity to define the motion along the path of the arc. The following figure defines a circular arc:

1	Circular Arc	3	Starting Position	5	Radius
2	Travel Angle	4	Start Angle	6	Ending Position

Circular arcs are not limited to $\pm 360^{\circ}$. Moves of over 4,000 circular revolutions in either direction can be started with one call to this function.

Refer to <u>NI 73xx Arc Angles in Degrees</u> for more information about representing angles numerically.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- The range for **radius** is 2 to 2^{31} -1 counts (steps).
- The range for **startAngle** is 0 to 359.999313° where angle 0 is along the positive x axis and values increase counterclockwise from the positive x axis in the xy plane.
- The range for **travelAngle** is -1,474,560° to +1,474,200° (-4,096 to +4,095 revolutions).

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- The range for **radius** is 0 to 2^{31} -1 counts (steps).
- The range for **startAngle** is 0 to 360° where angle 0 is along the positive x axis and values increase counterclockwise from the positive x axis in the xy plane.
- The range for **travelAngle** is the full 64-bit range.

flex_load_helical_arc

Device Compatibility

Load Helical Arc

Usage

status = flex_load_helical_arc(u8 boardID, u8 vectorSpace, u32 radius, f64 startAngle, f64 travelAngle, i32 linearTravel, u8 inputVector);

Purpose

Loads parameters for making a helical arc move in a 3D vector space.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
vectorSpace	u8	vector space to control
radius	u32	arc radius in counts or steps
startAngle	f64	starting angle for the arc move in the xy plane in degrees
travelAngle	f64	travel angle for the arc move in the xy plane in degrees
linearTravel	i32	linear travel of the z axis in counts or steps
inputVector	u8	source of the data for this function

Parameter Discussion

vectorSpace is the vector space to control. Valid values are NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, configuring non-existent vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Vector Spaces</u> for vector space resource IDs.

radius is the arc radius in counts (servo axes) or steps (stepper axes).

startAngle is the double precision floating point value, in degrees, of the starting angle of the arc.

travelAngle is the double precision floating point value, in degrees, of the angle to traverse. A positive **travelAngle** defines counterclockwise rotation in the xy plane.

Note Internally, the floating point values for **startAngle** and **travelAngle** are represented as scaled, fixed point numbers.

linearTravel is the linear travel of the z axis in counts (servo axes) or steps (stepper axes).

Note Loading a zero (0) for **linearTravel** reduces the helical arc to a circular arc.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to Input and Return Vectors for more detailed information.



N

Note The conversion from floating-point to fixed-point is performed on the host computer, not on the motion controller. To load arc functions from onboard variables, you must use the 16.16 fixed-point representation for all angles.

Using This Function

This function defines an arc in 3D vector space that consists of a circle in the xy plane and synchronized linear travel in the z axis.



Note Load the velocity and acceleration/deceleration before loading an arc.

The arc is specified by a radius, starting angle, travel angle, and z axis linear travel, and like all vector space moves, uses the loaded values of vector acceleration and vector velocity to define the motion along the helical path of the arc. The following figure defines a helical arc:

1	Helical Arc	4	Starting Position	6	Linear Travel
2	Travel Angle	5	Radius	7	Ending Position
3	Start Angle				

Like circular arcs, helical arcs are not limited to $\pm 360^{\circ}$. Moves of up to 4,096 helical twists in either direction can be started with one call to this function.

M

Note If you are doing onboard programming and are using **inputVector** to get the data this function needs, remember that this function reads the variables starting at the memory address pointed to by **inputVector** in the following order: **radius**, **startAngle**, **travelAngle**, **linearTravel**.

Refer to <u>NI 73xx Arc Angles in Degrees</u> for more information about representing angles numerically.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- The range for **radius** is 2 to $2^{31}-1$ counts (steps).
- The range for **startAngle** is 0 to 359.999313° where angle 0 is along the positive x axis and values increase counterclockwise from the positive x axis in the xy plane.
- The range for **travelAngle** is -1,474,560° to +1,474,200° (-4,096 to +4,095 revolutions).
- The range for **linearTravel** is $-(2^{31})$ to $+(2^{31}-1)$ counts (steps).

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- The range for **radius** is 0 to 2^{31} -1 counts (steps).
- The range for **startAngle** is 0 to 360° where angle 0 is along the positive x axis and values increase counterclockwise from the positive x axis in the xy plane.
- The range for **travelAngle** is the full 64-bit range.
- The range for **linearTravel** is -2^{31} to $+(2^{31}-1)$.

flex_load_spherical_arc

Device Compatibility

Load Spherical Arc

Usage

status = flex_load_spherical_arc(u8 boardID, u8 vectorSpace, u32 radius, f64
planePitch, f64 planeYaw, f64 startAngle, f64 travelAngle, u8 inputVector);

Purpose

Loads parameters for making a spherical arc move in a 3D vector space.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
vectorSpace	u8	vector space to control
radius	u32	arc radius in counts or steps
planePitch	f64	angle between the x' and x axes when the entire x'y'z' vector space is rotated around the y axis
planeYaw	f64	angle between the x' and x axes when the entire x'y'z' vector space is rotated around the z axis
startAngle	f64	starting angle for the arc move in the x'y' plane in degrees
travelAngle	f64	travel angle for the arc move in the x'y' plane in degrees
inputVector	u8	source of the data for this function

Parameter Discussion

vectorSpace is the vector space to control. Valid values are NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, configuring non-existent vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Vector Spaces</u> for vector space resource IDs.

radius is the arc radius in counts (servo axes) or steps (stepper axes).

planePitch is the double precision floating point value, in degrees, of the angle between the x' and x axes when the entire x'y'z' vector space is rotated around the y axis. The y' axis remains aligned with the y axis. When **planePitch** equals 90°, the positive x' axis is aligned with the negative z axis.

planeYaw is the double precision floating point value, in degrees, of the angle between the x' and x axes when the entire x'y'z' vector space is rotated around the z axis. The z' axis remains aligned with the z axis. When **planeYaw** equals 90°, the positive x' axis is aligned with the positive y axis.

startAngle is the double precision floating point value, in degrees, of the starting angle of the arc.

travelAngle is the double precision floating point value, in degrees, of the angle to traverse. A positive **travelAngle** defines counterclockwise rotation in the xy plane.

 $\overline{\mathbb{N}}$

Note Internally, the floating point values for **startAngle** and **travelAngle** are represented as scaled, fixed-point numbers.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to Input and Return Vectors for more detailed information.



Note The conversion from floating-point to fixed-point is performed on the host computer, not on the motion controller. To load arc functions from onboard variables, you must use the 16.16 fixed-point representation for all angles.

Using This Function

This function defines an arc in the x'y' plane of a coordinate system that has to be transformed by rotation in pitch and yaw from the normal 3D vector space (xyz). In the transformed x'y'z' space, the spherical arc is reduced to a simpler circular arc.



Note Load the velocity and acceleration/deceleration before loading an arc.

The spherical arc is specified by a radius, starting angle, and travel angle, and like all vector space moves, uses the loaded values of vector acceleration and vector velocity to define the motion along the path of the arc in the x'y' plane. The following figure shows a graphic representation of the transformation between the x'y'z' and xyz coordinate spaces:

Yaw and pitch transformations are inherently confusing because they interact. To avoid ambiguities, you can think about spherical arcs and coordinate transformations as follows:

- The spherical arc is defined as a circular arc in the x'y' plane of a transformed vector space x'y'z'. The original vector space xyz is defined by the <u>Configure Vector Space</u> function.
- The transformed vector space x'y'z' is defined in orientation only, with no absolute position offset. Its orientation is with respect to the xyz vector space and is defined in terms of yaw and pitch angles.
- Yaw angle rotation comes before pitch angle rotation.
- When rotating through the yaw angle, the y' axis never leaves the original xy plane as the newly defined x'y'z' vector space rotates around the original z axis.
- When rotating through the pitch angle, the y and y' axes stay aligned with each other while the x'z' plane rotates around them.
- At the beginning of the move, the axes are considered to be already on the arc in the x'y' plane. This avoids any impossible situations where the end point of the last move and the beginning of the arc move are not coincident.
- Spherical arcs allow full 3D curvilinear motion for robotic, solid modeling, and other advanced applications.



Note If you are doing onboard programming and are using **Inp Vect** to get the data this function needs, remember that this function reads the variables starting at the memory address pointed to by **Inp Vect** in the following order: radius, planePitch, planeYaw, startAngle, travelAngle.

Refer to <u>NI 73xx Arc Angles in Degrees</u> for more information about representing angles numerically.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- The range for **radius** is 2 to 2^{31} -1 counts (steps).
- The range for **planePitch** is 0 to 90°.
- The range for **planeYaw** is 0 to 359.999313°.
- The range for **startAngle** is 0 to 359.999313° where angle 0 is along the positive x axis and values increase counterclockwise from the positive x axis in the xy plane.
- The range for **travelAngle** is -1,474,560° to +1,474,200° (-4,096 to +4,095 revolutions).

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- The range for **radius** is 0 to 2^{31} -1 counts (steps).
- The range for **planePitch** is 0 to 360°.
- The range for **planeYaw** is 0 to 360°.
- The range for **startAngle** is 0 to 360° where angle 0 is along the positive x axis and values increase counterclockwise from the positive x axis in the xy plane.
- The range for **travelAngle** is the full 64-bit range.

Buffered Operations

Buffered operations allow you to create and manage memory buffers on the NI motion controller. You can create buffers that can hold up to 16,364 points, read and write data to and from these buffers, and check the status of a buffer. Buffers can even be stored in ROM for later use.

Onboard buffers allow you to store much more data than the 120 onboard variables. Data can be moved back and forth between buffers and onboard variables by using input and return vectors.

For example, an onboard buffer can be created to hold a series of target positions to move to, one after another. An onboard program is written to read a position from the buffer to an onboard variable, then use that value to load a target position, start a move, and wait for the move to complete. The same process is repeated for each target position until the entire buffer of positions is consumed. This approach allows for a large number of moves to execute independent of the host computer.

Besides single-shot applications where the buffer holds all of the data or is filled with all of the data only once, it is possible to use an onboard buffer in a continuous fashion, allowing for very large amounts of data to be transferred to or from the motion controller. Use the Check Buffer function to determine the number of points that can be safely written to or read from the onboard buffer at any point in time.

The 7350 and 7340 controllers have the following onboard memory:

Controller	RAM	ROM
7340	1 64k sector	2 64k sectors
7350	2 64k sectors	4 64k sectors

You can access a buffer from either RAM or ROM, but you cannot split buffers between the two, and you cannot split buffers between the 64 KB ROM sectors.

Contouring is a special movement mode implemented on top of the buffered operations. Contouring allows you to specify a series of positions *n* milliseconds apart for the motion controller to spline through.

You can use the Buffered Operations Functions to perform buffered highspeed captures and buffered breakpoints. These allow you to achieve faster rates than would otherwise be possible by automatically storing captures positions or automatically loading breakpoints. Refer to <u>High-Speed Capture Overview</u> and <u>Breakpoint Overview</u> for more information.

Note Configuring and clearing buffers is a processor-intensive operation on the NI motion controller that requires the allocation and de-allocation of memory. Configure and clear buffers only when motors are not moving and onboard programs are not running.

Configure Buffer Read Buffer Write Buffer Check Buffer Clear Buffer flex_check_buffer

Device Compatibility

Check Buffer

Usage

status = flex_check_buffer(u8 boardID, u8 buffer, u8 returnVector);

Purpose

Returns information about the current state of the buffer.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
buffer	u8	buffer to check
returnVector	u8	destination for the return data

Parameter Discussion

buffer is the buffer to check. Valid buffer numbers are 1 through 255 (0x01 through 0xFF).

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to Input and Return Vectors for more detailed information. **returnVector** returns data in the following order: **backlog**, **bufferState**, and **pointsDone**.

Using This Function

This function provides you with current information about the state of the buffer. In single iteration situations (**totalPoints** was set to **bufferSize** in the <u>Configure Buffer</u> function), the **bufferState** parameter is most useful in determining that the operation has completed.

For a finer resolution view of the state of the buffer, the backlog parameter indicates the number of points that can safely be written to or read from the buffer without violating the **oldDataStop** condition. Refer to Configure Buffer for more information.

pointsDone is the number of points read from the buffer by the controller (in the case of an output buffer) or written to the buffer by the controller (in the case of an input buffer). **pointsDone** is from the point of view of the controller, rather than the user. In the output case (contouring for example), even after the buffer has been filled by the user but the move has not yet been started, **pointsDone** is still 0, because the controller has not yet read any data from the buffer. After the move has started, and some points have been read from the buffer by the NI motion controller, **pointsDone** is non-zero. In the input case (high-speed capture readback for example), after the controller has filled the buffer with high-speed capture positions once, **pointsDone** is the size of the buffer, even if the user has not yet read any data out of the buffer.

Note This function returns data in the following order: **backlog**, **bufferState**, and **pointsDone**.

As an example, in the case of contouring (output buffer), the backlog parameter aids you in determining when you can write more data to the buffer, while the **pointsDone** parameter tells you how many points have actually been splined together by the controller. **pointsDone** reflects what is happening in the real world where motion occurs.

Example (Output Buffer)

After a 10-point buffer is configured for contouring, the contents are indeterminate and both the read and write heads are at zero.

Write: 0					
Read: 0					

After writing five points to the buffer, with the values 1 through 5, the write head has moved to the sixth position in the buffer, where the next write takes place. The read head is still at zero, because no data has been read from the buffer yet. Likewise, **pointsDone** is still 0.

				Write: 5			
1	2	3	4	5			
Read: 0							

The contouring operation is started by calling the <u>Start Motion</u> function, and the controller starts reading data from the buffer. After reading three points, the read head is at the fourth buffer element, with a value of 3. **pointsDone** is equal to 3.

				Write: 5			
1	2	3	4	5			
		Read: 3					

Upon calling the Check Buffer function, the backlog is 8, meaning that eight points can be safely written to the buffer without overwriting data still remaining to read by the controller. The **pointsDone** parameter is still equal to 3. At this point, another Format call of eight points with values between 6 and 13 results in a write head with a value of 13 at the fourth buffer element. In the meantime, another point has been read from the buffer by the controller in the course of doing the contouring operation, causing the read head to advance by one.

		Write: 13							
11	12	13	4	5	6	7	8	9	10
			Read: 4						

flex_check_buffer_rtn

Device Compatibility

Check Buffer Return

Usage

status = flex_check_buffer_rtn(u8 boardID, u8 buffer, u8 returnVector, u32* backlog, u16* bufferState, u32* pointsDone);

Purpose

Returns information about the current state of the buffer.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
buffer	u8	buffer to check
backlog	u32*	points that can be read from or written to the buffer
bufferState	u16*	current state of the buffer
pointsDone	u32*	number of points the controller has written or read

Parameter Discussion

buffer is the buffer to check. Valid buffer numbers are 1 through 255 (0x01 through 0xFF).

backlog is the number of 32-bit data points that can be safely read from or written to the buffer without violating the **oldDataStop** condition. This value is useful when **oldDataStop** is set to true in the <u>Configure Buffer</u> function.

Error	Number	Description
NIMC_BUFFER_NOT_EXIST	0	The buffer does not exist. This also returns error -70137.
NIMC_BUFFER_READY	1	The buffer is configured but is not being used by any resource.
NIMC_BUFFER_ACTIVE	2	The buffer is configured and is being used by a resource.
NIMC_BUFFER_DONE	3	The buffered operation is complete.
NIMC_BUFFER_OLDDATASTOP	4	The operation stopped on unread data that would be overwritten.

bufferState is the current state of the buffer.

pointsDone gives the total number of points read from the buffer by the controller or written to the buffer by the controller because the operation started.
Using This Function

This function provides you with current information about the state of the buffer. In single iteration situations (**totalPoints** was set to **bufferSize** in the <u>Configure Buffer</u> function), the **bufferState** parameter is most useful in determining that the operation has completed.

For a finer resolution view of the state of the buffer, the backlog parameter indicates the number of points that can safely be written to or read from the buffer without violating the **oldDataStop** condition. Refer to Configure Buffer for more information.

pointsDone is the number of points read from the buffer by the controller (in the case of an output buffer) or written to the buffer by the controller (in the case of an input buffer). **pointsDone** is from the point of view of the controller, rather than the user. In the output case (contouring for example), even after the buffer has been filled by the user but the move has not yet been started, **pointsDone** is still 0, because the controller has not yet read any data from the buffer. After the move has started, and some points have been read from the buffer by the NI motion controller, **pointsDone** is non-zero. In the input case (high-speed capture readback for example), after the controller has filled the buffer with high-speed capture positions once, **pointsDone** is the size of the buffer, even if the user has not yet read any data out of the buffer.

As an example, in the case of contouring (output buffer), the backlog parameter aids you in determining when you can write more data to the buffer, while the **pointsDone** parameter tells you how many points have actually been splined together by the controller. **pointsDone** reflects what is happening in the real world where motion occurs.

Example (Output Buffer)

After a 10-point buffer is configured for contouring, the contents are indeterminate and both the read and write heads are at zero.

Write: 0					
Read: 0					

After writing five points to the buffer, with the values 1 through 5, the write head has moved to the sixth position in the buffer, where the next write takes place. The read head is still at zero, because no data has been read from the buffer yet. Likewise, **pointsDone** is still 0.

				Write: 5			
1	2	3	4	5			
Read: 0							

The contouring operation is started by calling the <u>Start Motion</u> function, and the controller starts reading data from the buffer. After reading three points, the read head is at the fourth buffer element, with a value of 3. **pointsDone** is equal to 3.

				Write: 5			
1	2	3	4	5			
		Read: 3					

Upon calling the Check Buffer function, the backlog is 8, meaning that eight points can be safely written to the buffer without overwriting data still remaining to read by the controller. The **pointsDone** parameter is still equal to 3. At this point, another Format call of eight points with values between 6 and 13 results in a write head with a value of 13 at the fourth buffer element. In the meantime, another point has been read from the buffer by the controller in the course of doing the contouring operation, causing the read head to advance by one.

		Write: 13							
11	12	13	4	5	6	7	8	9	10
			Read: 4						

flex_clear_buffer

Device Compatibility

Clear Buffer

Usage

status = flex_clear_buffer(u8 boardID, u8 buffer);

Purpose

Clears the previously configured buffer and clears any associations between resources and the specified buffer.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
buffer	u8	buffer to clear

Parameter Discussion

buffer is the buffer to clear. Valid buffer numbers are 1 through 255 (0x01 through 0xFF).

Using This Function

After a buffered operation is completed, it is important to clear the buffer by calling the Clear Buffer function. Otherwise, the memory allocated for the buffer in the onboard RAM is not available for other uses, such as the allocation of another buffer or the storage of an onboard program.

A buffer cannot be cleared while the buffer is in use. An NIMC_bufferInUseError is generated in such a case.

Note Configuring and clearing buffers is a processor-intensive operation on the motion controller that requires the allocation and deallocation of memory. You must configure and clear buffers only when motors are not moving and onboard programs are not running. For example, if you wish to execute three simultaneous contouring operations on axis 1, axis 2, and vector space 1 (with axes 3 and 4), you must first configure all three buffers before starting any of the operations. You can start the contour operations independently, and at different times, but must wait until all operations are complete before clearing any of the buffers.

Advanced Uses

This function only clears a buffer from RAM. If you have saved a buffer to ROM using the Object Memory Management function, you must also delete the buffer from ROM using the same function. If you call the Clear Buffer function and specify a buffer that resides in ROM, the association between the resource and the buffer is cleared, but the buffer remains in ROM.



If multiple resources refer to the same buffer, all associations from all resources to the specified buffer are cleared. Refer to the *Advanced Uses* section in the <u>Configure Buffer</u> function description for more information.

This function may take longer than 62 ms to process, so it is not guaranteed to be compatible with real-time execution.

flex_configure_buffer

Device Compatibility

Configure Buffer

Usage

status = flex_configure_buffer(u8 boardID, u8 buffer, u8 resource, u16 bufferType, i32 buffersize, u32 totalPoints, u16 oldDataStop, f64 requestedInterval, f64* actualInterval);



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Purpose

Configures a buffer for use in buffered operations.

Parameters

Name	Туре	Description			
boardID	u8	assigned by Measurement & Automation Explorer (MAX)			
buffer	u8	buffer to configure			
resource	u8	resource to associate with buffer			
bufferType	u16	type of data in buffer			
buffersize	i32	size of buffer in 32-bit words			
totalPoints	u32	total number of points the buffer is used for			
oldDataStop	u16	if to stop on old data			
requestedInterval	f64	requested interval in milliseconds between successive points in buffer			
actualInterval	f64*	actual interval in milliseconds between successive points in buffer			

Parameter Discussion

buffer is the buffer to configure. Valid **buffer** numbers are 1 through 255 (0x01 through 0xFF).

resource is the axis or vector space to associate with the specified buffer.

bufferType specifies the type of data in the buffer.

bufferType Constant	bufferType Value	Direction
NIMC_GENERAL_PURPOSE_INPUT	0	Input
NIMC_GENERAL_PURPOSE_OUTPUT	1	Output
NIMC_POSITION_DATA	2	Output
NIMC_BREAKPOINT_DATA	3	Output
NIMC_HIGH- SPEED_CAPTURE_READBACK	4	Input
NIMC_CAMMING_POSITION	5	Output

Following is detailed information about each of the buffer types:

NIMC_GENERAL_PURPOSE_INPUTNIMC_GENERAL_PURPOSE_OU^r

The general-purpose input and output buffer types allow you to allocate a buffer of memory on the NI motion controller for your own use. You can freely read from and write to the buffer from the host computer or from onboard programs.

The direction of a buffer is from the point of view of the host computer. An output buffer is typically written to by a program on the host computer and read from by an onboard program running on the NI motion controller. An input buffer is typically read from by a program on the host computer and written to by an onboard program running on the NI motion controller, as shown in the following figure.



For example, you can configure a 300-point output buffer and fill it with position, velocity, and acceleration values for a sequence of 100 moves. You also can write an onboard program that reads values from the buffer and places them in onboard variables. Then, a sequence of Load Target Position, Load Velocity and Load Acceleration/Deceleration functions can be executed, using the onboard variables as the source of their data.

As another example, you can configure an input buffer to hold analog input values. If you want to record the value of an analog input every time a particular digital input changes state, you can write an onboard program to wait for the digital input to change state, at which time you read the analog input and write the value to the buffer with the <u>Write</u> <u>Buffer</u> function. This action can be done in a loop in your onboard program. When the operation is complete, or even while the buffer is being filled with analog input values, a program running on the host computer can read the values from the buffer with the <u>Read Buffer</u> function. For these two buffer types, the **resource** and **requestedInterval** parameters are ignored.

NIMC_POSITION_DATA

This buffer type allows you to specify a sequence of one, two, or threedimensional position data points to the NI motion controller. To specify two or three-dimensional position data, the resource parameter must be a vector space. The data is interpreted as absolute or relative contouring data depending on the value of the operation mode set through the <u>Set</u> <u>Operation Mode</u> function.

After you configure an axis or vector space for absolute or relative contouring in the Set Operation Mode function, and write the contouring data to the buffer with the <u>Write Buffer</u> function, you start the contouring operation by calling the <u>Start Motion</u> function. The NI motion controller processes the contouring data by splining between successive points. The **requestedInterval** parameter indicates the time between points in milliseconds. You are responsible for providing the points in a way that

results in the specified acceleration, velocity and deceleration.

Note If you are running a contoured move with a stepper axis, it is possible that the contour velocity exceeds the maximum step rate of the controller. In this case, the controller kills the axis and sets the following error status to true. Refer to the *Specifications* section of your controller user manual for the maximum step rate of your controller.

It is important to realize that all contoured moves are relative, meaning motion starts from the position of the axis or axes at the time the contouring move is started. Absolute contouring and relative contouring refer to the way the contouring data is interpreted. Absolute contouring data indicates the absolute position at each time interval, where the starting position is considered to be 0. Relative contouring data indicates incremental position differences between contouring points. The following two examples result in the same move taking place.

Absolute Contouring 1 3 6 10 14 18 22 25 27 28 Relative Contouring

1 2 3 4 4 4 4 3 2

If the axis was at position 0 to start with, it would end up at position 28 in both cases. If the axis was at position 10 to start with, it would end up at position 38 in both cases.

1

Choose the appropriate mode depending upon the kind of data you have or wish to provide. One benefit of the relative contouring mode is that you can specify multiple identical moves in the same direction with the same data. For example, the two-dimensional contour move shown in the following figure would be possible with 5 iterations of a relative contouring buffer, where **totalPoints** = five times the **bufferSize**, but would not be possible with absolute contouring data unless you provided five times as much data.

NIMC_BREAKPOINT_DATA

This buffer type allows you to specify a sequence of one-dimensional breakpoint position data points to send to the NI motion controller. The data is interpreted as absolute or relative depending on the value of the enable mode set through the <u>Configure Breakpoint</u> function.

After you configure an axis to include breakpoints and write the breakpoint data to the buffer with the Write Buffer function, you add the breakpoint by calling Configure Breakpoint.

NIMC_HIGH-SPEED_CAPTURE_READBACK

This buffer type allows you to specify a sequence of one-dimensional high-speed capture data points to send to the NI motion controller. This buffer is treated similarly to a general purpose output buffer, and you can freely read from the buffer from the host computer or from onboard programs. After you configure a buffer to contain high-speed capture data points and configure an axis to read the high-speed capture points from the buffer with the <u>Configure High-Speed Capture</u> function, you read the high-speed capture data by calling Read Buffer.

NIMC_CAMMING_POSITION

This buffer type allows you to specify a list of slave axis positions that will be used to calculate the camming profile. After you configure and write the slave axis position data, you can enable camming for a <u>single axis</u> or for <u>multiple axes</u>.

bufferSize is the size of the buffer in 32-bit data. If you configure a buffer for a three axis vector space with 1,000 3-dimensional points, the size of the buffer would be 3,000. For more information, refer to the <u>NI 73xx</u> <u>Controller Considerations</u> section of this function description.

totalPoints specifies the number of points that must be used or filled before the operation is complete. In most cases a value of **bufferSize** is appropriate. Valid values for **totalPoints** range from 1 to 2³²–1, inclusive. If **totalPoints** is greater than **bufferSize**, then read and write operations automatically wrap around to the beginning of the buffer.



Note If you set **totalPoints** to 2³²-1, the buffer will run until the move is stopped, or the last counter data is written. Refer to <u>Write</u>

Buffer for information about writing data to a buffer.

oldDataStop indicates to the motion controller if to stop reading from the buffer or writing to the buffer when old data is encountered. This is only relevant when **totalPoints** is greater than bufferSize. In the output case, such as contouring, when **oldDataStop** is true, and **totalPoints** is greater than **bufferSize**, you must write new data to the buffer before the motion controller finishes the buffer and starts over at the beginning. Likewise, for an input buffer, if the controller (or an onboard program running on the controller) is writing data to the buffer, such as buffered high-speed capture data, you must read data out of the buffer before the controller wraps around and writes new data on top of the old data.

If you set **oldDataStop** to false, the motion controller does not enforce the old data stop condition, and simply writes data to or reads data from the buffer as needed.

Range: True (1) or False (0)

requestedInterval specifies the requested time in milliseconds between contouring data points. **interval** is valid for only a **bufferType** of NIMC_POSITION_DATA, NIMC_BREAKPOINT_DATA, or NIMC_HIGH-SPEED_CAPTURE_READBACK.

The controller uses the closest legal value that is greater than or equal to the requested value for interval. Legal values for interval must be even multiples of the PID rate. See **actualInterval** for more information.

When used with NIMC_BREAKPOINT_DATA or NIMC_HIGH-SPEED_CAPTURE_READBACK, **requestedInterval** indicates how often, in milliseconds, the onboard data is updated from the hardware, and the value is a whole number.

actualInterval is the interval of time in milliseconds that the controller uses between successive points in the buffer. **actualInterval** may or may not be the same as the **requestedInterval**. For example, if the **requestedInterval** is physically impossible, then a different value is used. This value is represented as **actualInterval**.

For NIMC_BREAKPOINT_DATA or NIMC_HIGH-SPEED_CAPTURE_READBACK,

actualInterval = [requestedInterval]

Using This Function

After a buffer has been configured, you can write to the buffer with the <u>Write Buffer</u> function and read from the buffer with the <u>Read Buffer</u> function.

A contour operation that uses a buffer is initiated when the <u>Start Motion</u> function is called. You check the state of the buffer with the <u>Check Buffer</u> function to determine how many points have been read from the buffer or written to the buffer by the NI motion controller.

After a buffered operation is completed, it is important to clear the buffer by calling the <u>Clear Buffer</u> function. Otherwise, the memory allocated for the buffer in the onboard RAM is not available for other uses, such as the allocation of another buffer.

A buffer cannot be reconfigured or cleared while the buffer is in use. An NIMC_bufferInUseError is generated in such a case.

A buffer can only be used by one resource at a time. Multiple resources can be associated with a single buffer, but only one resource can use that buffer at any given time. Likewise, a single resource can make use of many buffers, but for buffers of the same type, only one of them can be associated with that resource at a given time. Refer to the <u>NI 73xx</u> <u>Controller Considerations</u> section of this function description for more information.



Note Configuring and clearing buffers is a processor-intensive operation on the NI motion controller that requires the allocation and de-allocation of memory. Configure and clear buffers only when motors are stopped and onboard programs are not running.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

• Using input vectors and return vectors with the Read Buffer and Write Buffer functions, you can write to and read from the buffer with data stored in onboard variables. You must limit the number of points to read or write, because there are only 120 onboard variables. Data is read from or written to onboard variables sequentially starting at the variable specified by the **inputVector** or **returnVector** in the Write Buffer function or Read Buffer function.

Specify a **bufferSize** of 0 to associate a resource (like an axis or vector space) to an existing buffer, without affecting the size of the buffer or the data contained in the buffer. The following examples illustrate the use of this technique:

- A single resource (e.g. axis or vector space) can use buffer 1 for a contouring operation, and then be switched over to buffer 2 for a subsequent operation. You can set up the two buffers ahead of time. When the first operation is complete, you can make an additional call to this function with a bufferSize of 0, the same value for resource, and the new buffer number for buffer.
- If you want to reuse a buffer with another resource, call this function with the same buffer number, a **bufferSize** of 0, and a new value for resource. Both resources can still use the single buffer, but not at the same time.

A buffer size of 0 allows you to associate a resource with an existing buffer, without changing the contents of the existing buffer. Also, a buffer size of 0 can be used to leave the current buffer alone, but change the value of another parameter, such as **totalPoints**. If you are reconfiguring a buffer with a **bufferSize** of 0, the read mark is set to 0. If the buffer is an output buffer, the write mark is set to **bufferSize**.

If the buffer is an input buffer, the write mark is set to 0.

- After a contouring operation, the same buffer of points can be contoured again, with a different time interval between points, or number of **totalPoints**. Pass in a 0 for buffer size and change the other parameters as appropriate.
- When reconfiguring an existing output buffer (e.g. contouring buffer) by passing a **bufferSize** of 0, the buffer is assumed to be full of data. If you want to change its contents, do so before reconfiguring the buffer.
- When a buffer has been saved to ROM using the Object Memory Management function, and the association between the buffer and the resource (for example, axis or vector space) has been lost (most likely through a power cycle or a call to Clear Buffer), the association between the resource and the buffer needs to be reestablished before a buffered operation can be performed with the buffer. Because the buffer already exists in ROM, specifying a bufferSize of 0 causes the NI motion controller to associate the specified resource with the specified buffer without allocating any new memory for the buffer. If the buffer does not exist, an NIMC_objectReferenceError is generated.
- The maximum buffer size is 16,364 data points on a 7340 and 7350 controller. Besides the data stored in each buffer, some space is used internally for bookkeeping and state information. Refer to <u>Buffered Operations</u> for information about the total amount RAM and ROM on each controller.
- When a buffer is configured, memory for the buffer is allocated in the controller's onboard RAM. The buffer persists until the buffer is cleared with the Clear Buffer function, or the controller is reset, usually through a power cycle. You can save the buffer to ROM with the <u>Object Memory Management</u> function for later use, even after the power has been cycled. When using a buffer stored in ROM, you cannot write to the buffer.
- The NIMC_HIGH-SPEED_CAPTURE_READBACK attribute is not supported for 7340 controllers.
- The NIMC_CAMMING_POSITION attribute is not supported for 73*xx* controllers.
- For the NIMC_POSITION_DATA attribute, use the following

equation to calculate the actual interval, rounded to the next microsecond:

 $a = \frac{p}{1,000} \times \left[\frac{r \times 1,000}{p} \right]$

where r = **requestedInterval** in milliseconds (ms) a = actual interval in milliseconds (ms) p = PID rate in microseconds (μ s)

For example, given a PID rate of 250 μ s, a time interval of 11.2 ms between points is physically impossible, but a **requestedInterval** of 11.25 ms can be accomplished. So a call to Configure Buffer with interval set to 11.2 configures the buffer for 11.25 ms, because that is the shortest possible interval that is greater than or equal to the requested interval. The actual value being used, 11.25 in this case, is returned as an output parameter. The following table shows the minimum and maximum intervals for all valid PID rates.

PID rate	Minimum Interval	Maximum Interval
62.5 µs	10.0 ms	11.3125 ms
125 µs	10.0 ms	22.625 ms
187.5 µs	10.125 ms	33.9375 ms
250 µs	10.0 ms	45.25 ms
312.5 µs	10.0 ms	56.5625 ms
375 µs	10.125 ms	67.875 ms
437.5 μs	10.0625 ms	79.1875 ms
500 µs	10.0 ms	90.5 ms

• You can change the PID rate in MAX or with the Enable Axis function.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- The NI SoftMotion Controller supports only the NIMC_POSITION_DATA and NIMC_CAMMING_POSITION attributes.
- The NI SoftMotion Controller for Copley Controls' CANopen devices supports minimum contour intervals as follows:

- 10 ms for a motion control system that uses 1 to 8 axes
- 20 ms for a motion control system that uses 9 to 15 axes
- 30 ms for a motion control system that uses 16 to 23 axes
- 40 ms for a motion control system that uses 24 to 30 axes
- If you set **oldDataStop** to false and **totalPoints** is greater than **bufferSize**, you must set **totalPoints** to a multiple of **bufferSize**. Otherwise, an error is returned.

For example, to repeat a contour of 2,000 points 100 times, set **oldDataStop** to false, **totalPoints** to 200,000, and **bufferSize** to 2,000.

• The size of the buffer is limited by the system resources.

flex_read_buffer

Device Compatibility

Read Buffer

Usage

status = flex_read_buffer(u8 boardID, u8 buffer, u32 numberOfPoints, u8
returnVector);

Purpose

Reads data from a previously configured buffer.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
buffer	u8	buffer to read
numberOfPoints	u32	number of data points to read from the buffer
returnVector	u8	destination for the return data

Parameter Discussion

buffer is the buffer to read from. Valid **buffer** numbers are 1 through 255 (0x01 through 0xFF). **numberOfPoints** is the number of 32-bit data points to read from the buffer.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.

When returning data to a **returnVector**, you must limit the **numberOfPoints** to a number that fits in a sequence of onboard variables starting with the variable specified by the **returnVector**.

Using This Function

An input buffer can be written to after it has been configured, usually by an onboard program. After data is available to read, you read the data from the buffer with this function.

When the buffer is configured for a single iteration, you can wait until the buffer is full by checking the status of the buffer with the <u>Check Buffer</u> function, and then read the entire contents of the buffer. You also can read the smaller chunks of data from the buffer while the operation is in progress. The number of points available to read can be determined with the backlog parameter returned by the Check Buffer function.

When using the buffer as a circular buffer (**totalPoints** > **bufferSize**), with **oldDataStop** turned on, you need to read data from the buffer faster than the buffer is being filled with new data. You can determine the number of points you can safely read from the buffer by calling the Check Buffer function. The backlog parameter indicates how many points you can safely read from the buffer.

If you try to read data that has not yet been written to the buffer by the controller, the data is not read, and the controller generates an NIMC_oldDataStopError. However, if you configured the buffer with **oldDataStop** set to false, you can read data freely from the buffer, and the controller does not enforce the **oldDataStop** condition.

Under no circumstances can data be read from an output buffer (for example, contouring) after the specified number of points (**totalPoints**) has already been read from the buffer. An NIMC_illegalBufferOperation error is generated in this case.

flex_read_buffer_rtn

Device Compatibility

Read Buffer Return

Usage

status = flex_read_buffer_rtn(u8 boardID, u8 buffer, u32 numberOfPoints, i32*
data);

Purpose

Reads data from a previously configured buffer.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
buffer	u8	buffer to read
numberOfPoints	u32	number of data points to read from the buffer
data	i32*	destination array for data read from the buffer

Parameter Discussion

buffer is the buffer to read from. Valid **buffer** numbers are 1 through 255 (0x01 through 0xFF). **numberOfPoints** is the number of 32-bit data points to read from the buffer.

data is the destination array for data read from the buffer.

Using This Function

An input buffer can be written to after it has been configured, usually by an onboard program. After data is available to read, you read the data from the buffer with this function.

When the buffer is configured for a single iteration, you can wait until the buffer is full by checking the status of the buffer with the <u>Check Buffer</u> function, and then read the entire contents of the buffer. You also can read the smaller chunks of data from the buffer while the operation is in progress. The number of points available to read can be determined with the backlog parameter returned by the Check Buffer function.

When using the buffer as a circular buffer (**totalPoints** > **bufferSize**), with **oldDataStop** turned on, you need to read data from the buffer faster than the buffer is being filled with new data. You can determine the number of points you can safely read from the buffer by calling the Check Buffer function. The backlog parameter indicates how many points you can safely read from the buffer.

If you try to read data that has not yet been written to the buffer by the controller, the data is not read, and the controller generates an NIMC_oldDataStopError. However, if you configured the buffer with **oldDataStop** set to false, you can read data freely from the buffer, and the controller does not enforce the **oldDataStop** condition.

Under no circumstances can data be read from an output buffer (for example, contouring) after the specified number of points (**totalPoints**) has already been read from the buffer. An NIMC_illegalBufferOperation error is generated in this case.

flex_write_buffer

Device Compatibility

Write Buffer

Usage

status = flex_write_buffer(u8 boardID, u8 buffer, u32 numberOfPoints, u16
regenerationMode, i32* data, u8 inputVector);
Purpose

Writes data to a previously configured buffer.

Parameters

Name	Туре	Description	
boardID	u8	assigned by Measurement & Automation Explorer (MAX)	
buffer	u8	buffer to write to	
numberOfPoints	u32	number of data points to write to the buffer	
regenerationMode	u16	for indicating end of data (last write)	
data	i32*	data to write to buffer	
inputVector	u8	source of the data for this function	

Parameter Discussion

buffer is the buffer to write to. Valid **buffer** numbers are 1 through 255 (0x01 through 0xFF). **numberOfPoints** is the number of 32-bit data points to write to the buffer.

regenerationMode allows you to specify that this write is the last write for the buffered operation. This parameter is normally set to 0 (No Change), but can be set to 1 (Last Write) when the total number of points you want to output is not known ahead of time.

When all of the data in the buffer is consumed, the operation then completes with a normal status, even if it completed by encountering older data.

data is the array of data to write to the buffer.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

After a buffer has been configured for an output operation such as contouring, write data to the buffer before starting the operation. When all of the data fits in the buffer (**totalPoints** is less than or equal to **bufferSize**), you need only fill the buffer one time with the appropriate data.

When using the buffer as a circular buffer (**totalPoints** > **bufferSize**), with **oldDataStop** turned on, you need to write fresh data to the buffer faster than the NI motion controller consumes the data. After filling the buffer once, and starting the buffered operation, you can determine the number of points you can safely write to the buffer by calling the <u>Check</u> <u>Buffer</u> function. The backlog parameter indicates how many points you can safely write to the buffer data that has yet to be consumed by the controller.

If you try to write new data on top of old data (data that has not been consumed by the controller), the data is not written, and the controller generates an NIMC_oldDataStopError. However, if you configured the buffer with **oldDataStop** set to false, you can write data freely to the buffer, and the controller does not enforce the **oldDataStop** condition.

Basic Uses

Suppose you have a contour move of 25,107 points, and configure a buffer with **bufferSize** of 1,000 points and **totalPoints** of 25,107. You set **oldDataStop** in the <u>Configure Buffer</u> function to True, and use the Check Buffer and Write Buffer functions as you write data to the buffer in small chunks while the data is being consumed by the NI motion controller. When you write the last points to the controller, you can still set **regenerationMode** to 0 (no change) and the NI motion controller knows you are done writing data to the buffer because you have written 25,107 points in total. When the controller is unable to read any more because the read mark has caught up to the write mark, the controller checks **regenerationMode**, recognizes this stop is intentional because the total points written to the buffer is equal to **totalPoints**, and completes the operation without generating an **oldDataStop** error.

Advanced Uses

To write data to the buffer from onboard variables, you must use an **inputVector** other than 0xFF. The variable you specify for the

inputVector indicates the starting point for the data for this function, including the **numberOfPoints** parameter and the **regenerationMode** parameter, followed by the data to write to the buffer. To write 10 data points to the buffer, you need to use 12 variables. Under no circumstances can data be written to an input buffer after the specified number of points has already been written to the buffer. A NIMC_illegalBufferOperation error is generated in this case. The number of points is generally defined as the number of iterations times the size of the buffer.

The **regenerationMode** parameter allows you to gracefully complete a buffered output operation without generating an NIMC_olddatastopError, even though the operation completes as the read mark catches up to the write mark and the operation terminates because there's no more data available for the controller to consume.

For example, assume you have a contour move of less than 16,000 points, so you configure a buffer for 1,000 points and 16,000 total points. You set the **oldDataStop** parameter to True with Configure Buffer, and use the Check Buffer and Write Buffer functions to write the points to the buffer as the data is being consumed by the NI motion controller. During runtime you are able to determine the exact number of points needed for the contoured operation, and you discover you only need to write *x* more points to the buffer instead of the originally configured 16,000. To force the controller to stop after reading those *x* remaining points and prevent **oldDataStop** error conditions, set the **regenerationMode** parameter to 1 (Last Write) when you write the last set of points to the buffer. When the controller is unable to read any more data from the buffer because the read mark has caught up to the write mark, the controller recognizes that this was intentional because the Last Write condition was set and does not generate an NIMC_olddatastopError.

Camming

You can use Camming functions to set up and control master-slave camming on the NI motion controller. It includes functions to configure a camming master, load a camming parameter, enable camming, and so on.

Camming is an advanced feature of NI-Motion and is used in applications where either the master axis is not under control of the NI motion controller or when extremely tight synchronization between multiple axes is required.



Note If you are gearing a stepper motor and the gear ratio is such that the slave axis exceeds the maximum stepper velocity when the master does not, the slave axis does not move, and may trip on following error. To prevent this, ensure the gear ratio does not permit the slave axis to exceed the maximum stepper velocity. Refer to your motion controller documentation for information about maximum stepper velocities.

Configure Camming Master

Enable Camming

Enable Camming Single Axis

Load Camming Parameter

flex_configure_camming_master

Device Compatibility

Configure Camming Master

Usage

i32 status = flex_configure_camming_master(i32 boardID, i32 axisID, i32 masterResource, f64 masterCycle);

Purpose

Configures the master device for camming operation.

Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description	
boardID	i32	assigned by Measurement & Automation Explorer (MAX)	
axisID	i32	slave axis to control	
masterResource	i32	axis, encoder or ADC channel to use as the master	
masterCycle	f64	repeat period for the master device	

Parameter Discussion

axisID is the slave axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC badResourceIDOrAxisError). Refer to Axes for axis resource

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

masterResource is the axis (NIMC_AXIS1 through NIMC_AXIS30), trajectory generator, encoder (NIMC_ENCODER1 through NIMC_ENCODER30), or ADC channel (NIMC_ADC1 through NIMC_ADC30) to use as the master for this slave axis. A zero (0) value means no master is assigned (default). On motion controllers that support fewer than thirty axes, configuring non-existent resources returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u>, <u>Encoders</u>, and <u>ADC Channels</u>, respectively for axis, encoder, and ADC channel resource IDs.

Note axisID and masterResource cannot be the same, because an axis cannot be its own master.

masterCycle is the repeat period, in user units, for the master device.

Using This Function

Use this function to assign a master axis, encoder, or ADC channel to the slave axis selected for the camming application. Any number of slave axes can have the same master, but each slave axis must have only one master.

Before starting the camming operation, you must configure the master device. Using an axis as the master device allows the slave axis to follow the trajectory generation output of the master axis. This functionality is useful when the master axis is an open-loop system or when using a feedback device (encoder or ADC) as the master, which causes high following error.

You also can use a feedback device, such as an encoder or ADC, as the master device so that any movement made by the master is immediately propagated to the slave.

The master device position is interpreted, or modularized, within the master cycle, which allows the camming operation to repeat continuously as the master device moves. Along with slave positions, the master cycle is used to calculate the camming ratio at any time.

For example, if the master cycle is 5000 and the current master position is 12,500, the master position inside the master cycle is

position = ((12,500 modulus 5,000) = 2,500)

Using the previous master cycle but with a master position of -4000, the master position inside the cycle is

position = ((-4,000 modulus 5,000) = 1,000)

You also can use the <u>Load Camming Parameter</u> function to change the camming cycle at any time.

Refer to the <u>Camming</u> section of the *NI-Motion Help* for more information about camming operations.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

• The NI SoftMotion Controller does not support using analog feedback as the camming master.

flex_enable_camming

Device Compatibility

Enable Camming

Usage

i32 status = flex_enable_camming(i32 boardID, u32 arraySize, NIMC_CAMMING_ENABLE_DATA* dataArray);

Purpose

Enables camming operation for multiple axes.

Parameters

Name	Туре	Description
boardID	i32	assigned by Measurement & Automation Explorer (MAX)
arraySize	u32	number of axes to enable camming on
dataArray	NIMC_CAMMING_ENABLE_DATA*	array of structures that indicates the axes that are enabled for camming

Parameter Discussion

arraySize is the number of axes, or elements in the **dataArray** parameter, to enable/disable camming on.

dataArray contains the information to enable/disable for the axes:

- Set **axisIndex** to the axis ID to start or stop camming.
- Set **enable** to True (1) to enable camming or False (0) to disable camming.
- Set **position** to the master position, in user units, where the enable/disable is going to take effect.

The **dataArray** structure is defined as follows:

struct{
 i32 axisIndex;
 u8 enable;
 f64 position;
} NIMC_CAMMING_ENABLE_DATA;

The valid range of values for the position element of **dataArray** is -1 or 0 through (master cycle -1).

Using This Function

Use this function to simultaneously enable or disable a camming operation on multiple axes. The default value for the position element of the **Enable Data** array is -1, which starts camming immediately. To start camming at a specific master position, set this parameter to a value between 0 and (master cycle -1).

Refer to the <u>Camming</u> section of the *NI-Motion Help* for more information about camming operations.

flex_enable_camming_single_axis

Device Compatibility

Enable Camming Single Axis

Usage

i32 status = flex_enable_camming_single_axis(i32 boardID, i32 axisID, u16 enable, f64 position);

Purpose

Enables camming operation for the current axis.

Parameters

Name	Туре	Description
boardID	i32	assigned by Measurement & Automation Explorer (MAX)
axisID	i32	axis to configure
enable	u16	enable/disable camming
position	f64	position to begin camming at

Parameter Discussion

axisID is the axis to enable/disable camming for. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

enable allows you to enable/disable camming on the specified **axis**. Set **enable** to True (1) to enable camming or False (0) to disable camming.

position is the position, in user units, to start or stop camming at on the specified **axis**. The valid range is -1 or 0 through (master cycle -1).

Using This Function

Use this function to start or stop a camming operation on an axis. You can start or stop the camming operation immediately by setting position to -1. To configure the camming operation to take place after the master device crosses a specific position inside the master cycle, set position to a value between 0 and (master cycle -1).

Refer to the <u>Camming</u> section of the *NI-Motion Help* for more information about camming operations.

Load Camming Parameter

Usage

i32 status = flex_load_camming_parameter(i32 boardID, i32 axisID, TnimcCammingParameter attribute, NIMC_DATA* data);

Purpose

Loads other camming parameters, such as master axis position offset and slave axis position offset.

Parameters

Name	Туре	Description
boardID	i32	assigned by Measurement & Automation Explorer (MAX)
axisID	i32	axis to configure
attribute	TnimcCammingParameter	attribute to configure
data	NIMC_DATA*	value to load for the specified attribute

Parameter Discussion

axisID is the axis to configure. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

attribute is one of the following camming parameters:

attribute	Number
TnimcCammingParameterMasterCycle	0
TnimcCammingParameterMasterOffset	1
TnimcCammingParameterSlaveOffset	2

data is the value to set for the specified **attribute**. Use the **doubleData** element of the following structure to set the attribute value:

struct{

i32 longData; u8 boolData; f64 doubleData; } NIMC_DATA;

Use one of the following steps to load the appropriate value:

- **TnimcCammingParameterMasterCycle**—Set the **doubleData** element of the NIMC_DATA structure to the master cycle in user units.
- **TnimcCammingParameterMasterOffset**—Set **doubleData** element of the NIMC_DATA structure to the master offset in user units.
- **TnimcCammingParameterSlaveOffset**—Set **doubleData** element of the NIMC_DATA structure to the slave offset in user units.

Using This Function

Use this function to configure various parameters of the camming operation:

• **Master Cycle** defines when the camming operation is going to repeat based on the master position. You can configure the master cycle at the same time you configure the camming master using the <u>Configure Camming Master</u> function.



Note You cannot change the master cycle after camming is enabled.

• **Master Offset** is applied to the master position when camming is enabled. This attribute shifts the position modulus to compensate for the fact that the camming cycle may not start at 0. For example, if you have a master cycle of 4000 and a master offset of 0, the camming operation repeats at ..., -8,000, -4,000, 0, 4,000, 8,000, ... If you change the master offset to 1000, the camming operation repeats at ..., -7,000, -3,000, 1,000, 5,000, 9,000,

The default value for the master offset is 0, and the valid range of values for the master offset is 0 to (Master Cycle -1).

Tip The master offset is applied only when camming is started or enabled.

• **Slave Offset** is applied when the camming cycle begins. This offset is used to adjust the camming profile when the beginning and ending positions for the slave axis are not the same. You can change the slave offset at anytime, but the change takes effect on the next camming cycle.

The default value is 0.

Refer to the <u>Camming</u> section of the *NI-Motion Help* for more information about camming operations.

Gearing

You can use Gearing functions to set up and control master-slave gearing on the NI motion controller. It includes functions to configure a gear master, load a gear ratio, and enable master-slave gearing.

Gearing is an advanced feature of NI-Motion and is used in applications where either the master axis is not under control of the NI motion controller or when extremely tight synchronization between multiple axes is required.



Note If you are gearing a stepper motor and the gear ratio is such that the slave axis exceeds the maximum stepper velocity when the master does not, the slave axis does not move, and may trip on following error. To prevent this, ensure the gear ratio does not permit the slave axis to exceed the maximum stepper velocity. Refer to your motion controller documentation for information about maximum stepper velocities.

<u>Configure Gear Master</u> <u>Load Gear Ratio</u> <u>Enable Gearing</u> <u>Enable Gearing Single Axis</u> flex_config_gear_master

Device Compatibility

Configure Gear Master

Usage

status = flex_config_gear_master(u8 boardID, u8 axis, u8
masterAxisOrEncoderOrADC);

Purpose

Assigns a master axis, encoder, or ADC channel for master-slave gearing.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	slave axis to control
masterAxisOrEncoderOrADC	u8	axis, encoder or ADC channel to use as the master
Parameter Discussion

axis is the slave axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

masterAxisOrEncoderOrADC is the axis (NIMC_AXIS1 through NIMC_AXIS30), trajectory generator, encoder (NIMC_ENCODER1 through NIMC_ENCODER30), or ADC channel (NIMC_ADC1 through NIMC_ADC30) to use as the master for this slave axis. A zero (0) value means no master is assigned (default). On motion controllers that support fewer than thirty axes, configuring non-existent resources returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u>, <u>Encoders</u>, and <u>ADC Channels</u>, respectively for axis, encoder, and ADC channel resource IDs.



Note axis and masterAxisOrEncoderOrADC cannot be the same, because an axis cannot be its own master.

Using This Function

This function assigns a master axis, encoder, or ADC channel to the slave axis selected. Any number of slave axes can have the same master, but each slave axis can have only one master. A slave axis may also serve as the master axis for any number of additional slave axes.

You must call this function prior to enabling master-slave gearing with the <u>Enable Gearing</u> function. Typically, the source for the master position is an independent encoder, an ADC channel, or the feedback resource of an enabled axis. In each case, you assign the resource, not the axis, as the master. The gear master, which can be an axis, encoder, or ADC, must be enabled before you can use this function. Use the <u>Enable Axis</u>, <u>Enable Encoders</u>, or <u>Enable ADCs</u> functions to enable the gear master.

When an axis is assigned as the master, its trajectory generator output (not its feedback position) is used as the master position command. This mode of operation can eliminate the following error skew between the master and slave axes and is especially useful in gantry applications. The master axis can be operating in any mode (including being a slave to another master).

Master-slave functionality of slave axes is in addition to their normal mode of operation. This allows a point-to-point move to be <u>superimposed</u> upon the slave while the slave axis is in motion due to being geared to its master. This functionality is useful for registration and reference offset moves. Refer to the <u>Gearing</u> section of the *NI-Motion Help* for information about using superimposed moves/registration applications. Also, refer to the <u>Rotating Knife</u> section of the *NI-Motion Help* for example code that includes superimposed moves.

Refer to the <u>Load Gear Ratio</u> and <u>Enable Gearing</u> functions for more information about master-slave gearing.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

• The NI SoftMotion Controller does not support using analog feedback as the gear master (analog gearing).

flex_enable_gearing

Device Compatibility

Enable Gearing

Usage

status = flex_enable_gearing(u8 boardID, u16 gearMap);

Purpose

Enables slave axes for master-slave gearing.

Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
gearMap	u16	bitmap of slave axes to enable for gearing

Parameter Discussion

gearMap is the bitmap of slave axes to enable for gearing.

D15	•••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis		Axis	0									
15		10	9	8	7	6	5	4	3	2	1	

D1 through D15:

1 = Gearing enabled

0 = Gearing disabled (default)

Using This Function

This function enables and disables master-slave gearing functionality of slave axes. When gearing is enabled, the positions of the slave axes and their corresponding masters are recorded as their absolute gearing reference. For axes and encoders, the current position is the value recorded. For ADCs, the binary value is recorded. From then on, as long as the gear ratio remains absolute, every incremental change of a master position is multiplied by the corresponding absolute gear ratio and applied to the slave axis. Refer to the Load Gear Ratio function for more information about absolute versus relative gear ratios.

Note Changing the absolute gear ratio causes the slave axes to jump to a new position specified by the new gear ratio. With servo motor systems the slave jumps immediately. With stepper systems, if the new slave position is within 4,000 steps, it jumps to the new position. Otherwise, the slave axis waits until the commanded slave position moves to within 4,000 steps and then jumps to the new position. The slave axis then resumes following the master axis according to the gear ratio.

You must call the <u>Configure Gear Master</u> and Load Gear Ratio functions prior to enabling master-slave gearing. In addition, you must enable and activate the slave axes before enabling gearing. An error is generated if a slave is killed when gearing is enabled. These checks ensure that the slave axis enables in a controlled fashion.

Note If you are running a slave axis move with a stepper axis, it is possible that the slave axis geared velocity exceeds the maximum step rate of the controller. In this case, the controller kills the axis and sets the following error status to true. Refer to the *Specifications* section of your controller user manual for the maximum step rate of your controller.

You can call this function at any time to disable gearing or to re-enable gearing with new absolute gearing reference positions. If gearing is disabled on a moving axis, the axis immediately stops but remains active. If the slave axis was also implementing a <u>superimposed</u> move, the superimposed move decelerates to a stop.

Executing the <u>Stop Motion</u> function on a slave axis stops the axis and automatically disables gearing for that axis. If the master trips on following error, the slave halt stops and disables gearing. <u>Find Reference</u> cannot be executed on slave axes with gearing enabled. An error is generated and the find sequence does not start.

 $\overline{\mathbb{N}}$

Note National Instruments recommends disabling gearing before resetting the position of the master axis or the slave axis. When the position of either the master axis or the slave axis is reset, the reset offset is treated as gearing offset, and the slave axis jumps to the new gear position in one PID period.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

To enable gearing on axes sixteen through thirty, use the <u>Enable Gearing</u> <u>Single Axis</u> function. flex_enable_gearing_single_axis

Device Compatibility

Enable Gearing Single Axis

Usage

status = flex_enable_gearing_single_axis(u8 boardID, u8 axis, u16 enable);

Purpose

Enables a slave axis for master-slave gearing.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to enable or disable
enable	u16	enable/disable value

Parameter Discussion

axis is the axis to enable or disable. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

enable indicates if the function enables or disables the slave axis for gearing.

1 = Gearing enabled

0 = Gearing disabled

Using This Function

This function is similar to the <u>Enable Gearing</u> function, but allows you to enable or disable gearing on a single axis without affecting the other axes.

The Enable Gearing Single Axis function enables and disables masterslave gearing functionality of a slave axis. When gearing is enabled, the position of the slave axis and its corresponding master is recorded as its absolute gearing reference. From then on, as long as the gear ratio remains absolute, every incremental change of a master position is multiplied by the corresponding absolute gear ratio and applied to the slave axis. Refer to the Load Gear Ratio function for more information about absolute versus relative gear ratios.

Note Changing the absolute gear ratio causes the slave axes to jump to a new position specified by the new gear ratio. With servo motor systems the slave jumps immediately. With stepper systems, if the new slave position is within 4,000 steps, it jumps to the new position. Otherwise, the slave axis waits until the commanded slave position moves to within 4,000 steps and then jumps to the new position. The slave axis then resumes following the master axis according to the gear ratio.

You must call the <u>Configure Gear Master</u> and Load Gear Ratio functions prior to enabling master-slave gearing. In addition, you must enable and activate the slave axes before enabling gearing. An error is generated if a slave is killed when gearing is enabled. These checks ensure that the slave axis enables in a controlled fashion.

 $\overline{\mathbb{N}}$

Note If you are running a slave axis move with a stepper axis, it is possible that the slave axis geared velocity exceeds the maximum step rate of the controller. In this case, the controller kills the axis and sets the following error status to true. Refer to the

Specifications section of your controller user manual for the maximum step rate of your controller.

You can call this function at any time to disable gearing or to re-enable gearing with new absolute gearing reference positions. If gearing is disabled on a moving axis, the axis immediately stops but remains active. If the slave axis was also implementing a <u>superimposed</u> move, the superimposed move decelerates to a stop.

Executing the <u>Stop Motion</u> function on a slave axis stops the axis and automatically disables gearing for that axis. If the master trips on following error, the slave halt stops and disables gearing. <u>Find Reference</u> cannot be executed on slave axes with gearing enabled. An error is generated and the find sequence does not start.



Note National Instruments recommends disabling gearing before resetting the position of the master axis or the slave axis. When the position of either the master axis or the slave axis is reset, the reset offset is treated as gearing offset, and the slave axis jumps to the new gear position in one PID period.

flex_load_gear_ratio

Device Compatibility

Load Gear Ratio

Usage

status = flex_load_gear_ratio(u8 boardID, u8 axis, u16 absoluteOrRelative, i16 ratioNumerator, u16 ratioDenominator, u8 inputVector);

Purpose

Loads the gear ratio for master-slave gearing.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	slave axis to control
absoluteOrRelative	u16	selects absolute or relative gearing between master and slave
ratioNumerator	i16	gear ratio numerator of the slave relative to the master
ratioDenominator	u16	gear ratio denominator of the slave relative to the master
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the slave axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

absoluteOrRelative selects absolute (NIMC_ABSOLUTE_GEARING) or relative (NIMC_RELATIVE_GEARING) gearing between the master and slave.

ratioNumerator is gear ratio numerator of the slave relative to the master. The numerator is a signed value between –32,768 to +32,767 to allow for both positive and negative gearing.

ratioDenominator is the gear ratio denominator of the slave relative to the master. The denominator must be between 1 to 32,767.

inputVector contains the type, numerator, denominator parameters that are the source of data for this function.

Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return</u> <u>Vectors</u> for more detailed information.

Using This Function

This function loads the gear ratio of the slave axis relative to its master and selects if this ratio is absolute or relative. The ratio is loaded as a numerator and denominator because it is a natural format for a ratio (numerator: denominator) and it allows a broad range of ratios, from 1:32,767 to 32,767:1. The ratio is always specified as slave relative to master (slave:master).

When you execute the Enable Gearing function, the positions of the slave and its master are recorded as their absolute gearing reference. From then on, as long as the gear ratio remains absolute, every incremental change of the master position is multiplied by the absolute gear ratio and applied to the slave axis.

If a relative gear ratio is selected and loaded after gearing is enabled, the position of the master is recorded as its relative reference point and every incremental change from this reference point is multiplied by the relative gear ratio and applied to the slave axis or axis.

 $\overline{\mathbb{N}}$

Note While changing an absolute gear ratio on the fly is allowed, you must be careful because the slave axis jumps with full torque to the position defined by the new ratio even when the master position has not changed. Similarly, resetting the position on a master axis causes its slave axes to jump to a new position relative to the new position of the master axis.

The Load Gear Ratio function must be called prior to enabling masterslave gearing with the Enable Gearing function. Often the positions of the master and slave are reset to zero or some known position prior to enabling gearing, though this is not always required. The execution of the Enable Gearing function stores both positions as offsets and gears them from that point onward.

Relative gearing changes the position of the slave axis only when the

master moves. It uses the length of the current move of the master axis to calculate how far to move the slave axis. At the end of the move, if the slave axis is not at the exact position, the position error is discarded. This may happen when the true target for the slave axis is a fractional position, which is impossible to actually reach. The position error can add up over time to become significant. If it become a problem, call this function with absolute gearing selected to correct for the accumulated error. The controller calculates where the slave is supposed to be relative to the offset of the master axis beginning from the position where gearing was enabled. With gearing ratios less than 1, this problem does not appear.

Master-slave functionality of slave axes is in addition to their normal mode of operation. This allows a point-to-point move to be <u>superimposed</u> upon the slave while the slave axis is in motion due to being geared to its master. This functionality is useful for registration and reference offset moves. Refer to the <u>Gearing</u> section of the *NI-Motion Help* for information about using superimposed moves/registration applications. Also, refer to the <u>Rotating Knife</u> section of the *NI-Motion Help* example code that includes superimposed moves.

Refer to the <u>Configure Gear Master</u> and Enable Gearing functions for more information about master-slave gearing.

Example

To load a slave to master gear absolute gear ratio of 3:2, call the Load Gear Ratio function with **absoluteOrRelative** =

NIMC_ABSOLUTE_GEARING (absolute), **ratioNumerator** = 3 and **ratioDenominator** = 2. For two axes with identical resolution, setting a gear ratio of 3:2 results in the slave axis rotating three revolutions for every two revolutions of the master.

Advanced Trajectory

Advanced Trajectory functions are useful in special applications and showcase some of NI-Motion's power and flexibility.

You can use Advanced Trajectory functions to acquire time-sampled position and velocity data into a large onboard buffer and then later read it out for analysis and display. These functions implement a **digital oscilloscope** that is useful during system setup, PID tuning, and general motion with data acquisition synchronization.

Load Velocity Override

Load Base Velocity

Load Blend Factor

Load S-Curve Time

Load Position Modulus

Load Velocity Threshold in RPM

Load Velocity Threshold

Load Torque Offset

Load Torque Limit

Read Target Position

Read Steps Generated

Read DAC

Read DAC Limit Status

Acquire Trajectory Data

Read Trajectory Data

Load Run/Stop Threshold

Device Compatibility

Acquire Trajectory Data

Usage

status = flex_acquire_trajectory_data(u8 boardID, u16 axisMap, u16 numberOfSamples, u16 timePeriod);

Purpose

Acquires time-sampled position and velocity data on multiple axes.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisMap	u16	bitmap of axes to acquire data for
numberOfSamples	u16	number of samples to acquire
timePeriod	u16	time period in milliseconds between samples

Parameter Discussion

axisMap is the bitmap of axes to acquire data for.

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis 15	 Axis 10	Axis 9	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	XXX

D1 through D8:

1 = Acquire samples on this axis

0 = Do not acquire samples (default)

numberOfSamples is the number of samples to acquire. The maximum number of samples depends upon the number of axes selected by **axisMap**.

numberOfSamples (max) = 4096/number of axes. With one axis selected, the maximum is 4,096 samples. With four axes selected, the maximum is 682 samples.

timePeriod is the time period between samples in ms. The range is from 3 (default) to 65,535 ms.

Using This Function

This function initiates the automatic acquisition of position and velocity data for the selected axes. The data is held in an onboard first-in-first-out (FIFO) buffer until later read back with the <u>Read Trajectory Data</u> function. You can select which axes to acquire data for and program the time period between samples. The Acquire Trajectory Data and Read Trajectory Data functions are used to acquire and read back time-sampled position and velocity data for analysis and display. These functions implement a digital oscilloscope that is useful during system setup, PID tuning, and general motion with data acquisition synchronization.

Caution Wait an appropriate amount of time before attempting to read back the trajectory data.

After it is started, this data acquisition operates autonomously in the background as a separate task. Motion control operates normally and you can execute other motion functions simultaneously. Depending upon the programmed time period and the total number of samples, this acquisition task can run anywhere from a few milliseconds to tens of hours.

Because host communication, event handling, arc point generation, contour point dispensing, buffered breakpoints, and buffered high-speed capture are a higher priority on the controller than acquiring trajectory data, you may notice irregular time spacing of sampled data when host communications or event notification is heavy. To minimize the chance of irregular data, limit host communication to a minimum while acquiring trajectory data. If you still notice irregular data, increase the time period.

Example

To acquire 100 samples of data on axes 1 and 2 at 10 ms/sample, call the Acquire Trajectory Data function with the following parameters:

D15	••••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis 15	••••	Axis 10	Axis 9	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	XXX

axisMap = 0x0006, corresponding to the following bitmap:

00 0 0 0 0 0 0 1 1

numberOfSamples = 100 timePeriod = 10

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

This function cannot handle axes greater than fifteen.

flex_load_base_vel

Device Compatibility

Load Base Velocity

Usage

status = flex_load_base_vel(u8 boardID, u8 axis, u16 baseVelocity);
Purpose

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

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to control
baseVelocity	u16	base velocity for the stepper axis in steps/second

Parameter Discussion

axis is the axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

baseVelocity is loaded in steps per second and is a 16-bit data word in the range of 0 through 65,535. The default value is 0.

Using This Function

Base velocity is the minimum step rate used by the trajectory generator during acceleration and deceleration. Base velocity is useful when the system uses full-step or half-step mode. Base velocity is not necessary if the system uses microstepping.

If the target velocity loaded with the <u>Load Velocity</u> function is lower than the base velocity, the base velocity is reduced to equal the loaded target velocity.



Note This function is valid only on axes configured as steppers, so you must configure an axis as a stepper using the <u>Configure</u> <u>Axis Resources</u> function before executing this function.

Example

If the base velocity loaded is 2000 steps/s and the loaded velocity is 5000, the axis starts at the base velocity and accelerates to the loaded velocity. The axis then decelerates to the base velocity and stops, as shown in the following figure.



flex_load_blend_fact

Device Compatibility

Load Blend Factor

Usage

status = flex_load_blend_fact(u8 boardID, u8 axisOrVectorSpace, i16 blendFactor, u8 inputVector);

Purpose

Loads the blend factor for an axis or vector space.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to control
blendFactor	i16	the mode and/or dwell used during blending
inputVector	u8	source of the data for this function

Parameter Discussion

axisOrVectorSpace is the axis or vector space to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

blendFactor is the blend factor mode and/or dwell time. –1 specifies normal blending (default), 0 specifies a start after the previous move is fully stopped, and values > 0 specify additional dwell time in milliseconds.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function controls how the <u>Blend Motion</u> function operates. Blending automatically starts a pending move on an axis or vector space when the move in process completes. Exactly when the pending move starts is determined by the loaded blend factor.

A blend factor of -1 causes the pending move to start when the existing move finishes its constant velocity segment and starts to decelerate, as shown in the following figure. This blends the two moves together at the optimum blend point.



If the two moves are at the same velocity, in the same direction, and have matching acceleration and deceleration, they superimpose perfectly without a dip or increase in axis velocity. Vector velocity dips based on the amount of deflection from a straight line.

For a vector move, if all of the axes are continuing in the same direction, the vector velocity remains constant. But, if one of the axes changes direction, the vector velocity does not remain constant during the transition phase. A blend factor of zero (0) causes the pending move to start when the existing move fully completes its profile, as shown in the following figure.



Positive blend factors allow for a dwell at the end of the first move before the automatic start of the pending move, as shown in the following figure. The blend factor dwell is programmed in milliseconds.

The maximum value of the positive blend factor depends upon the PIDrate that you set in the Enable Axes function, because the DSP delays the trajectory generators based on PID sample periods. The formula used to determine the maximum positive blend factor is as follows:

 $s = \frac{t \times 1,000}{PID}$

where *s* is the time in sample periods, *t* is the positive blend factor value in milliseconds, and PID is the PID rate in microseconds (62.5, 125, 188, 250, 312, 375, 438, or 500).

If s > 32,767, it is coerced to 32,767 sample periods. At a PIDrate of 500 μ s, the maximum value of the positive blend factor is 16,383 ms and at a PIDrate of 250 μ s, the maximum value is 8,192 ms.



If the first move has already completed when the <u>Blend Motion</u> function is executed, the second move still waits the dwell time before starting.

You can load blend factors to individual axes or to a vector space for coordinated blending of all axes in the vector space. When sent to a vector space, the blend factor is broadcast to all axes in the vector space to change the per-axis blend factors. If you later want to operate an axis independently with a different blend factor, you must execute the Load Blend Factor function again for that axis.

 $\overline{\mathbb{N}}$

Note All axes in a coordinate (vector) space must have the same blend factor. If the blend factors are different on each axis when you execute a <u>Blend Motion</u> function, an error is generated.

flex_load_follow_err

Device Compatibility

Load Following Error

Usage

status = flex_load_follow_err(u8 boardID, u8 axis, u16 followingError, u8
inputVector);

Purpose

Loads the following error trip point.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to control
followingError	u16	following error trip point in counts
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

followingError is the following error trip point in encoder counts. If the following error exceeds this value, the axis is automatically killed. The range is 0 to 32,767 with a default of 32,767 counts. Loading zero (0) is a special case that disables the following error trip functionality.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function sets the maximum allowable following error. Following error is the difference between the instantaneous commanded trajectory position and the feedback position. If the absolute value of this difference exceeds the trip point, an internal kill stop is issued and the axis is disabled.

If this axis is part of a coordinate (vector) space, all other axes are commanded to decelerate to a stop when one axis trips on following error.

This function is a safety feature used to protect the motion hardware and associated system components from damage when the position error gets excessive due to friction, binding, or a completely stalled motor. It also protects you in case you load unobtainable values for velocity and/or acceleration.

This feature is available on all servo and closed-loop stepper axes. It has no effect on stepper axes running in open-loop mode. You can completely disable the following error feature by loading a zero (0) value.

Caution Following error must not be disabled unless the application absolutely requires operating with greater than 32,767 counts of error.

You can monitor following error status with the <u>Read Trajectory Status</u> or <u>Read per Axis Status</u> functions. A following error trip always sets the Motor Off status. You can further diagnose the cause of the trip by checking the torque limit status with the <u>Read DAC Output Limit Status</u> function.

If an axis trips on following error, that axis, or any axes associated through a coordinate space, are prohibited from starting from an onboard program until a start from the host, pause of the onboard program, or run of an onboard program.

In general, a following error trip is considered normal operation and does not generate an error. There are a few cases where an unexpected following error trip generates a modal error: during <u>Find Reference</u> and while executing a stored program.



Note Following error limit cannot be set for a vector space, you

must set a following error limit for each axis individually. For coordinate spaces, following error data is truncated to a maximum of 32,767.

If an axis is tripped on following error, call the <u>Stop Motion</u> function with **stopType** set to NIMC_HALT_STOP to remove the error status.

Refer to the <u>Errors and Error Handling</u> topic for information about errors and error handling.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- The NI SoftMotion Controller does not support loading zero (0) in **followingError** to disable the following error.
- For the **inputVector** parameter, the NI SoftMotion Controller supports only the immediate return vector (0XFF).
- Starting a move with a following error returns an error. You must clear the following error by calling <u>Stop Motion</u> with **stopType** set to NIMC_HALT_STOP.

Device Compatibility

Load Move Constraints

Usage

i32 status = flex_load_move_constraint (i32 boardID, i32 axisID, TnimcMoveConstraint attribute, NIMC_DATA* value);

Purpose

Loads move constraints in user units.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	i32	assigned by Measurement & Automation Explorer (MAX)
axisID	i32	axis or vector space to control
attribute	TnimcMoveConstraint	type of move constraint to load
value	NIMC_DATA*	value of move constraint to load

Parameter Discussion

axisOrVectorSpace is the axis or vector space to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

attribute is the move constraint to load.

attribute Constant	attribute Value
TnimcMoveConstraintVelocity	0
TnimcMoveConstraintAcceleration	1
TnimcMoveConstraintDeceleration	2
TnimcMoveConstraintAccelerationJerk	3
TnimcMoveConstraintDecelerationJerk	4

value is the value to load for the specified attribute. Use the **doubleData** element of the following structure to load the attribute value:

```
struct
{
    i32 longData;
    u8 boolData;
    f64 doubleData;
} NIMC DATA;
```

Use one of the following steps to load the appropriate value:

- **TnimcMoveConstraintVelocity**—Set **doubleData** to velocity limit in user unit/second.
- **TnimcMoveConstraintAcceleration**—Set **doubleData** to acceleration limit in user unit/second².
- **TnimcMoveConstraintDeceleration**—Set **doubleData** to deceleration limit in user unit/second².
- **TnimcMoveConstraintAccelerationJerk**—Set **doubleData** to acceleration jerk limit in user unit/second³.
- TnimcMoveConstraintDecelerationJerk—Set doubleData to

deceleration jerk limit in user unit/second³.

Using This Function

This function loads move constraints in user units using the **doubleData** element of the NIMC_DATA structure. Use <u>Load Counts/Steps per</u> <u>Revolution</u> to set the user unit ratio.

Example 1

If you load 2,000 counts/rev using the <u>Load Counts/Steps per Revolution</u> function, and then you call Load Move Constraints, with NIMC_VELOCITY_CONSTRAINT set to 10.5, the velocity loaded is 10.5 revolutions/second or 21,000 counts/second.

Example 2

If you set **countsOrSteps** to 1 in the Load Counts/Steps per Revolution function, and then you call Load Move Constraints, with NIMC_VELOCITY_CONSTRAINT set to 1.3, the velocity loaded is 1.3 counts/second.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- 73xx controllers do not support the **TnimcMoveConstraintAccelerationJerk** and **TnimcMoveConstraintDecelerationJerk** attributes.
- Use <u>Load Velocity</u> and <u>Load Acceleration/Deceleration</u> for faster performance.

flex_load_pos_modulus

Device Compatibility

Load Position Modulus

Usage

status = flex_load_pos_modulus(u8 boardID, u8 axis, u32 positionModulus, u8
inputVector);

Purpose

Loads the position modulus for an axis.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to control
positionModulus	u32	position modulus value in counts or steps
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

positionModulus is the position modulus value in counts (servo axes) or steps (stepper axes). The modulus range is from 0 (default) to 2^{31} -1.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function sets the modulus used when the axis is operating in Modulus Position mode. It has no effect when the axis is operating in other modes. When a target position is loaded, it is interpreted within the boundaries of a modulus range.

Refer to the <u>Set Operation Mode</u> function for a complete description of the Modulus Position mode.

flex_load_rpm

Device Compatibility

Load Velocity in RPM

Usage

status = flex_load_rpm(u8 boardID, u8 axisOrVectorSpace, f64 RPM, u8
inputVector);

Purpose

Loads velocity for an axis or vector space in RPM.


Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to control
RPM	f64	velocity in RPM
inputVector	u8	source of the data for this function

Parameter Discussion

axisOrVectorSpace is the axis or vector space to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

RPM is the double precision floating point velocity value in RPM. The RPM range depends upon the motor counts or steps per revolution and the trajectory update rate.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function specifies the maximum trajectory velocity for individual axes or vector spaces. When executed on a vector space, the value controls the vector velocity along the vector move path. For velocity control applications, the sign of the loaded velocity specifies the move direction. This function requires previously loaded values of either counts per revolution (for servo axes) or steps per revolution (for stepper axes) to operate correctly.

RPM is double-buffered so you can load it on the fly without affecting the move in process, and it takes effect on the next <u>Start Motion</u> or <u>Blend</u> <u>Motion</u> call. After it is loaded, this parameter remains in effect for all subsequent motion profiles until re-loaded by this function. You do not need to load velocity before each move unless you want to change the velocity.



Note The velocity loaded with this function is the maximum move velocity. Actual velocity attainable is determined by many factors including PID tuning, length of move, acceleration and deceleration values, and physical constraints of the amplifier/motor/mechanical system.

You also can load velocity in counts/s or steps/s by calling the <u>Load Velocity</u> function.

- Tip Use Load Move Constraints instead of this function for more move options.
- Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- For 73xx motion controllers, RPM values stored in onboard variables are in double-precision IEEE format (f64). Refer to <u>Using</u> Inputs and Return Vectors with Onboard Variables for information about the number of variables required to hold an RPM value.
 - \mathbb{N}
 - Note NI-Motion firmware uses a 16.16 fixed-point format internally to represent velocity in counts/sample period or steps/sample period and acceleration/deceleration in counts/sample period² or steps/sample period². If you change counts or steps per revolution or change the update rate using Enable Axis, you must reload the velocity and acceleration/deceleration.
- Velocity values in RPM are converted to an internal 16.16 fixedpoint format in units of counts (steps) per sample period (update period) before being used by the trajectory generator. NI-Motion can control velocity to 1/65,536 of a count or step per sample. You can calculate this minimum velocity increment in RPM with the following formula:

minimum RPM = Vmin × $(\frac{1}{T_s})$ × 60 × $(\frac{1}{R})$

where:

Vmin is 1/65,536 count/sample or step/sample,

Ts is the sample period in seconds per sample,

60 is the number of seconds in a minute, and

R is the counts/steps per revolution. For a typical servo axis with 2,000 counts per revolution operating at the 250 μ s update rate, the minimum RPM increment is:

 $\left(\frac{1}{65,536}\right) \times 4,000 \times \left(\frac{60}{2000}\right) = 0.00183105 \text{ RPM}$

You can calculate the maximum velocity in RPM with the following equation:

maximum RPM = Vmax × $60 \times \frac{1}{R}$

where:

Vmax is 20 MHz for servos, 8 MHz for steppers on a 7350 controller, 4 MHz for steppers on a 7330/40/90 controller, and *R* is the counts/steps per revolution,

and is constrained by acceleration/deceleration according to the following equation:

```
velocity \leq (65536 × deceleration) – acceleration
```

where velocity is in counts/sample and acceleration and deceleration are in counts/sample².

From the example, the maximum RPM is:

 $(20 \times 10^6) \times (\frac{60}{2000}) = 600,000 \text{ RPM}$

• Use <u>Load Velocity</u> and <u>Load Acceleration/Deceleration</u> for faster performance.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- Use <u>Load Move Constraints</u> to load the velocity in user units/second.
- The NI SoftMotion Controller does not convert velocity values in RPM to a 16.16 fixed point number. All calculations performed by the NI SoftMotion Controller maintain f64 precision.
- For the **inputVector** parameter, the NI SoftMotion Controller supports only the immediate return vector (0XFF).

flex_load_rpm_thresh

Device Compatibility

Load Velocity Threshold in RPM

Usage

status = flex_load_rpm_thresh(u8 boardID, u8 axis, f64 threshold, u8
inputVector);

Purpose

Loads a velocity threshold for an axis in RPM.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to control
threshold	f64	velocity threshold in RPM
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

threshold is the velocity threshold value in RPM expressed as a doubleprecision floating point number. The RPM range depends upon the motor counts or steps per revolution and the trajectory update rate, and is always a positive number.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function establishes a velocity threshold in RPM for the specified axis which can then be monitored with the <u>Read Trajectory Status</u> function. The velocity threshold status is True when the absolute value of filtered axis velocity is above the threshold and False when the velocity drops below the threshold.

Velocity threshold is a status and does not have to be enabled or disabled. Loading a maximum value effectively disables the feature because the status is always off. Increasing the velocity filter time constant with the <u>Load Velocity Filter Parameter</u> function reduces quantization noise in the threshold status but at the expense of increasing threshold status latency.

Velocity threshold is typically used to monitor the acceleration and deceleration trajectory periods to see when or if an axis is up to speed. You can then change PID tuning or other parameters as a function of velocity.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

• The NI SoftMotion Controller loads the velocity threshold in user units/minute. To load the axis velocity threshold in counts/second, use Load Counts/Steps per Revolution to set countsOrSteps value to 1.

flex_load_rpsps

Device Compatibility

Load Accel/Decel in RPS/sec

Usage

status = flex_load_rpsps(u8 boardID, u8 axisOrVectorSpace, u16 accelerationType, f64 RPSPS, u8 inputVector);

Purpose

Loads the maximum acceleration and/or deceleration value for an axis or vector space in RPS/s.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to control
accelerationType	u16	selector for acceleration, deceleration or both
RPSPS	f64	acceleration value in revolutions/s/s
inputVector	u8	source of the data for this function

Parameter Discussion

axisOrVectorSpace is the axis or vector space to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

accelerationType is the selector for loading acceleration, deceleration, or both acceleration and deceleration (default).

accelerationType Constant	accelerationType Value
NIMC_BOTH	0 (default)
NIMC_ACCELERATION	1
NIMC_DECELERATION	2

RPSPS is the double precision floating point acceleration and/or deceleration value in motor revolutions/s/s (RPS/s). The range for acceleration in RPS/s depends upon the motor counts or steps per revolution and the trajectory update rate.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function specifies the maximum rate of acceleration and/or deceleration for individual axes or vector spaces in revolutions/s². When executed on a vector space, the value controls the vector acceleration (deceleration) along the vector move path. This function requires previously loaded values of either counts per revolution (for servo axes) or steps per revolution (for stepper axes) to operate correctly.

You can use this function to load separate limits for acceleration and deceleration or to set them both to the same value with one call. These parameters are double-buffered so you can load them on the fly without affecting the move in process, and they take effect on the next <u>Start</u> Motion or Blend Motion call. After they are loaded, these parameters remain in effect for all subsequent motion profiles until re-loaded by this function. You do not need to load acceleration before each move unless you want to change the acceleration and/or deceleration value.

Acceleration defines how quickly the axis or axes come up to speed and is typically limited to avoid excessive stress on the motor, mechanical system, and/or load. A separate, slower deceleration is useful in applications where coming to a gentle stop is important.



Tip Use <u>Load Move Constraints</u> instead of this function for more move options.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- RPS/s values stored in onboard variables are in double-precision IEEE format (f64). Refer to <u>Using Inputs and Return Vectors with</u> <u>Onboard Variables</u> for information about the number of variables required to hold an RPS/s value.
- NI-Motion firmware uses a 16.16 fixed-point format internally to represent velocity in counts/sample period or steps/sample period and acceleration/deceleration in counts/sample period² or steps/sample period². If you change counts or steps per revolution or change the update rate using <u>Enable Axis</u>, you must reload the velocity and acceleration/deceleration.
- You also can load acceleration and deceleration in counts/s² or steps/s² by calling the <u>Load Acceleration/Deceleration</u> function.
- If you are doing onboard programming and are using **inputVector** to get the data this function needs, notice that this function reads the variables starting at the memory address pointed to by **inputVector** in the following order: **accelerationType**, **RPSPS**.
- Acceleration and deceleration values in RPS/s are converted to an internal 16.16 fixed-point format in units of counts/sample² or steps/sample² before being used by the trajectory generator. You can calculate the minimum acceleration increment in RPS/s with the following formula:

 $RPS/s = Amin \times \left(\frac{1}{T_{S}}\right)^{2} \times \frac{1}{R}$

where: Amin is 1/65,536 counts/sample² or steps/sample², *Ts* is the sample period in seconds per sample, and *R* is the counts or steps per revolution.

• For a typical servo axis with 2,000 counts per revolution operating at the 250 ms update rate, calculate the minimum RPS/s increment using the following equation:

 $\left(\frac{1}{65,536}\right) \times \left(\frac{4,000^2}{2,000}\right) = 0.122070 \text{ RPS/s}$

• You can calculate the maximum RPS/s using the following equation:

maximum RPS/s = Amax × $(\frac{1}{T_s})^2$ × $(\frac{1}{R})$ where: *Amax* is 32 counts/sample *Ts* is the sample period in seconds per sample, and *R* is the counts or steps per revolution,

and is constrained according to the following equations: acceleration \leq 256 × deceleration

deceleration ≤ 65536 × acceleration

• Use <u>Load Velocity</u> and <u>Load Acceleration/Deceleration</u> for faster performance.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- The minimum acceleration/deceleration value is not limited by the fixed 16.16 calculation. This function takes the acceleration/deceleration value as an f64.
- For the **inputVector** parameter, the NI SoftMotion Controller supports only the immediate return vector (0XFF).

Load Run/Stop Threshold

Usage

status = flex_load_run_stop_threshold(u8 boardID, u8 axis, u16
runStopThreshold, u8 inputVector);

Purpose

Sets the Run/Stop threshold, which affects the run/stop status returned by <u>Read Trajectory Status</u>.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to configure
runStopThreshold	u16	sets the velocity threshold above which an axis is considered running.
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the axis to configure. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

runStopThreshold is the Run/Stop threshold velocity in counts/sample period for servo axes or steps/sample period for open-loop and closed-loop stepper axes. The range for this parameter is 1 (default) to 32,767 sample periods.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function sets the velocity threshold above which an axis is considered running. The **runStopThreshold** parameter has time units of update sample periods and is affected by the update rate set in the <u>Enable Axis</u> function.

If the velocity of an axis is above the step/count per sample period specified in this function, the run/stop status from Read Trajectory Status returns TRUE. If it is below the threshold, the run/stop status returns FALSE.

flex_load_scurve_time

Device Compatibility

Load S-Curve Time

Usage

status = flex_load_scurve_time(u8 boardID, u8 axisOrVectorSpace, u16
sCurveTime, u8 inputVector);

Purpose

Loads the s-curve time for an axis or vector space.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to control
sCurveTime	u16	smoothing time in update sample periods
inputVector	u8	source of the data for this function

Parameter Discussion

axisOrVectorSpace is the axis or vector space to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

sCurveTime is the time in update sample periods over which the acceleration profile is smoothed as it transitions from zero to the programmed value and back to zero. The s-curve range is from 1 to 32,767 with a default of 1 sample period.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function smooths the acceleration and deceleration portions of a motion profile, resulting in less abrupt transitions from <u>Start Motion</u> to acceleration, acceleration to constant velocity, constant velocity to deceleration, and deceleration to stop. Using s-curve acceleration limits the jerk in a motion control system.

Officially, jerk is defined as the derivative of acceleration (change of acceleration per unit time) and is measured in units of counts (steps)/s³. This function, however, allows you to load s-curve time in update sample periods rather than have to deal with the obscure units of jerk.

With the default s-curve time of one (1) sample period, there is virtually no effect on the motion profile, and the standard trapezoidal trajectory is executed. As s-curve time increases, the smoothing affect on the acceleration and deceleration portions of the motion profile increase, as shown in the following figure. Large values of s-curve time can override the programmed values of acceleration and deceleration by sufficiently smoothing the profile such that the acceleration and deceleration slopes are never reached.



Note A large s-curve values makes the time to accelerate to the commanded velocity longer. Therefore, the time it takes to reach the target position is also longer. If the s-curve value is too large, the value is ignored and full acceleration is used. This happens when the acceleration and deceleration become too long to perform smoothly.

You can load s-curve time to individual axes or to a vector space for

smoothing all axes in the vector space. When sent to a vector space, the s-curve time is broadcast to all axes in the vector space to change the per-axis s-curve times. If you later want to operate an axis independently with a different s-curve time, you must execute this function again for that axis.



Note All axes in a vector space must have the same s-curve time for best vector accuracy.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- When you use this function with the NI SoftMotion Controller, **sCurveTimeupdate** is the smoothing time in the host loop period.
- National Instruments suggests you use the <u>Load Move Constraints</u> function with the NI SoftMotion Controller instead of using this function.

flex_load_torque_lim

Device Compatibility

Load Torque Limit

Usage

status = flex_load_torque_lim(u8 boardID, u8 axis, i16 primaryPositiveLimit, i16 primaryNegativeLimit, i16 secondaryPositiveLimit, i16 secondaryNegativeLimit, u8 inputVector);
Purpose

Loads primary and secondary DAC torque limits for an axis.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to control
primaryPositiveLimit	i16	positive limit for primary DAC
primaryNegativeLimit	i16	negative limit for primary DAC
secondaryPositiveLimit	i16	positive limit for optional secondary DAC
secondaryNegativeLimit	i16	negative limit for optional secondary DAC
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

primaryPositiveLimit is the primary DAC positive torque (or velocity) limit. The range is -32,768 to + 32,767 (-10 V to +10 V) with a default value of 32,767 (+10 V).

primaryNegativeLimit is the primary DAC negative torque (or velocity) limit. The range is -32,768 to + 32,767 (-10 V to +10 V) with a default value of -32,768 (-10 V).

Note The positive limit cannot be less than the negative limit.

secondaryPositiveLimit is the optional secondary DAC positive torque (or velocity) limit. The range is -32,768 to +32,767 (-10 V to +10 V) with a default value of 32,767 (+10 V).

secondaryNegativeLimit is the optional secondary DAC negative torque (or velocity) limit. The range is -32,768 to +32,767 (-10 V to +10 V) with a default value of -32,768 (-10 V).

Note The positive limit cannot be less than the negative limit.

inputVector contains the primary positive, primary negative, secondary positive, and secondary negative parameters that are the source of data for this function.

Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return</u> <u>Vectors</u> for more detailed information.

Using This Function

This function allows you to limit the output range of the DAC output(s) on the selected servo axis. This function has no effect on stepper axes or independent DAC outputs that are not mapped to an axis.

By limiting the output range of a DAC, it is possible to control the maximum torque (when connected to a torque block servo amplifier) or velocity (when connected to a velocity block servo amplifier). This function is also helpful when interfacing to amplifiers that do not support the standard ±10 V command range.

Primary and secondary DACs can have different limits, and the positive and negative limits can be both positive or both negative to limit the DAC output to a unipolar range. The only restriction is that a positive DAC limit cannot be less than the negative DAC limit.

You also can set a torque offset on the primary and secondary DAC outputs. Refer to the <u>Load Torque Offset</u> function for more information.

Example

Calling the Load Torque Limit function with the following parameters limits the output ranges of the primary and secondary DACs mapped to the axis, as shown in the following figure.



The result of this function call is to limit the primary DAC to only half its range in either direction, or ± 5 V. The secondary DAC can only travel over a quarter of its positive range but has its full negative range.

flex_load_torque_offset

Device Compatibility

Load Torque Offset

Usage

status = flex_load_torque_offset(u8 boardID, u8 axis, i16 primaryOffset, i16 secondaryOffset, u8 inputVector);

Purpose

Loads primary and secondary DAC torque offsets for an axis.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to control
primaryOffset	i16	offset for primary DAC
secondaryOffset	i16	offset for secondary DAC
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

primaryOffset is the primary DAC torque (or velocity) offset. The offset range is -32,768 to +32,767 (-10 V to +10 V) with a default value of 0 (0 V).

secondaryOffset is the secondary DAC torque (or velocity) offset. The offset range is –32,768 to +32,767.



Note The offset value must be within the range limits set by the Load Torque Limit function.

inputVector contains the primary and secondary parameters that are the source of data for this function.

Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return</u> <u>Vectors</u> for more detailed information.

Using This Function

This function loads offset values for the DACs mapped to the selected servo axis. This function has no effect on stepper axes or independent DAC outputs that are not mapped to an axis. When a DAC is connected to a velocity block servo amplifier, the torque offset functions as a velocity offset.

A torque (or velocity) offset shifts the DAC output(s) by the programmed offset value without requiring any action from the PID loop. In a servo system, this can be used to overcome amplifier input offsets, system imbalances, or the effects of outside forces such as gravity. Different torque offsets can be loaded for the primary and secondary DAC.



Note When an axis is killed, its DAC outputs are zeroed regardless of the torque offset loaded.

DAC offsets can be used in conjunction with DAC range limits to interface to servo amplifiers with unipolar input ranges (for example, 0 to 5 V or 0 to 10 V).

Example

Calling the Load Torque Offset function with **primaryOffset** = 4,096 and **secondaryOffset** = 0 shifts the output ranges of the primary DAC mapped to the axis as shown in the following figure.



The result of this function call is to limit the primary DAC to a range of - 8.75 V to +10 V with an offset or null value of +1.25 V. This is because even when the PID loop is commanding full negative torque, the torque offset is added and the resulting output is -8.75 V. In the positive direction, the DAC cannot go above +10 V no matter what the offset is.

The function call leaves the secondary DAC offset at its default value of zero (0). This example assumes the full torque range is available and not limited by the Load Torque Limit function.



Note The offset value must be within the range limits set by the Load Torque Limit function.

Device Compatibility

Load Velocity Override

Usage

status = flex_load_velocity_override(u8 boardID, u8 axisOrVectorSpace, f32
overridePercentage, u8 inputVector);

Purpose

Loads an instantaneous velocity override for an axis or vector space.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to control
overridePercentage	f32	velocity override scale factor
inputVector	u8	source of the data for this function

Parameter Discussion

axisOrVectorSpace is the axis or vector space to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

overridePercentage directly scales the programmed velocity. The default value is 100% (no effect).

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function scales the operating velocity on an axis or vector space from 0 to 150%. Velocity override is not double-buffered. The function takes effect immediately and does not require a <u>Start Motion</u> or <u>Blend</u> <u>Motion</u> function execution to change the operating velocity. All velocity changes use the loaded values of acceleration, deceleration, and s-curve to smoothly transition the velocity to its new value.

Velocity override scales velocity in linear interpolation as well as in circular, helical, and spherical arcs.

Note Velocity override is not valid for contoured moves.

You can load velocity override to individual axes or to a vector space for coordinated velocity scaling. When sent to a vector space, the velocity override is broadcast to all axes in the vector space to change the peraxis overrides. If you later want to operate an axis independently with a different velocity override, you must execute this function again for that axis. Note that this also works with gearing. In addition to following the position of the master axis, the geared slave axis follows the velocity of the master axis.

After it is loaded, velocity override remains in effect until changed by another call to this function. All subsequent moves are at velocities scaled by the most recent override percentage. At power-up reset, velocity override is always reset to 100%.

Velocity override is commonly used in machine tool and other applications to reduce the speed of a programmed motion sequence and can be used to implement a feed hold by setting the value to zero (0). You can directly use a scaled value from an analog input as the velocity override value.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- The conversion from floating-point to fixed-point is performed on the host computer, not on the motion controller. To load velocity override from an onboard variable, you must use the integer representation of 0 to 384.
- The resolution of this function is approximately 0.4%.
- All axes in a vector space must have the same velocity override. If axes have different velocity overrides, the vector move cannot function as expected. This mode is legal and does not generate an error, but it causes the axes to finish their moves at different times.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- If velocity override is loaded on an axis, it is applied only during single-axis moves.
- If the velocity override is loaded on a vector space, it does not overwrite the values for the axes that are in the vector space. In this case, the velocity override is applied only during vector space moves.
- The range for the **overridePercentage** is the full 64-bit range, and is not limited to 150% for the NI SoftMotion Controller.

Load Velocity Threshold

Usage

status = flex_load_velocity_threshold(u8 boardID, u8 axis, u32 threshold, u8
inputVector);

Purpose

Loads a velocity threshold for an axis.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to control
threshold	u32	velocity threshold in counts/s or steps/s
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

threshold is the velocity threshold in counts/s (servo axes) or steps/s (stepper axes). For servo axes, the threshold range is 1 to 16,000,000 counts/s. For stepper axes, it is 1 to 1,500,000 steps/s. The factory default value for threshold is the maximum, so the feature is effectively disabled until a threshold is loaded.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function establishes a velocity threshold for the specified axis, which can then be monitored with the <u>Read Trajectory Status</u> function. The velocity threshold status is True when the absolute value of filtered axis velocity is above the threshold and False when the velocity drops below the threshold.

Velocity threshold is a status and does not have to be enabled or disabled. Loading a maximum value effectively disables the feature because the status is always off. Increasing the velocity filter time parameter with the <u>Configure Velocity Filter</u> function reduces quantization noise in the threshold status but at the expense of increasing threshold status latency.

Velocity threshold is typically used to monitor the acceleration and deceleration trajectory periods to see when or if an axis is up to speed.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

• Use Load Counts/Steps per Revolution to set countsOrSteps value to 1. Otherwise, NI-Motion returns an error.

flex_read_blend_status

Device Compatibility

Read Blend Status

Usage

status = flex_read_blend_status(u8 boardID, u8 axisOrVectorSpace, u8
returnVector);

Purpose

Reads the Blend Complete status for all axes or vector spaces.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space selector
returnVector	u8	destination for the return data

Parameter Discussion

axisOrVectorSpace can select an axis (NIMC_AXIS1 through NIMC_AXIS30), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously reading the blend status of multiple axes or vector spaces, the **blendStatus** parameter indicates which axes or vector spaces are involved. On motion controllers that support fewer than thirty axes, reading non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no returnVector is required.

Using This Function

Blending smoothly combines two move segments on an axis, axes, or vector space(s). When continuously blending move segments into each other, it is necessary to wait until the blend is complete between the previous two moves before you load the trajectory parameters for the next move to blend. The status information returned by this function indicates that the previous blend is complete and the axis, axes, or vector space(s) are ready to receive the next blend move trajectory data.

Attempting to execute a Blend Motion function before the previous blend is complete on the axes involved generates a modal error.

Example

While blending linearly interpolated moves in a 2D vector space, you call the Read Blend Status function with **axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL to select vector space status. If the blend on vector space 1 is still pending, this function returns **blendStatus** = 0x000C, which corresponds to the following bitmap:

D15	••••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS15		VS10	VS9	VS8	VS7	VS6	VS5	VS4	VS3	VS2	VS1	XXX
0	••••	0	0	0	0	0	0	0	1	1	0	0

The blend is complete (1) on vector spaces 2 and 3 (or they do not exist), but the blend is still pending (0) on vector space 1. For your programming convenience, two utility functions—<u>Check Blend Complete Status</u> and <u>Wait for Blend Complete</u>—allow you to specify an axis, vector space, group of axes, or group of vector spaces, and find out if a blend is complete, or wait until a blend is complete. These functions return a simple true/false value indicating if a blend is complete.

Device Compatibility

Read Blend Status Return

Usage

status = flex_read_blend_status_rtn(u8 boardID, u8 axisOrVectorSpace, u16* blendStatus);

Purpose

Reads the Blend Complete status for all axes or vector spaces.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space selector
blendStatus	u16*	bitmap of blend complete status for all axes or vector spaces

Parameter Discussion

axisOrVectorSpace can select an axis (NIMC_AXIS1 through NIMC_AXIS30), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously reading the blend status of multiple axes or vector spaces, the **blendStatus** parameter indicates which axes or vector spaces are involved. On motion controllers that support fewer than thirty axes, reading non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector</u> <u>Spaces</u> for axis and vector space resource IDs.

blendStatus is a bitmap of blend complete status for all axes or all vector spaces.

When reading blend status for axes (**axisOrVectorSpace** = NIMC_AXIS_CTRL):

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis	 Axis	Axis	0								
15	10	9	8	7	6	5	4	3	2	1	

D1 through D15:

1 = Blend complete on axis

0 = Blend pending

When reading blend status for vector spaces (**axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL):

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS 15	 VS 10	VS 9	VS 8	VS 7	VS 6	VS 5	VS 4	VS 3	VS 2	VS 1	0

D1 through D15:

1 = Blend complete on vector space

0 = Blend pending

Using This Function

Blending smoothly combines two move segments on an axis, axes, or vector space(s). When continuously blending move segments into each other, it is necessary to wait until the blend is complete between the previous two moves before you load the trajectory parameters for the next move to blend. The status information returned by this function indicates that the previous blend is complete and the axis, axes, or vector space(s) are ready to receive the next blend move trajectory data.

Attempting to execute a Blend Motion function before the previous blend is complete on the axes involved generates a modal error.

Example

While blending linearly interpolated moves in a 2D vector space, you call the Read Blend Status function with **axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL to select vector space status. If the blend on vector space 1 is still pending, this function returns **blendStatus** = 0x000C, which corresponds to the following bitmap:

D15	••••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS 15		VS 10	VS 9	VS 8	VS 7	VS 6	VS 5	VS 4	VS 3	VS 2	VS 1	XXX
0		0	0	0	0	0	0	0	1	1	0	0

The blend is complete (1) on vector spaces 2 and 3 (or they do not exist), but the blend is still pending (0) on vector space 1. For your programming convenience, two utility functions—<u>Check Blend Complete Status</u> and <u>Wait for Blend Complete</u>—allow you to specify an axis, vector space, group of axes, or group of vector spaces, and find out if a blend is complete, or wait until a blend is complete. These functions return a simple true/false value indicating if a blend is complete.
Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:



Note You cannot execute this function on multiple axes using the NIMC_AXIS_EX_CTRL resource parameter with axes sixteen through thirty.

flex_read_dac

Device Compatibility

Read DAC (to var)

Usage

status = flex_read_dac(u8 boardID, u8 axisOrDAC, u8 returnVector);

Purpose

Reads the commanded DAC output value for an axis.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrDAC	u8	axis or DAC to read
returnVector	u8	destination for the return data

Parameter Discussion

axisOrDAC is the axis or DAC to read. Valid axis or DAC resources are NIMC_AXIS1 through NIMC_AXIS30 and NIMC_DAC1 through NIMC_DAC30. On motion controllers that support fewer than thirty axes, reading non-existent axes or DACs returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>DAC</u> <u>Outputs</u> for axis and DAC resource IDs.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to Input and Return Vectors for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no returnVector is required.

Using This Function

This function returns the value of the specified DAC output. When sent to an axis, this function returns the value of the primary DAC mapped to that axis. The signed 16-bit value returned corresponds to the ± 10 V full scale range of the DAC.

This function is used to monitor the output command from the PID loop. When the DAC output is connected to a torque block servo amplifier, you can use this value to calculate motor torque or to monitor the acceleration and deceleration portions of a trajectory to see how close the control loop is to saturating at its maximum torque limits.

When the DAC output is connected to a velocity block servo amplifier, the DAC value read is a direct representation of the instantaneous commanded velocity.

flex_read_dac_output_limit_status

Device Compatibility

Read DAC Output Limit Status

Usage

status = flex_read_dac_output_limit_status(u8 boardID, u8 returnVector);

Purpose

Reads the status of the DAC limits.

Parameters

Name	Туре	Description		
boardID	u8	assigned by Measurement & Automation Explorer (MAX)		
returnVector	u8	destination for the return data		

Parameter Discussion

Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns the positive and negative torque limit status of the DAC output. Independent DACs that are not mapped to axes do not have torque limits, so those DACs always return False.

A DAC torque limit status is True (1) when the DAC output is saturated at the corresponding limit. This information tells you that the motor is operating at its maximum torque, probably due to an excessively high value of acceleration or deceleration. It also can indicate excessive friction on the axis, a completely stalled motor, or some other system fault.

When an axis is active (not in the killed, motor off state), this function returns the instantaneous state of the torque limit circuits. If the axis trips out on following error (a typical occurrence when operating at the torque limits), the DAC limit status is latched so you can tell which limit, positive or negative, caused the following error trip. The status remains latched until the axis is activated again by a <u>Start Motion</u>, <u>Stop Motion</u>, or <u>Blend Motion</u> function.

flex_read_dac_output_limit_status_rtn Device Compatibili

Read DAC Output Limit Status Return

Usage

status = flex_read_dac_output_limit_status_rtn(u8 boardID, u16* positiveStatus, u16* negativeStatus);

Purpose

Reads the status of the DAC limits.



Parameters

Name	Туре	Description
boardID u8		assigned by Measurement & Automation Explorer (MAX)
positiveStatus	u16*	bitmap of positive DAC limit status
negativeStatus	u16*	bitmap of positive DAC limit status

Parameter Discussion

positiveStatus is a bitmap of the positive DAC torque limit status.

D8	D7	D6	D5	D4	D3	D2	D1	D0
DAC 8	DAC 7	DAC 6	DAC 5	DAC 4	DAC 3	DAC 2	DAC 1	XXX

For D1 through D8:

1 = DAC output at positive limit

0 = DAC output below positive limit

negativeStatus is a bitmap of the negative DAC torque limit status.

D8	D7	D6	D5	D4	D3	D2	D1	D0
DAC 8	DAC 7	DAC 6	DAC 5	DAC 4	DAC 3	DAC 2	DAC 1	XXX

For D1 through D8:

1 = DAC output at negative limit

0 = DAC output above negative limit

Using This Function

This function returns the positive and negative torque limit status of the DAC output. Independent DACs that are not mapped to axes do not have torque limits, so those DACs always return False.

A DAC torque limit status is True (1) when the DAC output is saturated at the corresponding limit. This information tells you that the motor is operating at its maximum torque, probably due to an excessively high value of acceleration or deceleration. It also can indicate excessive friction on the axis, a completely stalled motor, or some other system fault.

When an axis is active (not in the killed, motor off state), this function returns the instantaneous state of the torque limit circuits. If the axis trips out on following error (a typical occurrence when operating at the torque limits), the DAC limit status is latched so you can tell which limit, positive or negative, caused the following error trip. The status remains latched until the axis is activated again by a <u>Start Motion</u>, <u>Stop Motion</u>, or <u>Blend Motion</u> function.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

To read the DAC limit status of DACs nine through thirty, use the <u>Read</u> <u>Motion I/O Data</u> function. flex_read_dac_rtn

Device Compatibility

Read DAC

Usage

status = flex_read_dac_rtn(u8 boardID, u8 axisOrDAC, i16* DACValue);

Purpose

Reads the commanded DAC output value for an axis.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrDAC	u8	axis or DAC to read
DACValue	i16*	commanded DAC output value

Parameter Discussion

axisOrDAC is the axis or DAC to read. Valid axis or DAC resources are NIMC_AXIS1 through NIMC_AXIS30 and NIMC_DAC1 through NIMC_DAC30. On motion controllers that support fewer than thirty axes, reading non-existent axes or DACs returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>DAC</u> <u>Outputs</u> for axis and DAC resource IDs.

DACValue is the 16-bit commanded DAC output value from the PID loop, where +32,767 corresponds to +10 V output and -32,768 corresponds to -10 V output.

Using This Function

This function returns the value of the specified DAC output. When sent to an axis, this function returns the value of the primary DAC mapped to that axis. The signed 16-bit value returned corresponds to the ± 10 V full scale range of the DAC.

This function is used to monitor the output command from the PID loop. When the DAC output is connected to a torque block servo amplifier, you can use this value to calculate motor torque or to monitor the acceleration and deceleration portions of a trajectory to see how close the control loop is to saturating at its maximum torque limits.

When the DAC output is connected to a velocity block servo amplifier, the DAC value read is a direct representation of the instantaneous commanded velocity.

flex_read_follow_err

Device Compatibility

Read Following Error

Usage

status = flex_read_follow_err(u8 boardID, u8 axisOrVectorSpace, u8
returnVector);

Purpose

Reads the instantaneous following error for an axis or vector space.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to read
returnVector	u8	destination for the return data

Parameter Discussion

axisOrVectorSpace is the axis or vector space to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, reading non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector</u> <u>Spaces</u> for axis and vector space resource IDs.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns the instantaneous following error for the axis or vector space specified. For vector spaces, following error is the root-mean-square of the following errors for the individual axes that make up the vector space.

flex_read_follow_err_rtn

Device Compatibility

Read Following Error Return

Usage

status = flex_read_follow_err_rtn(u8 boardID, u8 axisOrVectorSpace, i16*
followingError);

Purpose

Reads the instantaneous following error for an axis or vector space.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space to read
followingError	i16*	instantaneous following error for an axis or vector space in counts
Parameter Discussion

axisOrVectorSpace is the axis or vector space to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15. On motion controllers that support fewer than thirty axes, reading non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector</u> <u>Spaces</u> for axis and vector space resource IDs.

followingError is the instantaneous difference between the commanded trajectory position and the actual feedback position in counts for servo systems and steps for stepper systems.

Using This Function

This function returns the instantaneous following error for the axis or vector space specified. For vector spaces, following error is the root-mean-square of the following errors for the individual axes that make up the vector space.

flex_read_mcs_rtn

Device Compatibility

Read Move Complete Status

Usage

status = flex_read_mcs_rtn(u8 boardID, u16* moveCompleteStatus);

Purpose

Reads the Move Complete Status register.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
moveCompleteStatus	u16*	bitmap of Move Complete Status for all axes

Parameter Discussion

moveCompleteStatus is a bitmap of Move Complete Status for axes one through eight. The bitmap also includes the state of the three User Status bits.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	E
Sts15	Sts14	Sts13	XXX	XXX	XXX	XXX	MC 8	MC 7	MC 6	MC 5	MC 4	Ν

D1 through D8:

- 1 = Move is complete
- 0 = Axis is moving

D13 through D15 User Status (Sts):

- 1 = True
- 0 = False



Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- This function performs a direct read of the Move Complete Status (MCS) register on the 73xx controller. Because a register read is virtually instantaneous and does not affect communication processing or other NI-Motion operations, you can call this function repeatedly to get the most up to date status for the axes.
- Move Complete Status is configurable individually for each axis with the <u>Configure Move Complete Criteria</u> function. The criteria for considering motion to be complete include Profile Complete, Run/Stop, In Position, Settling time delay, and so on.
 - Note Reading the MCS register immediately after calling the <u>Start Motion</u> function might not return the status you expected. The Start Motion can still be buffered in the communications FIFO when the instantaneous read of the MCS occurs.
- This function also returns the state of the User Status bits. You can set and reset these three bits during onboard program execution as general-purpose flags to the host computer. Refer to the <u>Set</u> <u>User Status MOMO</u> function for more information.
 - Note When the 73xx motion controller is in the Power-Up state, the MCS register contains a power-up code that describes why the controller is in the Power-Up state. Refer to the <u>Clear Power Up Status</u> function for a list of these power-up codes.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

• During the power-up state on the NI SoftMotion Controller, this

function returns zero (0). It does not return a power-up code.

- The User Status bits are not supported by the NI SoftMotion Controller.
- To view the Move Complete Status for axes above eight, use the Read per Axis Status Return function.

flex_read_steps_gen

Device Compatibility

Read Steps Generated

Usage

status = flex_read_steps_gen(u8 boardID, u8 axisOrStepperOutput, u8
returnVector);

Purpose

Reads the number of steps generated by a stepper output.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrStepperOutput	u8	axis or stepper output to read
returnVector	u8	destination for the return data

Parameter Discussion

axisOrStepperOutput is the axis or stepper output to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_STEP_OUTPUT1 through NIMC_STEP_OUTPUT30. On motion controllers that support fewer than thirty axes, reading non-existent axes or stepper outputs returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Stepper Outputs</u> for axis and stepper output resource IDs.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns a number based on steps generated by a stepper axis or stepper output resource. For open-loop stepper axes, this function returns exactly the same value as the <u>Read Position</u> function.

For closed-loop stepper axes, this function returns the actual number of steps generated while the Read Position function returns the feedback position converted from counts to steps. The number of steps generated includes extra steps added during any pull-in move required to reach the target position.

To reset the value returned by this function, use <u>Reset Position</u>. Reset Position sets the **steps** parameter to the reset position value.

Closed-Loop Stepper Example

Suppose you execute a move from position 0 to position 100, and 10 additional steps are generated for the pull-in move. Read Position returns 100, but Read Steps Generated returns 110. If you execute another move to position 50, Read Position returns 50, and Read Steps Generated returns 60.

flex_read_steps_gen_rtn

Device Compatibility

Read Steps Generated Return

Usage

status = flex_read_steps_gen_rtn(u8 boardID, u8 axisOrStepperOutput, i32*
steps);

Purpose

Reads the number of steps generated by a stepper output.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrStepperOutput	u8	axis or stepper output to read
steps	i32*	is a value based on actual steps generated, as opposed to feedback position.

Parameter Discussion

axisOrStepperOutput is the axis or stepper output to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_STEP_OUTPUT1 through NIMC_STEP_OUTPUT30. On motion controllers that support fewer than thirty axes, reading non-existent axes or stepper outputs returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Stepper Outputs</u> for axis and stepper output resource IDs.

steps is the number of steps generated because the stepper axis last had its position reset.

Using This Function

This function returns a number based on steps generated by a stepper axis or stepper output resource. For open-loop stepper axes, this function returns exactly the same value as the <u>Read Position</u> function.

For closed-loop stepper axes, this function returns the actual number of steps generated while the Read Position function returns the feedback position converted from counts to steps. The number of steps generated includes extra steps added during any pull-in move required to reach the target position.

To reset the value returned by this function, use <u>Reset Position</u>. Reset Position sets the **steps** parameter to the reset position value.

Closed-Loop Stepper Example

Suppose you execute a move from position 0 to position 100, and 10 additional steps are generated for the pull-in move. Read Position returns 100, but Read Steps Generated returns 110. If you execute another move to position 50, Read Position returns 50, and Read Steps Generated returns 60.

flex_read_target_pos

Device Compatibility

Read Target Position

Usage

status = flex_read_target_pos(u8 boardID, u8 axis, u8 returnVector);

Purpose

Reads the destination position of the current motion trajectory.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to read
returnVector	u8	destination for the return data

Parameter Discussion

axis is the axis to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns the destination position of the motion trajectory currently in process. If the axis is stopped, it returns the target position of last trajectory completed.

Loading a new target position with <u>Load Target Position</u> does not affect the value read until a start motion occurs. When calling <u>Blend Motion</u>, the value read does not reflect the last target position loaded until after the blend occurs.

This function differs from the <u>Read Position</u> function in that it returns the commanded target (destination) position rather than the actual feedback position.

flex_read_target_pos_rtn

Device Compatibility

Read Target Position Return

Usage

status = flex_read_target_pos_rtn(u8 boardID, u8 axis, i32* targetPosition);

Purpose

Reads the destination position of the current motion trajectory.

Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to read
targetPosition	i32*	destination position of the current motion trajectory in counts or steps

Parameter Discussion

axis is the axis to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

targetPosition is the destination position of the current motion trajectory in counts (servo axes) or steps (stepper axes).

Using This Function

This function returns the destination position of the motion trajectory currently in process. If the axis is stopped, it returns the target position of last trajectory completed.

This function differs from the <u>Read Position</u> function in that it returns the commanded target (destination) position rather than the actual feedback position.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

 Loading a new target position with Load Target Position does not affect the value read until a start motion occurs. When calling <u>Blend Motion</u>, the value read does not reflect the last target position loaded until after the blend occurs.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- The NI SoftMotion Controller returns the target position loaded regardless of whether the <u>Start Motion</u> function has been called.
- The NI SoftMotion Controller returns the target position of individual axes, but does not return the target position loaded by the Load Vector Space Position function, even if the axis is configured as part of the vector space.

The NI SoftMotion Controller does not support reading the target position of a vector space.

flex_read_trajectory_data

Device Compatibility

Read Trajectory Data

Usage

status = flex_read_trajectory_data(u8 boardID, u8 returnVector);
Purpose

Reads a sample of acquired data from the samples buffer.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
returnVector	u8	destination for the return data

Parameter Discussion

returnVector contains position and velocity. Position is returned in onboard variable n in counts (steps), and velocity is returned in onboard variable n+1 in counts/s (steps/s).

Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

The Read Trajectory Data function is used to read back a single sample of acquired data from the onboard sample buffer. The number of samples, the time between samples and the size of each sample is set when you execute the <u>Acquire Trajectory Data</u> function. The sample buffer operates first-in-first-out (FIFO), so multiple calls to this function return samples in their correct time sequence.

While it is possible to read the sample buffer while samples are still being acquired, you must wait enough time between calls to this function to avoid emptying the buffer.



Note Attempting to read an empty sample buffer generates an error.

The Acquire Trajectory Data and Read Trajectory Data functions are used to acquire and read back time-sampled position and velocity data for analysis and display. These functions implement a digital oscilloscope that is useful during system set up, PID tuning, and general motion with data acquisition synchronization.

Example

The Acquire Trajectory Data function is executed with axes 1, 2, and 4 selected. Each call to the Read Trajectory Data function returns one sample with an array size of six and the following data in the array:

returnData[] = {Axis 1 position, Axis 1 velocity, Axis 2 position, Axis 2 velocity Axis 4 position, Axis 4 velocity}

Device Compatibility

Read Trajectory Data Return

Usage

status = flex_read_trajectory_data_rtn(u8 boardID, i32* returnData);

Purpose

Reads a sample of acquired data from the samples buffer.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
returnData	[i32]*	Array of position and velocity data for selected axes

Parameter Discussion

returnData contains a **single sample** of data as an array of position and velocity data for the selected axes. The size of the **returnData** array depends upon the number of axes selected with the Acquire Trajectory Data function.

For each axis selected, this function returns two array elements: position is returned in onboard variable n in counts (steps), and velocity is returned in onboard variable n+1 in counts/s (steps/s). The maximum array size is 8 when all 4 axes are selected.

Using This Function

The Read Trajectory Data function is used to read back a **single sample** of acquired data from the onboard sample buffer. The number of samples, the time between samples and the size of each sample is set when you execute the <u>Acquire Trajectory Data</u> function. The sample buffer operates first-in-first-out (FIFO), so multiple calls to this function return samples in their correct time sequence.

While it is possible to read the sample buffer while samples are still being acquired, you must wait enough time between calls to this function to avoid emptying the buffer.



Note Attempting to read an empty sample buffer generates an error.

The Acquire Trajectory Data and Read Trajectory Data functions are used to acquire and read back time-sampled position and velocity data for analysis and display. These functions implement a digital oscilloscope that is useful during system set up, PID tuning, and general motion with data acquisition synchronization.

Example

The Acquire Trajectory Data function is executed with axes 1, 2, and 4 selected. Each call to the Read Trajectory Data function returns one sample with an array size of six and the following data in the array:

returnData[] = {Axis 1 position, Axis 1 velocity, Axis 2 position, Axis 2 velocity Axis 4 position, Axis 4 velocity}

Device Compatibility

Read Trajectory Status

Usage

status = flex_read_trajectory_status(u8 boardID, u8 axisOrVectorSpace, u16 statusType, u8 returnVector);

Purpose

Reads the selected motion trajectory status of all axes or vector spaces.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	selects axis control or vector space control for move complete, or axis control for all other types
statusType	u16	status selector
returnVector	u8	destination for the return data

Parameter Discussion

axisOrVectorSpace can select multiple axes (NIMC_AXIS_CTRL) or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

statusType is the selector for the type of trajectory status to read.

statusType Constant	statusType Value	axisOrVectorSpa
NIMC_RUN_STOP_STATUS	0	NIMC_AXIS_CTR
NIMC_MOTOR_OFF_STATUS	1	NIMC_AXIS_CTR
NIMC_VELOCITY_THRESHOLD_STATUS	2	NIMC_AXIS_CTR
NIMC_MOVE_COMPLETE_STATUS	3	NIMC_AXIS_CTR NIMC_VECTOR_!

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns a multi-axis status bitmap of the status type selected. You can select one of the following three trajectory status types:

NIMC_RUN_STOP_STATUS Run/Stop status is updated based on the filtered velocity and run/stop criteria set by Load Run/Stop Threshold. The status is set when the axis is travelling fast enough to be considered running. The axis is considered to be running when the change in position per sample period exceeds the Run/Stop threshold set with the Load Run/Stop Threshold function.

NIMC_MOTOR_OFF_STATUS

A motor can be Off for two reasons. Either a kill stop was executed or the following error trip point was exceeded. A Motor Off condition also means that an enabled inhibit output is active. Refer to the <u>Configure Inhibit</u> <u>Outputs</u> function for more information.

NIMC_VELOCITY_THRESHOLD_STATUS

The Velocity Threshold status indicates if the axis velocity is above (True) or below (False) the programmed velocity threshold. Refer to the <u>Load</u> <u>Velocity Threshold</u> function for information about setting and using a velocity threshold.

NIMC_MOVE_COMPLETE_STATUS

The Move Complete status indicates if an axis or vector space is in the move complete state, which is the default when an axis or vector space is idle. While a move is in progress, the move complete status is false. For a move to be complete on a vector space, the move complete status must be true on all axes in the vector space.

During a vector space move, if one axis in a vector space trips out on a following error, that axis is killed, and the move complete status remains false. The other axes in the vector space decelerate to a stop, and the move complete status is true. For the vector space as a whole, the move complete status is false, because the move did not complete properly.

Use the <u>Configure Move Complete Criteria</u> function to change the conditions that cause a move to be evaluated as complete. For example, by changing the move complete criteria to be profile complete (default) OR motor off, the previous situation would result in a true move complete status when one of the axes in the vector space exceeded the programmed following error limit.

Note You can get all four trajectory statuses for a single axis by calling the <u>Read per Axis Status</u> function.

Example

To get Motor Off status, call the Read Trajectory Status function with **axisOrVectorSpace** = NIMC_AXIS_CTRL and **statusType** = NIMC_MOTOR_OFF_STATUS. Assume the returned **status** = 0x0062. This corresponds to the following bitmap:

D15	••••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
XXX	••••	XXX	XXX	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	XXX
0		0	0	0	0	0	0	0	1	1	0	0

Axes 1 and 4 are Off.

For your programming convenience, two utility functions—<u>Check Move</u> <u>Complete Status</u> and <u>Wait for Move Complete</u>—are provided, which allow you to specify an axis, vector space, group of axes, or group of vector spaces, and find out if a move is complete, or wait until a move is complete. These functions return a simple true/false value indicating if a move is complete.



flex_read_trajectory_status_rtn Device Compatibility

Read Trajectory Status Return

Usage

status = flex_read_trajectory_status_rtn(u8 boardID, u8 axisOrVectorSpace, u16 statusType, u16* status);

Purpose

Reads the selected motion trajectory status of all axes or vector spaces.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	selects axis control or vector space control for move complete, or axis control for all other types
statusType	u16	status selector
status	u16*	bitmap of selected status for all axes

Parameter Discussion

axisOrVectorSpace can select multiple axes (NIMC_AXIS_CTRL) or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

status is the bitmap of multi-axis or vector space status.

For **axisOrVectorSpace** = NIMC_AXIS_CTRL:

D15	•••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis		Axis	0									
15		10	9	8	7	6	5	4	3	2	1	

For **axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL:

D15		D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS 15	••••	VS 10	VS 9	VS 8	VS 7	VS 6	VS 5	VS 4	VS 3	VS 2	VS 1	0

D1 through D15 (axes) or D1 through D15 (vector spaces):

For NIMC_RUN_STOP_STATUS:

- 1 = Axis running
- 0 = Axis stopped

For NIMC_MOTOR_OFF_STATUS:

- 1 = Axis off
- 0 = Axis on

For NIMC_VELOCITY_THRESHOLD_STATUS:

- 1 = Velocity above threshold
- 0 = Velocity below threshold

For NIMC_MOVE_COMPLETE_STATUS:

- 1 = Move complete
- 0 = Move not complete

Using This Function

This function returns a multi-axis status bitmap of the status type selected. You can select one of the following three trajectory status types:

NIMC_RUN_STOP_STATUS

Run/Stop status is updated based on the filtered velocity and run/stop criteria set by Load Run/Stop Threshold. The status is set when the axis is travelling fast enough to be considered running. The axis is considered to be running when the change in position per sample period exceeds the Run/Stop threshold set with the Load Run/Stop Threshold function.

NIMC_MOTOR_OFF_STATUS

A motor can be Off for two reasons. Either a kill stop was executed or the following error trip point was exceeded. A Motor Off condition also means that an enabled inhibit output is active. Refer to the <u>Configure Inhibit</u> <u>Outputs</u> function for more information.

NIMC_VELOCITY_THRESHOLD_STATUS

The Velocity Threshold status indicates if the axis velocity is above (True) or below (False) the programmed velocity threshold. Refer to the <u>Load</u> <u>Velocity Threshold</u> function for information about setting and using a velocity threshold.

NIMC_MOVE_COMPLETE_STATUS

The Move Complete status indicates if an axis or vector space is in the move complete state, which is the default when an axis or vector space is idle. While a move is in progress, the move complete status is false. For a move to be complete on a vector space, the move complete status must be true on all axes in the vector space.

During a vector space move, if one axis in a vector space trips out on a following error, that axis is killed, and the move complete status remains false. The other axes in the vector space decelerate to a stop, and the move complete status is true. For the vector space as a whole, the move complete status is false, because the move did not complete properly.

Use the <u>Configure Move Complete Criteria</u> function to change the conditions that cause a move to be evaluated as complete. For example, by changing the move complete criteria to be profile complete (default) OR motor off, the previous situation would result in a true move complete status when one of the axes in the vector space exceeded the programmed following error limit.

Note You can get all four trajectory statuses for a single axis by calling the <u>Read per Axis Status</u> function.

Example

To get Motor Off status, call the Read Trajectory Status function with **axisOrVectorSpace** = NIMC_AXIS_CTRL and **statusType** = NIMC_MOTOR_OFF_STATUS. Assume the returned **status** = 0x0062. This corresponds to the following bitmap:

D15		D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis 15	••••	Axis 10	Axis 9	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	0
0		0	0	0	0	1	1	0	0	0	1	0

For your programming convenience, two utility functions—<u>Check Move</u> <u>Complete Status</u> and <u>Wait for Move Complete</u>—are provided, which allow you to specify an axis, vector space, group of axes, or group of vector spaces, and find out if a move is complete, or wait until a move is complete. These functions return a simple true/false value indicating if a move is complete.



Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following includes considerations you must make when you are using this function with the NI SoftMotion Controller:

To read the trajectory status on axes sixteen through thirty, use the <u>Read</u> <u>per Axis Status</u> function.

Start & Stop Motion

You can use Start & Stop Motion functions to start, blend, and stop motion.

You can execute all of the NI-Motion start and stop functions on an individual axis, simultaneously on multiple axes, on a vector space, or simultaneously on multiple vector spaces. These functions give complete control over the state of the motors in the system and with the addition of the <u>Find Reference</u> functions, are the only NI-Motion functions that can actually initiate motion.

Start Motion

Blend Motion

Stop Motion

flex_blend

Device Compatibility

Blend Motion

Usage

status = flex_blend(u8 boardID, u8 axisOrVectorSpace, u16 axisOrVSMap);

Purpose

Blends motion on a single axis, single vector space, multiple axes, or multiple vector spaces.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space selector
axisOrVSMap	u16	bitmap of axes or vector spaces to blend

Parameter Discussion

axisOrVectorSpace can select an axis (NIMC_AXIS1 through NIMC_AXIS30), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously blending multiple axes or vector spaces, the **axisOrVSMap** parameter indicates which axes or vector spaces are involved. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

axisOrVSMap is the bitmap of axes or vector spaces to blend. It is only required when you select multiple axes or vector spaces with the **axisOrVectorSpace** parameter. NI-Motion ignores additional axes and vector spaces if you select non-existent axes.

When blending axes (axisOrVectorSpace = NIMC_AXIS_CTRL):

D15	•••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis 15	•••	Axis 10	Axis 9	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	0

For D1 through D15:

1 = Blend axis

0 = Do not blend axis

When blending vector spaces (**axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL):

D15		D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS 15	••••	VS 10	VS 9	VS 8	VS 7	VS 6	VS 5	VS 4	VS 3	VS 2	VS 1	0

For D1 through D15:

1 = Blend vector space

0 = Do not blend vector space

To blend a single axis or vector space, set the **axisOrVectorSpace** selector to the specified axis or vector space. The **axisOrVSMap** bitmap is ignored.

To blend multiple axes, the axisOrVectorSpace selector is set to

NIMC_AXIS_CTRL and the **axisOrVSMap** bitmap defines the axes to blend. Similarly, to blend multiple vector spaces, the **axisOrVectorSpace** selector is set to NIMC_VECTOR_SPACE_CTRL and the **axisOrVSMap** bitmap defines the vector spaces to blend.



Note It is not possible to combine the blend of an axis and the blend of a vector space in a single use of this function. To accomplish this, create a single axis vector space and then execute a multi-vector space blend.

Using This Function

The primary difference between a <u>Start Motion</u> function and a Blend Motion function is that the Start Motion is immediate and preemptive, while Blend Motion waits and starts the next move upon the completion of the previous move.

This function is used to blend motion profiles on axes or vector spaces, either simultaneously or individually. A blend is similar to a normal start and has the same requirements for valid trajectory parameters as the <u>Start Motion</u> function. The blended move uses the most recently loaded values of acceleration, velocity, target position, s-curve, operation mode and so on to generate the motion profile.



Note If a stepper axis is in a killed state (not energized), halt the axis using the <u>Stop Motion</u> function, with **stopType** set to NIMC_HALT_STOP, before you execute a Start Motion or Blend Motion function. After you halt the axis, you might need to wait before executing a Start Motion or Blend Motion function, so that the stepper drive comes out of reset state. If the stepper drive does not come out of reset state before you execute the function, the stepper axis might lose some steps during acceleration. To determine if you need to wait before executing the function, refer to the stepper drive documentation or vendor.

Blend starting smoothly blends two move segments on an axis, axes, or vector space(s). There are three types of blends, controlled by the blend factor:

- Blend moves by superimposing the deceleration profile of the previous move with the acceleration profile of the next move (blend factor = -1).
- Blend moves by starting the next move at the exact point when the previous move has stopped (blend factor = 0).
- Start the next move after a programmed delay time between the end of the previous move and the start of the next move (blend factor > 0 ms).

Refer to the <u>Load Blend Factor</u> function for more information about how blend factor controls the blending of motion profiles.

Caution For sequencing multiple moves with blends, NI-Motion

must complete one blend before parameters for the next move are loaded. Refer to the <u>Read Blend Status</u> function for more information about blend sequencing.

If motion on any axis involved in a blend is illegal due to a limit or other error condition, the entire Blend Motion function is not executed and a modal error is generated. None of the axes are affected and the move(s) in process completes normally and stops.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:



Note You cannot execute this function on multiple axes using the NIMC_AXIS_EX_CTRL resource parameter with axes sixteen through thirty.

flex_start

Device Compatibility
Start Motion

Usage

status = flex_start(u8 boardID, u8 axisOrVectorSpace, u16 axisOrVSMap);

Purpose

Starts motion on a single axis, single vector space, multiple axes, or multiple vector spaces.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space selector
axisOrVSMap	u16	bitmap of axes or vector spaces to start

Parameter Discussion

axisOrVectorSpace can select an axis (NIMC_AXIS1 through NIMC_AXIS30), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously starting multiple axes or vector spaces, the **axisOrVSMap** parameter indicates which axes or vector spaces are involved. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

axisOrVSMap is the bitmap of axes or vector spaces to start. It is only required when multiple axes or vector spaces are selected with the **axisOrVectorSpace** parameter. NI-Motion ignores additional axes and vector spaces if you select non-existent axes.

When starting axes (**axisOrVectorSpace** = NIMC_AXIS_CTRL):

D15	••••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis		Axis	0									
15		10	9	8	7	6	5	4	3	2	1	

For D1 through D15:

1 = Start axis

0 = Do not start axis

When starting vector spaces (**axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL):

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS 15	 VS 10	VS 9	VS 8	VS 7	VS 6	VS 5	VS 4	VS 3	VS 2	VS 1	0

For D1 through D15:

1 = Start vector space

0 = Do not start vector space

To start a single axis or vector space, set the **axisOrVectorSpace** selector to the specified axis or vector space. The **axisOrVSMap** bitmap is ignored.

To start multiple axes, the axisOrVectorSpace selector is set to

NIMC_AXIS_CTRL and the **axisOrVSMap** bitmap defines the axes to start. Similarly, to start multiple vector spaces, the **axisOrVectorSpace** selector is set to NIMC_VECTOR_SPACE_CTRL and the **axisOrVSMap** bitmap defines the vector spaces to start.



Note It is not possible to combine the start of an axis and the start of a vector space in a single use of this function. To accomplish this, create a single axis vector space and then execute a multivector space start.

Using This Function

This function is used to start a motion profile on axes or vector spaces, either simultaneously or individually. A start is preemptive and uses the most recently loaded values of acceleration, velocity, target position, s-curve, operation mode, and so on to generate the motion profile.

Note If a stepper axis is in a killed state (not energized), halt the axis using the <u>Stop Motion</u> function, with **stopType** set to NIMC_HALT_STOP, before you execute a Start Motion or <u>Blend</u> Motion function. After you halt the axis, you might need to wait before executing a Start Motion or Blend Motion function, so that the stepper drive comes out of reset state. This is especially important for stepper axes configured for P-command mode to avoid position error between the controller and the drive. If the stepper drive does not come out of reset state before you execute the function, the stepper axis might lose some steps during acceleration. Refer to the stepper drive documentation or vendor to determine if you need to wait before executing the function.

You also can use the Start Motion function to update trajectory parameters to a move that is already in process. Trajectory parameters loaded after the start take effect immediately upon the next start without requiring the motion to come to a stop. You can use this feature for velocity profiling and other continuous motion applications. The execution of a preemptive start depends on the parameters entered, the type of move, and the controller. Refer to the following table for more information.

Move Type	Controller Type	Same Direction	Opposite Direction
Single Axis Move	73 <i>xx</i> Controller	Move starts immediately ¹	Motion decelerates to a stop, then starts ²
	NI SoftMotion Controller	Move starts immediately	Motion decelerates to a stop, then starts ²
Vector Space Move	73 <i>xx</i> Controller	Motion decelerates to a stop, then starts ²	Motion decelerates to a stop, then starts ²

NI SoftMotion Controller	Move starts immediately	Motion decelerates to a stop, then starts ²
--------------------------------	----------------------------	--

¹ In most cases, the move starts immediately. However, if the new target position is closer to the current position than the previous target position, the move decelerates to a stop, then starts. Refer to the figures below for more information.

 2 The time required to start the next move depends on the deceleration time. If the deceleration time is very slow, the next move may take longer than expected to start.

The first image demonstrates the case where Start Motion is called with a new target position on the same side of the original target position as the current position. In this case, motion decelerates to a stop, then starts again. The second image demonstrates the case where Start Motion is called with a new target position on the opposite side of the original target position as the current position. In this case, the move continues to the new target position without stopping.



Motion starts on properly configured and enabled axes. If motion on any axis involved in a start is illegal due to a limit or other error condition, the entire Start Motion function is not executed and a modal error is generated. None of the axes are started or updated.

Example 1

To execute a multi-axis start on axes 2 and 4, call the Start Motion

function with the following parameters:

axisOrVectorSpace = NIMC_AXIS_CTRL
axisOrVSMap = 0x14

The **axisOrVSMap** value of 0x14 corresponds to the following bitmap:

D15		D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis 15		Axis 10	Axis 9	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	0
0	•••	0	0	0	0	0	0	1	0	1	0	0

Example 2

To start motion on vector space 2, call the Start Motion function with the following parameters:

axisOrVectorSpace = NIMC_VECTOR_SPACE2
axisOrVSMap = Don't care

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following includes considerations you must make when you are using this function with the NI SoftMotion Controller:

You cannot execute this function on multiple axes using the NIMC_AXIS_EX_CTRL resource parameter with axes sixteen through thirty.

flex_stop_motion

Device Compatibility

Stop Motion

Usage

status = flex_stop_motion(u8 boardID, u8 axisOrVectorSpace, u16 stopType, u16 axisOrVSMap);

Purpose

Stops motion on a single axis, single vector space, multiple axes, or multiple vector spaces. Three types of stops can be executed: decelerate to stop, halt stop, and kill stop.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space selector
stopType	u16	type of stop to execute
axisOrVSMap	u16	bitmap of axes or vector spaces to stop

Parameter Discussion

axisOrVectorSpace can select an axis (NIMC_AXIS1 through NIMC_AXIS30), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously stopping multiple axes or vector spaces, the **axisOrVSMap** parameter indicates which axes or vector spaces are involved. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

stopType is the type of stop to execute.

stopType Constant	stopType Value	Action
NIMC_DECEL_STOP	0	decelerate smoothly to a stop
NIMC_HALT_STOP	1	immediate, full torque/stop
NIMC_KILL_STOP	2	zero the command and activate the inhibit/output

axisOrVSMap is the bitmap of axes or vector spaces to stop. It is only required when multiple axes or vector spaces are selected with the **axisOrVectorSpace** parameter. NI-Motion ignores additional axes and vector spaces if you select non-existent axes.

When stopping axes (axisOrVectorSpace = NIMC_AXIS_CTRL):

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis	 Axis	Axis	0								
15	10	9	8	7	6	5	4	3	2	1	

For D1 through D15:

1 =Stop axis

0 = Do not stop axis

When stopping vector spaces (**axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL):

D15		D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS 15	••••	VS 10	VS 9	VS 8	VS 7	VS 6	VS 5	VS 4	VS 3	VS 2	VS 1	0

For D1 through D15:

- 1 = Stop vector space
- 0 = Do not stop vector space

To stop a single axis or vector space, set the **axisOrVectorSpace** selector to the specified axis or vector space. The **axisOrVSMap** bitmap is ignored.

To stop multiple axes, the **axisOrVectorSpace** selector is set to NIMC_AXIS_CTRL and the **axisOrVSMap** bitmap defines the axes to stop. Similarly, to stop multiple vector spaces, the **axisOrVectorSpace** selector is set to NIMC_VECTOR_SPACE_CTRL and the **axisOrVSMap** bitmap defines the vector spaces to stop.



Note It is not possible to combine the stop of an axis and the stop of a vector space in a single use of this function. To accomplish this, create a single axis vector space and then execute a multivector space stop.

Using This Function

This function is used to stop a motion profile on axes or vector spaces, either simultaneously or individually. You can execute three different types of stops with the Stop Motion function: decel stop, halt stop, and kill stop.

Decel Stop

When a decel stop is executed (NIMC_DECEL_STOP), the axis, axes, or vector space(s) immediately begin to follow the deceleration portion of their trajectory profile as controlled by previously loaded deceleration and s-curve parameters. The actual stopped position is therefore dependent upon this deceleration trajectory.

Halt Stop

In contrast, a halt stop (NIMC_HALT_STOP) has the following behavior depending on the state of the axis:

- If the axis is not moving and not energized, the axis is energized.
- If the axis is not moving and energized, calling a halt stop has no effect.
- If the axis is moving:
 - On servo axes, full torque is applied to stop the motor(s) as quickly as possible.
 - On stepper axes, the step pulses are ceased as soon as the command is received by the motion controller. However, it is possible that there may be a small amount of following error after halting the move. This following error will be compensated for with a pull-in move during the next move.

In both cases, NI-Motion attempts to stop the motor(s) with a near infinite rate of deceleration. There is no trajectory profile and motion is not controlled by previously loaded deceleration and S-curve parameters.

 $\overline{\mathbb{Z}}$

Note For stepper axes configured for p-command mode, ensure that the axis is not moving before halting an axis that was previously killed to avoid position error between the controller and the drive. In this case, a call to halt stop while the axis is moving resets your reference position to an undefined location, and you should perform a Find Home reference move before executing

another move.

Kill Stop

On servo axes, a kill stop (NIMC KILL STOP) disables the control loop and zeros the output DAC, allowing frictional forces alone to stop the motion. On stepper axes, a kill stop ceases the stepper pulse generation as soon as the command is received by the motion controller. On both axis types, there is no trajectory profile during a kill stop. If enabled, the inhibit output is activated to inhibit (disable) the servo amplifier or stepper driver. You can enable the inhibit outputs and set their polarity as active high or active low with the Configure Inhibit Outputs function.

Caution When an axis is killed, the motor is allowed to freewheel, and could possibly move if external forces are acting on it. If the axis moves into an enabled limit switch, the axis is energized and held in position. If you do not want the axis to become energized under any circumstances, you must disable the axis after killing it.

The Stop Motion function may or may not affect the motion of other axes that are not explicitly referenced in the function. If a slave axis is killed, master-slave gearing is automatically disabled.

Example 1

To execute a multi-axis kill stop on axes 1 through 4, call the Stop Motion function with the following parameters:

```
axisOrVectorSpace = NIMC_AXIS_CTRL
stopType = NIMC KILL STOP
axisOrVSMap = 0x1E
```

The **axisOrVSMap** value of 0x1E corresponds to the following bitmap:

D15		D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis 15	••••	Axis 10	Axis 9	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	0
0		0	0	0	0	0	0	1	1	1	1	0

Example 2

To decelerate stop motion on vector space 1, call the Stop Motion function with the following parameters:

axisOrVectorSpace = NIMC VECTOR SPACE1

stopType = NIMC_DECEL_STOP
axisOrVSMap = Don't care

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

• If an axis that is part of a vector space is individually killed, the other axes in the vector space are decel stopped. If a program attempts to start axes that have been manually stopped by the host computer, it is overruled and put into the paused state.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- If an axis that is part of a vector space is individually killed, the other axes in the vector space are also killed.
- You can execute this function only on an axis that is enabled.
- You cannot execute this function on multiple axes using the NIMC_AXIS_EX_CTRL resource parameter with axes sixteen through thirty.

Motion I/O

You can use Motion I/O functions to set up and control the motion I/O features of the NI motion controller. Motion I/O functions include functions to set active state and enable limit and home inputs, high-speed capture inputs and inhibit outputs, functions to configure and control breakpoint outputs, and functions to read the status of all the motion I/O signals, high-speed captured position, and software limit status.

All of the dedicated motion I/O also can functions as general-purpose digital I/O when they are not being used for their motion specific features. You can set and reset outputs, you can read inputs at any time, and you can change their active state as required.

Motion I/O functions include limits and other basic Motion I/O functions, <u>Breakpoint functions</u>, and <u>High-Speed Capture functions</u>.

Set Limit Input Polarity Enable Axis Limit

Load Software Limit Positions

Set Home Input Polarity

Enable Home Inputs

Set Inhibit MOMO

Configure Inhibit Output

Read Axis Limit Status

Read Home Input Status

Device Compatibility

Configure Inhibit Output

Usage

status = flex_config_inhibit_output(u8 boardID, u8 axis, u16 enable, u16
polarity, u16 driveMode)

Purpose

Configures polarity and enables the per-axis inhibit outputs.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u16	axis to configure
enable	u16	enables/disables inhibit output on the selected axis
polarity	u16	specifies either active low/active open or active high/active closed
driveMode	u16	specifies either Open Collector or Totem Pole mode

Parameter Discussion

axis is the axis to configure. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

enable is the inhibit output to enable. An enabled inhibit monitors the motor off (killed) state of the axis.

polarity is the stepper output configured as active low/active open or active high/active closed.

Constant	Value	Description
NIMC_ACTIVE_HIGH	0	Logical True or On
NIMC_ACTIVE_LOW	1	Logical False or Off

driveMode specifies either **Open Collector** or **Totem Pole** mode. **driveMode** is available only on the NI 7350 and is not configurable on NI 7330/40/90 motion controllers.

Constant	Value	Description
NIMC_OPEN_COLLECTOR	0	Configures output to be Open Collector
NIMC_TOTEM_POLE	1	Configures output to be Totem Pole

Using This Function

This function enables/disables and sets the polarity of the axis inhibit outputs as active low or active high. When enabled, a per-axis inhibit output is linked to the motor off state of the corresponding axis. A killed axis (motor off) forces the corresponding inhibit output On. When the axis is active, the inhibit output is Off. Note that these On and Off states are logical states. The actual state depends on the polarity of the system.

Inhibit outputs are typically used to disable the servo amplifier or stepper drive for power saving, safety, or specific application reasons.

Note Killing a servo axis also zeros its DAC output. With torque block amplifiers, this means that the motor freewheels regardless of if the amplifier is disabled. With velocity block servo amplifiers or stepper drives, the motor does not freewheel unless the amplifier/drive is disabled with the inhibit output.

You also can use inhibit outputs as general-purpose outputs. Disabled inhibit outputs ignore the state of their corresponding axis and can be directly controlled through the <u>Set Inhibit Output MOMO</u> function.

You can configure the polarity of each inhibit output as active low or active high. When configured as active low, the output is active when there is a low signal on the output pin. Conversely, active high means that the output is active when there is a high signal on the output pin. The inhibit polarity is always in effect, regardless of if the inhibit is linked to its axis (enabled) or directly controlled through the Set Inhibit Output MOMO function.

For more information about **Open Collector** and **Totem Pole** drive modes, refer to the *NI 7350 Hardware User Manual*. Configuring **Drive Mode** is supported only on 7350 controllers and is not configurable on NI 7330/40/90 motion controllers.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following includes considerations you must make when you are using this function with a 73xx motion controller:

The PCI-7390 does not support the **polarity** parameter. To set the active state on a PCI-7390 use <u>Write Digital I/O Data</u>.

Device Compatibility

Enable Home Inputs

Usage

status = flex_enable_home_inputs(u8 boardID, u16 homeMap);

Purpose

Enables/disables the home inputs.



Parameters

Name	Туре	Description			
boardID	u8	assigned by Measurement & Automation Explorer (MAX)			
homeMap	u16	bitmap of home inputs to enable			

Parameter Discussion

homeMap is the bitmap of home inputs to enable.

D15	 D10	D9	D8	D7	D6	D5	D4
Home 15	 Home 10	Home 9	Home 8	Home 7	Home 6	Home 5	Home

For D1 through D8:

1 = Home input enabled

0 = Home input disabled (default)

Using This Function

This function enables/disables any combination of axis home inputs. An enabled home input causes a halt stop on the axis when the input becomes active. You can configure each home input to active low or active high with the <u>Set Home Input Polarity</u> function. You also can use a home input as a general-purpose input and read its status with the <u>Read</u> <u>Home Input Status</u> function.

Home inputs are an enhancement on the NI motion controller and are not required for basic motion control. You can operate all motion control functions without enabling or using the home inputs except the Find Reference function, which requires enabled limit and home inputs for operation when the find reference type is "home."

To configure the property of Find Reference Home, such as **Initial search direction** or **Home edge to stop on**, use the <u>Load Reference</u> <u>Parameter</u> function.

Note An active (and enabled) home input transition on an axis that is part of a vector space move causes that axis to halt stop and the other axes in the vector space to decelerate to a stop.

Example

To enable the home inputs for axes 2 and 4, call the Enable Home Inputs function with **homeMap** = 0x0014, which corresponds to the following bitmap:

D15	••••	D10	D9	D8	D7	D6	D5	D4
Home 15		Home 10	Home 9	Home 8	Home 7	Home 6	Home 5	Home
0		0	0	0	0	0	0	1

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

To enable home inputs on axes sixteen through thirty, use the <u>Write</u> <u>Motion I/O Data</u> function. flex_enable_axis_limit

Device Compatibility

Enable Axis Limits

Usage

status = flex_enable_axis_limit(u8 boardID, u16 limitType, u16 forwardLimitMap, u16 reverseLimitMap);
Purpose

Enables/disables either the forward and reverse limit inputs or the forward and reverse software position limits.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
limitType	u16	hardware/software limit selector
forwardLimitMap	u16	bitmap of forward limits to enable
reverseLimitMap	u16	bitmap of reverse limits to enable

Parameter Discussion

limitType selects the type of limit to enable, either the hardware limit switch inputs or the software position limits:

limitType Constant	limitType Value
NIMC_LIMIT_INPUTS	0
NIMC_SOFTWARE_LIMITS	1

forwardLimitMap is the bitmap of forward limits to enable (either inputs or software):

D15	 D10	D9	D8	D7	D6	D
Forward 15	 Forward 10	Forward 9	Forward 8	Forward 7	Forward 6	F٢

For D1 through D15:

- 1 = Forward limit enabled
- 0 = Forward limit disabled (default)

reverseLimitMap is the bitmap of reverse limits to enable (either inputs or software).

D15	 D10	D9	D8	D7	D6	Ľ
Reverse 15	 Reverse 10	Reverse 9	Reverse 8	Reverse 7	Reverse 6	F

For D1 through D15:

1 = Reverse limit enabled

0 = Reverse limit disabled (default)

Using This Function

This function enables/disables any combination of axis limits. You can enable the physical limit inputs (hardware) or the logical position limits (software) depending upon the **limitType** selected. You can enable or disable forward and reverse limits separately. To enable both software and hardware limits on an axis or axes, call this function twice.

The limit inputs are typically connected to end-of-travel limit switches or sensors. An enabled limit input causes a halt stop on the axis when the input becomes active. You can configure each limit input to active low or active high with the <u>Set Limit Input Polarity</u> function. Active limit inputs also prohibit attempts to start motion that would cause additional travel in the direction of the limit. You also can use limit inputs as general-purpose inputs and read their status with the <u>Read Axis Limit Status</u> function.

Note For the end-of-travel limits to function correctly, the forward limit switch or sensor must be located at the positive (count up) end of travel and the reverse limit at the negative (count down) end of travel.

Similarly, software limits are often used to restrict the range of travel further and avoid ever hitting the hardware limit switches. An enabled software limit causes the axis to smoothly decelerate to a stop when the limit position is reached or exceeded. To enable the software limits, you must first find the reference point using the <u>Find Reference</u> function.

Even when disabled, you can use the host computer to poll the software limits to warn of an out of range position. Refer to the <u>Load Software</u> <u>Limit Positions</u> and the <u>Read Limit Status</u> functions for information about loading and reading the forward and reverse software limits.

Hardware limit inputs and software position limits are enhancements on the NI motion controllers and are not required for basic motion control. You can operate all motion control functions without enabling or using these limits except the <u>Find Reference</u> function, which requires enabled limit and home inputs for operation. Find Reference does not require enabled software limits. Refer to the following functions for more information about enabling home inputs: <u>Set Home Input Polarity</u>, <u>Enable</u> <u>Home Inputs</u>, and <u>Read Home Input Status</u>.

Example

To enable the forward and reverse software limits on axes 1 and 3, and disable the forward and reverse software limits on the remaining axes, call the <u>Enable Limits</u> function with the following parameters:

limitType = NIMC_SOFTWARE_LIMITS

forwardLimitMap = 0x0A, which corresponds to the following bitmap:

D15	••••	D10	D9	D8	D7	D6	D
Forward 15		Forward 10	Forward 9	Forward 8	Forward 7	Forward 6	F
0		0	0	0	0	0	0

reverseLimitMap = 0x0A, which corresponds to the following bitmap:

D15	••••	D10	D9	D8	D7	D6	C
Reverse 15		Reverse 10	Reverse 9	Reverse 8	Reverse 7	Reverse 6	F
0	••••	0	0	0	0	0	(

This example affects only the software limits, because the **limitType** parameter is set to NIMC_SOFTWARE_LIMITS (1). To affect the hardware limits, set **limitType** to NIMC_HARDWARE_LIMITS (0).

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- An active (and enabled) limit input transition on an axis that is part of a vector space move causes that axis to halt stop and the other axes in the vector space to decelerate to a stop.
- If any axis in a vector space move exceeds an enabled software limit position, all axes in the vector space decelerate to a stop.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- An active (and enabled) limit on an axis that is part of a vector space causes all of the axes in the vector space to halt stop.
- To enable limits on axes sixteen through thirty, use the Write Motion I/O Data function.

flex_load_sw_lim_pos

Device Compatibility

Load Software Limit Positions

Usage

status = flex_load_sw_lim_pos(u8 boardID, u8 axis, i32 forwardLimit, i32 reverseLimit, u8 inputVector);

Purpose

Loads the forward and reverse software limit positions for an axis.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to control
forwardLimit	i32	forward software limit position in counts or steps
reverseLimit	i32	reverse software limit position in counts or steps
inputVector	u8	source of the data for this function

Parameter Discussion

axis is the axis to control. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

forwardLimit is the forward software limit position in counts (servo axes) or steps (stepper axes). Software limit positions can be anywhere within the 32-bit position range, $-(2^{31})$ to $+(2^{31}-1)$. The default value for the forward software limit is $+(2^{30}-1)$ counts (steps).

reverseLimit is the reverse software limit position in counts (servo axes) or steps (stepper axes). Software limit positions can be anywhere within the 32-bit position range, $-(2^{31})$ to $+2^{31}-1$). The default value for the reverse software limit is -2^{30} counts (steps).

Note forwardLimit cannot be less than reverseLimit.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function sets the forward and reverse position limit values for the selected axis. When enabled with the Enable Axis Limits function, a software limit causes the axis to smoothly decelerate to a stop when the limit position is reached or exceeded.

Even when disabled, you can poll the software limits by the host computer or use an onboard program to warn of an out of range position. Refer to the Read Axis Limit Status function for information about reading the software limit status.

The forward software limit is considered active if the current position is greater than or equal to the forward software limit position. The reverse software limit is considered active if the current position is less than or equal to the reverse software limit position.

Software limits are often used to restrict the range of travel and avoid hitting the hardware end-of-travel limit switches. For example, you can travel at a high velocity until hitting the software limit switch, and then move more slowly until hitting the hardware limit switch.



Caution After an axis has stopped due to encountering a software limit switch, you can still move further in the same direction if you command the axis to do so. This behavior is not possible with hardware limits, but is appropriate for software limits.

Read Drive Signal Status

Usage

status = flex_read_drive_signal_status(u8 boardID, u8 axis, u8 returnVector);

Purpose

Reads the status of the drive signal when an active signal is detected.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to configure
returnVector	u8	destination for the return data

Parameter Discussion

axis is the axis to configure. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

returnVector indicates the desired destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and do not return data (0). Refer to <u>Input and</u> <u>Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

Use this function to read the status of the drive signal when an active drive signal is detected. If In-Position is active, bit0 = 1. If Drive Fault is active, bit1 = 1.



Tip Use <u>Read Motion I/O Data</u> instead of this function for more options.

Device Compatibility

Read Drive Signal Status

Usage

status = flex_read_drive_signal_status_rtn(u8 boardID, u8 axis, u16*
driveStatus);

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to configure
driveStatus	u16*	contains the status of the drive signal

Parameter Discussion

axis is the axis to configure. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

driveStatus contains the status of the drive signal, where bit0 = In-Position and bit1 = Drive Fault.

Using This Function

Use this function to read the status of the drive signal when an active drive signal is detected. If In-Position is active, bit 0 = 1. If Drive Fault is active, bit 1 = 1.



Tip Use <u>Read Motion I/O Data</u> instead of this function for more options.

Read Home Input Status

Usage

status = flex_read_home_input_status(u8 boardID, u8 returnVector);

Purpose

Reads the instantaneous status of the home inputs.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
returnVector	u8	destination for the return data

Parameter Discussion

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns the logical state of the home inputs. You can execute this function at anytime to monitor the home inputs, regardless of if they are enabled. A home input enabled with the <u>Enable Home Inputs</u> function causes a halt stop on an axis when its home input becomes active (True). You also can use a home input as a general-purpose input and read its status with this function. The home input status during a <u>Find</u> <u>Reference</u> move is undefined.



Note This function returns undefined values for axes that are currently performing find reference moves.

To determine the home found status, use <u>Read Reference Status</u> and set the reference type to NIMC_HOME_FOUND.



Note This function reads the logical state (On or Off, True or False) of the home inputs. The polarity of the home inputs determines if an On state is active high or active low. Refer to the <u>Set Home Input Polarity</u> function for more information.

flex_configure_drive_signal

Device Compatibility

Configure Drive Signal

Usage

status = flex_configure_drive_signal(u8 boardID, u8 axis, u16 port, u16 pin, u16 mode, u16 polarity);

Purpose

Configures the drive signal.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axis	u8	axis to configure
port	u16	general-purpose I/O port (NIMC_IO_PORT1 to NIMC_IO_PORT8) or HSC software port (NIMC_HSC_PORT) to control
pin	u16	specific bit in the port (0 to 7)
mode	u16	mode that the pin will be reserved for
polarity	u16	polarity applied to the requested pin

Parameter Discussion

axis is the axis to configure. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

port is the general purpose I/O port (NIMC_IO_PORT1 to NIMC_IO_PORT8) or HSC software port (NIMC_HSC_PORT) that you are controlling with the drive signal. Refer to <u>73xx Controller General-</u><u>Purpose I/O Port IDs</u> for I/O port resource IDs and constants.

pin is the specific bit in the port (0 to 7) specified by port. For the HSC port, trigger 1 = pin 0, trigger 2 = pin 1, trigger 3 = pin 2, trigger 4 = pin 3, trigger 5 = pin 4, trigger 6 = pin 5, trigger 7 = pin 6, and trigger 8 = pin 7.

mode is the mode that the pin will be reserved for. Options for mode are NIMC_DRIVE_SIGNAL_CLEAR_ALL (0),

NIMC_IN_POSITION_MODE (1) or NIMC_DRIVE_FAULT_MODE (2). If a DIO or high-speed capture line has been reserved for a drive signal, I/O functionality is blocked for the reserved pin. You must use the <u>Read Drive</u> <u>Signal Status</u> function to read the status of this signal.

NIMC_DRIVE_SIGNAL_CLEAR_ALL (0) clears any reservations for In-Position or Drive Fault signals for the specified axis. Use this mode to unconfigure I/O lines and reset them to their default functionality.

polarity is the polarity applied to the requested pin. Valid options are NIMC_ACTIVE_HIGH (0) or NIMC_ACTIVE_LOW (1).

Using This Function

Use this function to configure and reserve I/O lines to receive signals from the drive. The following drive signals are available: In-Position and Drive Fault (Servo Alarm).



Tip Use <u>Configure Motion I/O Map</u> instead of this function for additional drive signal support.

Use the In-Position drive signal when the drive is controlling a servo motor and the input to the drive is step/direction. This signal tells the motion controller when the drive considers the motor to be at the commanded position.

When the In-Position drive signal is configured, the Move Complete status is linked to the state of the In-Position signal. The move is not considered complete unless the In-Position signal is active.

 $\overline{\mathbb{N}}$

Note Refer to <u>Write Trajectory Data</u> to add or remove this signal from the move complete criteria.

Use the Drive Fault drive signal to connect a servo alarm signal to the motion controller so that when an alarm or other drive fault occurs, the motion is stopped using a kill stop. A kill stop disables the control loop and zeroes the DAC so that frictional forces stop the motion.

flex_read_home_input_status_rtn

Device Compatibility
Read Home Input Status Return

Usage

status = flex_read_home_input_status_rtn(u8 boardID, u16* homeStatus);

Purpose

Reads the instantaneous status of the home inputs.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
homeStatus	u16*	bitmap of the logical state of the home inputs

Parameter Discussion

homeStatus is the bitmap of the logical state of the home inputs:

D15	 D10	D9	D8	D7	D6	D5	D4
Home 15	 Home 10	Home 9	Home 8	Home 7	Home 6	Home 5	Home

For D1 through D15:

1 = Home input True (On)

0 = Home input False (Off)

Using This Function

This function returns the logical state of the home inputs. You can execute this function at anytime to monitor the home inputs, regardless of if they are enabled. A home input enabled with the <u>Enable Home Inputs</u> function causes a halt stop on an axis when its home input becomes active (True). You also can use a home input as a general-purpose input and read its status with this function. The home input status during a <u>Find</u> <u>Reference</u> move is undefined.



Note This function returns undefined values for axes that are currently performing find reference moves.

To determine the home found status, use <u>Read Reference Status</u> and set the reference type to NIMC_HOME_FOUND.



Note This function reads the logical state (On or Off, True or False) of the home inputs. The polarity of the home inputs determines if an On state is active high or active low. Refer to the <u>Set Home Input Polarity</u> function for more information.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

To read the home input status on axes sixteen through thirty, use the <u>Read Motion I/O Data</u> function.

Device Compatibility

Read Axis Limit Status

Usage

status = flex_read_axis_limit_status(u8 boardID, u16 limitType, u8
returnVector);

Purpose

Reads the instantaneous state of either the hardware limit inputs or the software limits.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
limitType	u16	hardware/software limit selector
returnVector	u8	destination for the return data

Parameter Discussion

limitType selects the type of limit status to read, either the hardware limit switch status or the software position limit status.

limitType Constant	limitType Value
NIMC_LIMIT_INPUTS	0
NIMC_SOFTWARE_LIMITS	1

Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns either the hardware limit input status or the software position limit status, depending on the limit type selected. When **limitType** = NIMC_LIMIT_INPUTS (0), this function returns the logical state of the forward and reverse limit inputs.



Note The polarity of the limit inputs determines if an On state is active high or active low. Refer to the <u>Set Limit Input Polarity</u> function for more information.

Alternatively, when **limitType** = NIMC_SOFTWARE_LIMITS (1), this function returns the state of the forward and reverse software limits. A True (On) indicates that the forward or reverse limit position for the corresponding axis has been reached or exceeded. The limit input status during a <u>Find Reference</u> move is undefined.

You can read the status of the limit inputs and the software position limits at any time, regardless of if the limits are enabled. Enabled limits cause axes to stop when their state transitions True. Refer to the <u>Enable Axis</u> <u>Limits</u> function for more information.

flex_read_axis_limit_status_rtn Device Compatibility

Read Axis Limit Status Return

Usage

status = flex_read_axis_limit_status_rtn(u8 boardID, u16 limitType, u16*
forwardLimitStatus, u16* reverseLimitStatus);

Purpose

Reads the instantaneous state of either the hardware limit inputs or the software limits.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
limitType	u16	hardware/software limit selector
forwardLimitStatus	u16*	bitmap of forward limit status
reverseLimitStatus	u16*	bitmap of reverse limit status

Parameter Discussion

limitType selects the type of limit status to read, either the hardware limit switch status or the software position limit status.

limitType Constant	limitType Value		
NIMC_LIMIT_INPUTS	0		
NIMC_SOFTWARE_LIMITS	1		

forwardLimitStatus is the bitmap of forward limit status (either hardware or software):

D15	•••	D10	D9	D8	D7	D6	D
Forward 15		Forward 10	Forward 9	Forward 8	Forward 7	Forward 6	F

For D1 through D15:

1 = Forward limit True (On)

0 = Forward limit False (Off)

reverseLimitStatus is the bitmap of reverse limit status (either hardware or software).

D15	 D10	D9	D8	D7	D6	[
Reverse 15	 Reverse 10	Reverse 9	Reverse 8	Reverse 7	Reverse 6	F

For D1 through D15:

1 = Reverse limit True (On)

0 = Reverse limit False (Off)

Using This Function

This function returns either the hardware limit input status or the software position limit status, depending on the limit type selected. When **limitType** = NIMC_LIMIT_INPUTS (0), this function returns the logical state of the forward and reverse limit inputs.



Note The polarity of the limit inputs determines if an On state is active high or active low. Refer to the <u>Set Limit Input Polarity</u> function for more information.

Alternatively, when **limitType** = NIMC_SOFTWARE_LIMITS (1), this function returns the state of the forward and reverse software limits. A True (On) indicates that the forward or reverse limit position for the corresponding axis has been reached or exceeded. The limit input status during a <u>Find Reference</u> move is undefined.

You can read the status of the limit inputs and the software position limits at any time, regardless of if the limits are enabled. Enabled limits cause axes to stop when their state transitions True. Refer to the <u>Enable Axis</u> <u>Limits</u> function for more information.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

To read the hardware and software limit status on axes sixteen through thirty, use the <u>Read Motion I/O Data</u> function.

flex_set_home_polarity

Device Compatibility

Set Home Input Polarity

Usage

status = flex_set_home_polarity(u8 boardID, u16 homePolarityMap);

Purpose

Sets the polarity of the home inputs as either active low/active open or active high/active closed.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
homePolarityMap	u16	bitmap of active polarities for the home inputs

Parameter Discussion

homePolarityMap is the bitmap of active polarities for the home inputs:

D15	 D10	D9	D8	D7	D6	D5	D4
Home 15	 Home 10	Home 9	Home 8	Home 7	Home 6	Home 5	Home

For D1 through D15:

1 = Active low/Active open (default)

0 = Active high/Active closed

Using This Function

This function defines the active state for each home input as either active low/active open or active high/active closed.

When configured as active low, the input is active when there is a low signal on the input pin. Conversely, active high means that the input is active when there is a high signal on the input pin.

Configuring an active state of active open or active closed does not correspond to the level of the signal on the input pin. Instead, an active open state means that the input is active when current is not flowing through the optocoupled input. Conversely, an active closed state means that the input is active when current is flowing through the optocoupled input.

You can enable home inputs to cause halt stops when the input becomes active with the <u>Enable Home Inputs</u> function. You also can use a home input as a general-purpose input and read its status with the <u>Read Home Input Status</u> function.

To find the home input reference on a system, use <u>Find Reference</u> and configure the search type as "home".

Example

To set the polarity of the home inputs on axes 1, 3, and 4 as active low and the home inputs on the rest of the axes as active high, call the Set Home Input Polarity function with **homePolarityMap** = 0x001A, which corresponds to the following bitmap:

D15	 D10	D9	D8	D7	D6	D5	D4
Home 15	 Home 10	Home 9	Home 8	Home 7	Home 6	Home 5	Home
0	 0	0	0	0	0	0	1

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

To set home polarity on axes 16 through 30, call the <u>Write Motion I/O</u> <u>Data</u> function on each axis individually. flex_set_inhibit_output_momo Device Compatibility

Set Inhibit Output MOMO

Usage

status = flex_set_inhibit_output_momo(u8 boardID, u16 mustOn, u16 mustOff);

Purpose

Sets the inhibit outputs using the MustOn/MustOff protocol.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
mustOn	u16	bitmap of inhibit outputs to force on
mustOff	u16	bitmap of inhibit outputs to force off

Parameter Discussion

mustOn is the bitmap of inhibit outputs to force on:

D8	D7	D6	D5	D4	D3	D.
mustOn 8	mustOn 7	mustOn 6	mustOn 5	mustOn 4	mustOn 3	m

For D1 through D8:

1 = Inhibit output forced On

0 = Inhibit output unchanged (default)

mustOff is the bitmap of inhibit outputs to force off:

mustOff 8	mustOff 7	mustOff 6	mustOff 5	mustOff 4	mustOff 3	n
D8	D7	D6	D5	D4	D3	С

For D1 through D8:

1 = Inhibit output forced Off

0 = Inhibit output unchanged (default)

Using This Function

This function controls disabled inhibit outputs being used as generalpurpose outputs. You can directly set the inhibit outputs to a logical On or Off state.



Note This function has no effect on enabled inhibit outputs. These outputs are directly controlled by their corresponding axes.

Using the MustOn/MustOff protocol allows you to set or reset individual inhibit outputs without affecting the other inhibit outputs. This gives you tri-state control over each output: On, Off, or Unchanged. A one (1) in a bit location of the **MustOn** bitmap turns the inhibit On, while a one (1) in the corresponding location of the **MustOff** bitmap turns the inhibit Off. A zero (0) in either bitmap has no affect, so leaving both the **MustOn** and **MustOff** bits at zero is effectively a hold, and the state of the inhibit output is unchanged. If you set both the **MustOn** and **MustOff** bits to one (1), it is interpreted as a **MustOn** condition and the inhibit is turned On.

Must On –> Must Off	1	0
1	On	Off
0	On	Unchanged

Note This function sets the logical state of an inhibit output On or Off (True or False). The polarity of the inhibit outputs determine if an On state is active high or active low. Refer to the <u>Configure</u> <u>Inhibit Outputs</u> function for more information.

The Set Inhibit MOMO function allows individual control of the inhibit outputs without requiring a shadow value to remember the state of other outputs not being set or reset with the function.

Example

To turn inhibit output 1 On, output 4 off, and leave the other inhibit outputs unchanged, call the Set Inhibit MOMO function with the following parameters:

mustOn = 0x02, which corresponds to the following bitmap:

mustOn 8	mustOn 7	mustOn 6	mustOn 5	mustOn 4	mustOn 3
D8	D7	D6	D5	D4	D3

0	0	0	0	0	0

mustOff = 0x10, which corresponds to the following bitmap:

D8	D7	D6	D5	D4	D3
mustOff 8	mustOff 7	mustOff 6	mustOff 5	mustOff 4	mustOff
0	0	0	0	1	0

Device Compatibility

Set Limit Input Polarity

Usage

status = flex_set_limit_input_polarity(u8 boardID, u16 forwardPolarityMap, u16
reversePolarityMap);

Purpose

Sets the polarity of the forward and reverse limit inputs as either active low/active open or active high/active closed.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.
Parameters

Name	Туре	Description
boardID u8		assigned by Measurement & Automation Explorer (MAX)
forwardPolarityMap	u16	bitmap of active polarities for the forward limits
reversePolarityMap	u16	bitmap of active polarities for the reverse limits

Parameter Discussion

forwardPolarityMap is the bitmap of active polarities for the forward limit inputs:

D15		D10	D9	D8	D7	D6
Forward 15	••••	Forward 10	Forward 9	Forward 8	Forward 7	Forward

For D1 through D8:

- 1 = Active low/Active open (default)
- 0 = Active high/Active closed

reversePolarityMap is the bitmap of active polarities for the reverse limit inputs:

D15	••••	D10	D9	D8	D7	D6
Reverse 15	••••	Reverse 10	Reverse 9	Reverse 8	Reverse 7	Reverse

For D1 through D8:

- 1 = Active low/Active open (default)
- 0 = Active high/Active closed

Using This Function

This function defines the active state for each forward and reverse limit input as either active low/active open or active high/active closed.

When configured as active low, the input is active when there is a low signal on the input pin. Conversely, active high means that the input is active when there is a high signal on the input pin.

Configuring an active state of active open or active closed does not correspond to the level of the signal on the input pin. Instead, an active open state means that the input is active when current is not flowing through the optocoupled input. Conversely, an active closed state means that the input is active when current is flowing through the optocoupled input.

You can enable limit inputs to cause halt stops when the input becomes active with the <u>Enable Limits</u> function. You also can use a limit input as a general-purpose input and read its status with the <u>Read Axis Limit Status</u> function.

Example

To set the polarity of the forward and reverse limit inputs on axes 1, 2, and 3 as active low and the forward and reverse limit inputs on the remaining axes as active high, call the Set Limit Input Polarity function with the following parameters:

D15	••••	D10	D9	D8	D7	D6
Forward 15		Forward 10	Forward 9	Forward 8	Forward 7	Forward
0		0	0	0	0	0

forwardPolarityMap = 0x0E, which corresponds to the following bitmap:

reversePolarityMap = 0x0E, which corresponds to the following bitmap:

D15	••••	D10	D9	D8	D7	D6
Reverse 15	••••	Reverse 10	Reverse 9	Reverse 8	Reverse 7	Reverse
0	••••	0	0	0	0	0

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

To set the polarity of the forward and reverse limits on axes sixteen through thirty, call the <u>Write Motion I/O Data</u> function on each axis individually.

Breakpoint

Position breakpoints are an enhancement to the encoder decoders on the NI motion controller and are available when the encoders operate as axis feedback or as independent encoder resources. Breakpoint functionality is available on servo and closed-loop stepper axes. You can use Breakpoint functions to configure, enable, and read the position breakpoint status. Advanced functions, such as periodic breakpoint and buffered breakpoint can be used to achieve higher-frequency breakpoints. Refer to <u>Buffered Operations Functions</u> to configure buffers for buffered breakpoints. You also can load a breakpoint position modulus. Like all motion I/O, breakpoint outputs also can function as general-purpose outputs with the Set Breakpoint Output MOMO function.



Note All breakpoints can be affected by jitter in the motion system. For example, if you have breakpoint positions very close to each other, the jitter in the motion system could cause the position to change enough to reach the breakpoint when a breakpoint is not intended. Increase the value of the breakpoint window using <u>Set u32</u> to compensate for system jitter.

Refer to the following table for the maximum breakpoint rates for each type of breakpoint. Not all breakpoints are available on all controllers.

Breakpoint Type	Maximum Frequency
Single	150 Hz
Buffered, PID rate of 62.5 μs to 250 μs	2 kHz
Buffered, PID rate greater than 250 μs	1 kHz
Periodic	4 MHz

Load Breakpoint Position

Load Breakpoint Modulus

Configure Breakpoint

Configure Breakpoint Output

Enable Breakpoint Output

Set Breakpoint Output MOMO

Read Breakpoint Status

flex_configure_breakpoint

Device Compatibility

Configure Breakpoint

Usage

status = flex_configure_breakpoint(u8 boardID, u8 axisOrEncoder, u16
enableMode, u16 actionOnBreakpoint, u16 operation);

Purpose

Configure a position breakpoint on an axis or encoder.

Parameters

Name	Туре	Description	
boardID	u8	assigned by Measurement & Automation Explorer (MAX)	
axisOrEncoder	u8	axis or encoder to control	
enableMode u16		breakpoint mode	
actionOnBreakpoint	u16	action to perform when breakpoint is reached	
operation	u16	selects between single and buffered breakpoint operation	

Parameter Discussion

axisOrEncoder is the axis or encoder to control. You can enable breakpoints on encoders mapped to axes NIMC_AXIS1 through NIMC_AXIS30 or directly on encoders NIMC_ENCODER1 through NIMC_ENCODER30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

enableMode is the breakpoint enable mode.

enableMode Constant	enableMode Value	enableMode Restrictions
NIMC_ABSOLUTE_BREAKPOINT	1	—
NIMC_RELATIVE_BREAKPOINT	2	—
NIMC_MODULO_BREAKPOINT	3	Not applicable for buffered breakpoints.
NIMC_PERIODIC_BREAKPOINT	4	Not applicable for buffered breakpoints.

actionOnBreakpoint is the action to perform when the breakpoint is reached.

actionOnBreakpoint	actionOnBreakpoint Value	actio Restri
NIMC_NO_CHANGE	0	
NIMC_RESET_BREAKPOINT	1	Not a buffer
NIMC_SET_BREAKPOINT	2	Not a buffer
NIMC_TOGGLE_BREAKPOINT (7330/40/90 only)	3	
NIMC_PULSE_BREAKPOINT (7350 only)	4	

operation selects between single and buffered breakpoint operation. Valid values are NIMC_OPERATION_SINGLE (0) and NIMC_OPERATION_BUFFERED (1)

Using This Function

The Configure Breakpoint function configures the breakpoint to the specified mode, operation, and action. It also defines the action to perform when the breakpoint is reached—leave the breakpoint output unchanged, reset the breakpoint output to low, set the breakpoint output to high, or toggle the state of breakpoint output. Refer to the <u>Synchronization</u> section of the *NI-Motion Help* for more information about the types of breakpoints you can configure.

Note For modulo breakpoints, the magnitude of the breakpoint value must be less than the breakpoint modulus. If this range is exceeded, a modal error is generated when you execute the Enable Breakpoint Output function.

The **enableMode** parameter determines how the previously loaded breakpoint position is interpreted. Absolute breakpoints can be anywhere in the 32-bit position range. Relative breakpoints are relative to the instantaneous encoder position when the breakpoint is enabled. Modulo breakpoints are interpreted within the range of the loaded breakpoint modulus. Refer to the Load Breakpoint Modulus function for more information about modulo breakpoints.

When an enabled breakpoint is reached, a breakpoint event occurs. You can use the <u>Read Breakpoint Status</u> function to see if a breakpoint has occurred.

A breakpoint event also can cause the state of the corresponding breakpoint output to change. The **actionOnBreakpoint** parameter indicates if the output changes to low, high, toggles state, or does not change when the breakpoint event occurs. If the breakpoint output is presently in the state defined by **actionOnBreakpoint**, it is forced to the opposite state when the breakpoint is enabled. This guarantees that the specified transition occurs when the breakpoint is reached.

 $\overline{\mathbb{N}}$

Note NI-Motion does not support breakpoint and high-speed capture functionality on an axis when the primary feedback for that axis is an analog feedback.

All of the following modes are available for all NI motion controllers unless otherwise specified.

Absolute

This mode is available in both single and buffered operation. The breakpoint position is interpreted with reference to the zero position (origin). This also applies to all the position data in the buffer for buffered operation.

Relative

This mode is available in both single and buffered operation. For single operation, the breakpoint position is interpreted with reference to the encoder position when the breakpoint is enabled. For buffered operation, the first position data is interpreted with reference to the encoder position when the breakpoint is enabled. The subsequent position data is interpreted relative to the previous breakpoint such that:

```
Breakpoint[0] = Position on enable + Buffer[0]
Breakpoint[n] = Buffer[n-1] + Buffer[n],
where n>=1.
```

Modulus (7330/40/90)

This mode is available only in single operation. Modulus breakpoint is used to specify multiple breakpoints with a constant distance between them. The breakpoint position is interpreted with the modulus range such that

Breakpoint[n] = Breakpoint position + ($n \times$ Breakpoint modulus), where n is any integer.

When the breakpoint is enabled, the controller enables the two breakpoints closest to the current position, one in the forward direction, and one in the reverse. After a breakpoint occurs, you can re-enable the next breakpoint without loading a new position.

Periodic (7350)

This mode is available only in single operation. Periodic breakpoint is an enhanced version of modulus breakpoint. The period determines the distance between each breakpoint:

```
Breakpoint[n] = Breakpoint position + (n \times Breakpoint period), where n is any integer.
```

When you load a breakpoint position, it is interpreted as the first

breakpoint. After this breakpoint has occurred, the breakpoint status is asserted and the subsequent breakpoint is automatically loaded and enabled by the controller.

flex_configure_breakpoint_output

Device Compatibility

Configure Breakpoint Output

Usage

i32 status = flex_configure_breakpoint_output (u8 boardID, u8 axisOrEncoder, u16 polarity, u16 driveMode);

Purpose

Configures the drive mode and polarity of the breakpoint output.

Parameters

Name	Туре	Description		
boardID	u8	assigned by Measurement & Automation Explorer (MAX)		
axisOrEncoder	u8	ixis or encoder to configure		
polarity	u16	specifies either active low or active high polarity		
driveMode	u16	specifies either Open Collector or Totem Pole mode		

Parameter Discussion

axisOrEncoder is the axis or encoder to configure. You can configure breakpoints on encoders mapped to axes NIMC_AXIS1 through NIMC_AXIS30 or directly on encoders NIMC_ENCODER1 through NIMC_ENCODER30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

polarity is the breakpoint output configured as active low or active high. When configured as active low, the output is active when there is a low signal on the output pin. Conversely, active high means that the output is active when there is a high signal on the output pin.

Constant	Value	Description
NIMC_ACTIVE_HIGH	0	Logical True or On
NIMC_ACTIVE_LOW	1	Logical False or Off

driveMode specifies either **Open Collector** or **Totem Pole** mode. **driveMode** is available only on the 7350 and is ignored on 7330/40/90 motion controllers.

Constant	Value	Description
NIMC_OPEN_COLLECTOR	0	Configures output to be Open Collector
NIMC_TOTEM_POLE	1	Configures output to be Totem Pole

Using This Function

Use the Configure Breakpoint Output function to configure the **driveMode** and **polarity** of the breakpoint signal. For more information about **Open Collector** and **Totem Pole** drive modes, refer to the documentation for your 7350 device. Call this function at the beginning of the application before enabling the breakpoint with the <u>Enable</u> <u>Breakpoints</u> function.

flex_enable_breakpoint

Device Compatibility

Enable Breakpoint Output

Usage

status = flex_enable_breakpoint(u8 boardID, u8 axisOrEncoder, u8 enable);

Purpose

Enables a position breakpoint on an axis or encoder.

Parameters

Name	Туре	Description	
boardID	u8	assigned by Measurement & Automation Explorer (MAX)	
axisOrEncoder	u8	axis or encoder to control	
enable	u8	enable/disable value	

Parameter Discussion

axisOrEncoder is the axis or encoder to control. You can enable breakpoints on encoders mapped to axes NIMC_AXIS1 through NIMC_AXIS30 or directly on encoders NIMC_ENCODER1 through NIMC_ENCODER30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

enable indicates if the function enables or disables the high-speed capture input.

0 = Breakpoint disable

1 = Breakpoint enable

Using This Function

This function enables or disables the breakpoint with configuration set by the <u>Configure Breakpoint</u> function. For single breakpoints, you must reenable if you want another breakpoint. Buffered and periodic breakpoints only need to be enabled once. When an enabled breakpoint is reached, a breakpoint event occurs. You can use the <u>Read Breakpoint Status</u> function to see if a breakpoint has occurred yet or not.

You can enable only one breakpoint per encoder or axis at a time. When an enabled breakpoint is reached, the breakpoint is automatically disabled. You must explicitly re-enable the breakpoint to use it again.



Note Enabled breakpoints are also automatically disabled when you execute a <u>Reset Position</u> or <u>Reset Encoder Position</u> function on the corresponding axis.

Breakpoints are fully functional on independent encoders that are not mapped to axes. In this case, you enable breakpoints directly on the encoder resource.

Note If you use modulus breakpoints or enable the same breakpoint repeatedly at low velocity (< 50 counts/sec), you may get a breakpoint output immediately after you re-enable it, because the position may not have changed from the last breakpoint.

To avoid this problem, use MAX or the <u>Set u32</u> function to change the breakpoint window. The breakpoint window is a buffer around the breakpoint position in which a breakpoint is not enabled while the current position remains inside the buffer.

Note All breakpoints can be affected by jitter in the motion system. For example, if you have breakpoint positions very close to each other, the jitter in the motion system could cause the position to change enough to reach the breakpoint when a breakpoint is not intended. Increase the value of the breakpoint window using <u>Set u32</u> to compensate for system jitter.

Note Enabling a high-speed capture when a breakpoint is enabled may cause the breakpoint to be missed. If you are using a breakpoint on the same axis on which you are performing the highspeed capture, ensure that both the <u>breakpoint</u> and the <u>high-speed</u> <u>capture</u> are configured and enabled before starting the move.

flex_load_bp_modulus

Device Compatibility

Load Breakpoint Modulus

Usage

status = flex_load_bp_modulus(u8 boardID, u8 axisOrEncoder, u32 breakpointModulus, u8 inputVector);

Purpose

Load the breakpoint modulus for a position breakpoint.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer
axisOrEncoder	u8	axis or encoder to control
breakpointModulus	u32	breakpoint modulus in counts
inputVector	u8	source of the data for this function

Parameter Discussion

axisOrEncoder is the axis or encoder to control. You can load **breakpointModulus** on encoders mapped to axes NIMC_AXIS1 through NIMC_AXIS30 or directly on encoders NIMC_ENCODER1 through NIMC_ENCODER30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to Axes and Encoders for axis and encoder resource IDs.

breakpointModulus is the period, for periodic breakpoints, or the modulus, for modulus breakpoints. The range for period is 1 to 2^{15} –1. The range for modulus is 0 to 2^{31} –1. A modulus value of 0 makes a modulus breakpoint behave the same as an absolute breakpoint on 7330/40/90 controllers.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function loads a position modulus/period, depending on the breakpoint mode. This value is double-buffered in the controller and does not take effect until you execute the <u>Enable Breakpoint Output</u> function.

Modulo breakpoints are used in applications that require repetitive breakpoints equally spaced. When using a breakpoint modulus, it is no longer necessary to load ever increasing (or decreasing) breakpoint positions. It is still necessary, however, to re-enable the breakpoint after each use.

 $\overline{\mathbb{N}}$

M

Note For closed-loop stepper motion systems in which the encoder counts per revolution differs from the steps per revolution, the breakpoint position is loaded in encoder counts.

When you enable a modulo breakpoint, two breakpoint positions, one in front and one behind the present encoder position, are enabled.

Note If you use modulus breakpoint or enable the same breakpoint repeatedly at low velocity (< 50 counts/sec), you may get a breakpoint output immediately after you re-enable it, because the breakpoint is re-enabled before you have moved from the previous breakpoint position.

To avoid this problem, use MAX or the <u>Set u32</u> function to change the breakpoint window. The breakpoint window is a buffer around the breakpoint position in which a breakpoint is not enabled while the current position remains inside the buffer.

Example

An application requires breakpoints every 2,000 counts offset at –500 counts: ...–4,500, –2,500, –500, 1,500, 3,500, and so on. To accomplish this, you load a breakpoint position of –500 with the <u>Load Breakpoint</u> <u>Position</u> function and a breakpoint modulus of 2,000. If the instantaneous encoder position is 2,210 counts when you execute the Enable Breakpoint Output function (in modulo mode), the breakpoints at 1,500 counts and 3,500 counts are both enabled.

flex_load_pos_bp

Device Compatibility

Load Breakpoint Position

Usage

status = flex_load_pos_bp(u8 boardID, u8 axisOrEncoder, i32 breakpointPosition, u8 inputVector);

Purpose

Loads the breakpoint position for an axis or encoder in counts.
Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrEncoder	u8	axis or encoder to control
breakpointPosition	i32	breakpoint position in counts
inputVector	u8	source of the data for this function

Parameter Discussion

axisOrEncoder is the axis or encoder to control. You can load **breakpointPosition** on encoders mapped to NIMC_AXIS1 through NIMC_AXIS30 or directly on encoders NIMC_ENCODER1 through NIMC_ENCODER30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to Axes and Encoders for axis and encoder resource IDs.

breakpointPosition is the breakpoint position in quadrature counts. Breakpoint positions can be anywhere within the 32-bit position range, $-(2^{31})$ to $+(2^{31}-1)$. The default value is zero (0).

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function loads the breakpoint position value for the axis or encoder specified. The breakpoint position is interpreted differently depending on the breakpoint mode and the encoder position when the breakpoint is enabled. Breakpoint position is double-buffered and not actually used until you execute the <u>Enable Breakpoint</u> function.



Note For modulo breakpoints, the magnitude of the breakpoint value must be less than the breakpoint modulus. If this range is exceeded, a modal error is generated when you execute the <u>Enable Breakpoint Output</u> function.



Note Refer to the <u>Write Buffer</u> function to load breakpoint positions for buffered breakpoints. The Load Breakpoint Position function does not have any affect on buffered breakpoints.

When the breakpoint position is reached, a breakpoint event is generated and the associated high-speed breakpoint output immediately transitions.

High-speed breakpoint functionality is performed by the encoder resources themselves. When this function is sent to an axis, it is actually being sent to the mapped encoder resource. Therefore, for closed-loop stepper systems, the breakpoint position is the encoder count. If the steps/revolution and encoder counts/revolution values are different, be aware that, for this function, the encoder count is most useful.

Breakpoints are only available on the encoder resources (NIMC_ENCODER1 through NIMC_ENCODER30) and are always loaded in quadrature counts.

When the same breakpoint position is used repeatedly, it is not necessary to reload the position each time. It is necessary, however, to re-enable the breakpoint after each use.



Note For closed-loop stepper motion systems in which the encoder counts per revolution differs from the steps per revolution, the captured position is in encoder counts.

Device Compatibility

Read Breakpoint Status

Usage

status = flex_read_breakpoint_status(u8 boardID, u8 axisOrEncoder, u16 breakpointType, u8 returnVector);

Purpose

Reads the breakpoint status for all axes or encoders.



Note You cannot use this function to read the status of buffered breakpoints. Use <u>Check Buffer</u> to get the status of the buffer.

Parameters

Name	Туре	Description		
boardID	u8	assigned by Measurement & Automation Explorer (MAX)		
axisOrEncoder u8		axis control or encoder control selector		
breakpointType	u16	reserved (must be 0)		
returnVector	u8	destination for the return data		

Parameter Discussion

axisOrEncoder is the axis control or encoder control selector. For multiaxis status, use NIMC_AXIS_CTRL. For multi-encoder status, use NIMC_ENCODER_CTRL. Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

breakpointType is a reserved input that must be set to NIMC_POSITION_BREAKPOINT (0).

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function allows you to see if a breakpoint has occurred or is pending. When you enable a breakpoint, the corresponding status bit is reset to indicate that the breakpoint is pending. For single breakpoints, this status is set when a breakpoint occurs. Periodic breakpoints only set the breakpoint status after the first breakpoint has occurred, and the status remains asserted across the subsequent breakpoints. When the breakpoint position is reached, its status bit is set to True (1).

Note Read Breakpoints Status returns the breakpoint status for either all axes or all encoders, based on which value is passed into axisOrEncoder.

Example

Executing the Read Breakpoint Status function with **axisOrEncoder** = NIMC_ENCODER_CTRL and **breakpointType** = 0 returns **breakpointStatus** = 0x0012, which corresponds to the following bitmap:

D15	••••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	C
XXX		XXX	XXX	Enc 8	Enc 7	Enc 6	Enc 5	Enc 4	Enc 3	Enc 2	Enc 1	Х
0		0	0	0	0	0	0	1	0	0	1	0

On encoders 1 and 4, breakpoints have occurred, but on encoders 2 and 3, breakpoints are pending or were never enabled.

flex_read_breakpoint_status_rtn

Device Compatibility

Read Breakpoint Status Return

Usage

status = flex_read_breakpoint_status_rtn(u8 boardID, u8 axisOrEncoder, u16 breakpointType, u16* breakpointStatus);

Purpose

Reads the breakpoint status for all axes or encoders.



Note You cannot use this function to read the status of buffered breakpoints. Use <u>Check Buffer</u> to get the status of the buffer.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrEncoder	u8	axis control or encoder control selector
breakpointType	u16	reserved (must be 0)
breakpointStatus	u16*	bitmap of breakpoint status

Parameter Discussion

axisOrEncoder is the axis control or encoder control selector. For multiaxis status, use NIMC_AXIS_CTRL. For multi-encoder status, use NIMC_ENCODER_CTRL. Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

breakpointType is a reserved input that must be set to NIMC_POSITION_BREAKPOINT (0).

breakpointStatus is the bitmap of breakpoint status for all axes or all encoders.



Note The breakpoint status does not reflect the voltage level of the breakpoint output.

When reading breakpoint status for axes (axisOrEncoder = NIMC_AXIS_CTRL):

D15	•••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
XXX	••••	XXX	XXX	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	XXX

For D1 through D8:

1 = Breakpoint occurred

0 = Breakpoint pending or disabled

When reading breakpoint status for encoders (**axisOrEncoder** = NIMC_ENCODER_CTRL):

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	С
XXX	 XXX	XXX	Enc 8	Enc 7	Enc 6	Enc 5	Enc 4	Enc 3	Enc 2	Enc 1	Х

For D1 through D8:

1 = Breakpoint occurred

0 = Breakpoint pending or never enabled

Using This Function

This function allows you to see if a breakpoint has occurred or is pending. When you enable a breakpoint, the corresponding status bit is reset to indicate that the breakpoint is pending. When the breakpoint position is reached, its status bit is set to True (1).



Note Read Breakpoints Status returns the breakpoint status for either all axes or all encoders, based on which value is passed into **axisOrEncoder**.

Example

Executing the Read Breakpoint Status function with **axisOrEncoder** = NIMC_ENCODER_CTRL and **breakpointType** = 0 returns **breakpointStatus** = 0x0012, which corresponds to the following bitmap:

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	C
XXX	 XXX	XXX	Enc 8	Enc 7	Enc 6	Enc 5	Enc 4	Enc 3	Enc 2	Enc 1	Х
0	 0	0	0	0	0	0	1	0	0	1	0

On encoders 1 and 4, breakpoints have occurred, but on encoders 2 and 3, breakpoints are pending or were never enabled.

flex_set_breakpoint_output_momo

Device Compatibility

Set Breakpoint Output MOMO

Usage

status = flex_set_breakpoint_output_momo(u8 boardID, u8 axisOrEncoder, u16
mustOn, u16 mustOff, u8 inputVector);

Purpose

Sets the breakpoint outputs using the **mustOn/mustOff** protocol.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrEncoder	u8	axis or encoder selector
mustOn	u16	bitmap of breakpoint outputs to force On
mustOff	u16	bitmap of breakpoint outputs to force Off
inputVector	u8	source of the data for this function

Parameter Discussion

axisOrEncoder is the axis or encoder selector. To set breakpoint outputs on multiple axes, use NIMC_AXIS_CTRL. To set breakpoint outputs on multiple encoder resources, use NIMC_ENCODER_CTRL. Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

mustOn is the bitmap of breakpoint outputs to force On.

D15		D10	D9	D8	D7	D6	D5	D4
XXX	••••	XXX	XXX	mustOn 8	mustOn 7	mustOn 6	mustOn 5	mustOn

For D1 through D15:

1 = Breakpoint output forced On

0 = Breakpoint output unchanged (default)

mustOff is the bitmap of breakpoint outputs to force Off.

D15		D10	D9	D8	D7	D6	D5	D4
XXX	••••	XXX	XXX	mustOff 8	mustOff 7	mustOff 6	mustOff 5	mustOf

For D1 through D15:

1 = Breakpoint output forced Off

0 = Breakpoint output unchanged (default)

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function directly controls the breakpoint outputs and sets them high or low. You can use this function to set breakpoint outputs to a known state or to control them as general-purpose outputs in non-breakpoint applications. You can only control the output using this function if the breakpoint is disabled.



Note The Set Breakpoint MOMO function only affects the output state of the breakpoint. It does not change the breakpoint status.

Breakpoint functionality is performed by the encoder resources themselves. When this function is sent to axes, the NI-Motion firmware consults the mapping of axes to encoders and actually sends the command to the mapped encoder resources. Breakpoints are only available on encoder resources NIMC_ENCODER1 through NIMC_ENCODER30.

Using the **mustOn/mustOff** protocol allows you to set or reset individual breakpoint outputs without affecting the other breakpoint outputs. This gives you tri-state control over each output: On, Off, or Unchanged. A one (1) in a bit location of the **mustOn** bitmap sets the breakpoint high, while a one (1) in the corresponding location of the **mustOff** bitmap resets the breakpoint low. A zero (0) in either bitmap has no effect, so leaving both the **mustOn** and **mustOff** bits at zero is effectively a hold, and the state of the breakpoint output is unchanged. If you set both the **mustOn** and **mustOff** bits to one (1), it is interpreted as a **mustOn** condition and the breakpoint is set high.

Note If the breakpoint is enabled using the Enable Breakpoint function, you cannot change the output value using Set Breakpoint Output MOMO. The command is ignored by the motion controller.

Note The first input vector source indicates the value for the **mustOn** input and the second vector source indicates the value for the **mustOff** input. It is not a bitmap of the breakpoint output.

If you are doing onboard programming and are using **inputVector** to get the data this VI needs, remember that this VI reads the variables starting at the memory address pointed to by **inputVector** in the following order: **mustOn**, **mustOff**.

High-Speed Capture

High-speed capture inputs are an enhancement to the encoder FPGA and are available when the encoders are operating as axis feedback or as independent encoder resources. High-speed capture functionality is available on servo and closed-loop stepper axes.

You can use High-Speed Capture functions to configure, enable, and read the status of a high-speed inputs. Also available on the NI-7350 motion controller is buffered high-speed capture. Buffered capture is a more advanced use of high-speed capture where the captured positions are stored in a previously-configured buffer. This helps the high-speed capture perform at higher capture frequencies. Refer to the <u>Buffered</u> <u>Operations Overview</u> for details on buffer operations in buffered highspeed capture for the 7350 controller.

The high-speed capture inputs also can functions as latching generalpurpose inputs. Configure as you would for high-speed capture operation, but ignore the captured position. You can then read the state of the latched inputs.

Configure High-Speed Capture

Enable High-Speed Capture

Read Captured Position

Read High-Speed Capture Status

flex_configure_hs_capture

Device Compatibility

Configure High-Speed Capture

Usage

status = flex_configure_hs_capture(u8 boardID, u8 axisOrEncoder, u16 captureMode, u16 operation);

Purpose

Configures the high-speed capture input for the specified signal behavior.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrEncoder	u8	axis or encoder to configure
captureMode	u16	how the controller interprets incoming signal
operation	u16	selects between single and buffered high-speed capture operation

Parameter Discussion

Inverting Digital Input

axisOrEncoder is the axis or encoder to configure. Valid values are axes NIMC_AXIS1 through NIMC_AXIS30 or directly on encoders NIMC_ENCODER1 through NIMC_ENCODER30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

captureMode **Constant Name** Value NIMC HS NON INVERTING LEVEL 0 Non-inverting Level 1 **Inverting Level** NIMC HS INVERTING LEVEL 2 NIMC HS LOW TO HIGH EDGE Low-to-high Edge High-to-low Edge NIMC HS HIGH TO LOW EDGE 3 Non-inverting Digital Input NIMC HS NON INVERTING DI 4

captureMode specifies how the incoming high-speed capture is interpreted by the motion controller.

Modes 0 through 3 define the state of the input signal that results in a high-speed capture event. Inverting level means an active low input triggers a capture. Conversely, non-inverting level means an active high input triggers a capture. Notice that if the signal is already at the specified level when high-speed capture is enabled, the capture occurs immediately. High-to-low edge requires that the signal make a transition from high to low to trigger a capture. Low-to-high edge requires that the signal make a transition from low to high to trigger a capture.

NIMC HS INVERTING DI

5

Modes 4 and 5 let you configure the high-speed capture input as a simple digital input independent of the high-speed capture circuitry. You use the <u>Read High-Speed Capture Status</u> function to read this state of the digital input. This functionality is useful when you do not need the high-speed capture capability, but need an extra digital input for general-purpose use. When configured in these modes, there is no need to enable the high-speed capture input.

operation selects between single and buffered high-speed capture operation. Valid values are NIMC_OPERATION_SINGLE (0) and NIMC_OPERATION_BUFFERED (1)



Note The mode set by Configure High Speed Capture does not take effect until Enable High-Speed Capture is called.

Using This Function

 $\overline{\mathbb{N}}$

Note NI-Motion does not support breakpoint and high-speed capture functionality on an axis when the primary feedback for that axis is analog feedback.

After configuring the high-speed capture input, enable the high-speed capture circuitry with the <u>Enable High-Speed Capture</u> function. The <u>Read</u> <u>High-Speed Capture Status</u> function tells you when the capture event has occurred. Finally, you read the captured value with the <u>Read Captured</u> <u>Position</u> function.

High-speed capture inputs are an integral part of the encoder resources. You can execute this function indirectly on axes or directly on encoder resources. After it is enabled, the controller captures the instantaneous encoder position when the input becomes active, as specified in the **captureMode** parameter.

High-speed capture works only on axes that have an encoder as primary feedback or on encoders directly.

High-speed capture functionality is available in both single and buffered mode. In single high-speed capture, the input trigger is configured to capture a single encoder position, and must be re-enabled after each capture. You can use buffered high-speed capture to support higher capture frequencies. Buffered high-speed capture allows captures positions to be stored into a buffer and automatically re-enabled by the controller. Refer to the <u>Configure Buffer</u> function to configure a buffer for high-speed capture.

You can use a high-speed capture input as a general-purpose input and read its status with <u>Read High-Speed Capture Status</u>. In modes 0 through 3, the input has latching behavior. This means that after the high-speed capture input has been enabled, if the input signal ever becomes active (as defined by the **captureMode** parameter), the high-speed capture status is true until the input is reenabled. In modes 4 and 5, the inputs simply reflect the current state of the input signal. The input does not need to be enabled or reenabled at any time in these modes.



Note When configured as a digital input, <u>Enable High-Speed</u> <u>Capture</u> and <u>Read Captured Position</u> have no effect. The highspeed capture input is used as a general-purpose digital input. Refer to <u>Read High-Speed Capture Status</u> for information about reading the input status.

Note National Instruments suggests that you do not change the capture mode when the high-speed capture is already enabled. Doing so might cause a false capture immediately after the Configure High-Speed Capture function is invoked.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- You can synchronize NI 73xx motion controllers with other National Instruments devices using the RTSI bus. Refer to the <u>Select Signal</u> function for information about using the RTSI bus as the high-speed capture trigger.
- NI 73xx controllers do not support multiple simultaneous highspeed captures per axis. NIMC_SECOND_HS_CAPTURE is not supported.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

• The NI SoftMotion Controller supports multiple simultaneous highspeed captures per axis.

To configure the first high-speed capture line on an axis, use the constant for that axis. For example, if the first high-speed capture is on axis 1, the function call appears as

```
status = flex_configure_hs_capture ( boardID, NIMC_AXIS1,
NIMC_HS_LOW_TO_HIGH_EDGE,
NIMC_OPERATION_SINGLE);
```

To configure the second high-speed capture line on an axis, use the second high-speed capture constants NIMC_SECOND_HS_CAPTURE1 through NIMC_SECOND_HS_CAPTURE30. For example, if the second high-speed capture is on axis 1, the function call appears as

```
status = flex_configure_hs_capture ( boardID,
NIMC_SECOND_HS_CAPTURE1,
NIMC_HS_LOW_TO_HIGH_EDGE,
NIMC_OPERATION_SINGLE);
```

• The NI SoftMotion Controller does not support buffered high-speed

captures.

- The NI SoftMotion Controller for the Copley Controls CANopen devices, Accelnet and Xenus, does not support high-speed captures.
- The NI SoftMotion Controller does not support RTSI synchronization.

flex_enable_hs_capture

Device Compatibility

Enable High-Speed Capture

Usage

status = flex_enable_hs_capture(u8 boardID, u8 axisOrEncoder, u16 enable);

Purpose

Enables or disables the specified high-speed capture input.

Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID u8		assigned by Measurement & Automation Explorer (MAX)
axisOrEncoder	u8	axis or encoder to enable or disable
enable	u16	enable/disable value
Parameter Discussion

axisOrEncoder is the axis or encoder selector. Valid values are axes NIMC_AXIS1 through NIMC_AXIS30 or directly on encoders NIMC_ENCODER1 through NIMC_ENCODER30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

enable indicates if the function enables or disables the high-speed capture input.

1 = Capture enabled0 = Capture disabled

Using This Function

This function enables high-speed capture inputs to capture instantaneous encoder position when an input becomes active. The position capture is implemented in the encoder resource to reduce capture latency to the sub-100 ns range.



Note If your application uses the <u>Select Signal</u> function to route the high-speed capture source, you must call the Select Signal function before enabling the high-speed capture.

High-speed capture functionality is performed by the encoder resources themselves. When this function is sent to an axis, it is actually being sent to the mapped encoder resource.

The high-speed inputs have programmable polarity and edge behavior. You can set the polarity of the input with the <u>Configure High-Speed</u> <u>Capture</u> function. You can determine the results of the high-speed capture from the <u>Read High-Speed Capture Status</u> and <u>Read Captured</u> <u>Position</u> functions.

Using single high-speed capture, you can enable only one high-speed capture per axis or encoder at a time. When an enabled event is captured, the high-speed capture is automatically disabled. You must explicitly re-enable the high-speed capture to use it again. To disable a previously-enabled operation, call this function again with enable set to 0.

For buffered high-speed capture, you need to enable the input trigger only once. Subsequent high-speed captures are automatically re-enabled by the controller. This function has no effect when the input trigger is configured as a digital input.

If you are using buffered high-speed capture, ensure that the buffer is configured before you execute this function. Refer to <u>Configure Buffer</u> for more information.

High-speed capture is useful in registration and synchronization applications. You can calculate subsequent moves relative to the captured position. Refer to the <u>Set Operation Mode</u> function for information about relative-to-capture mode. Refer to the <u>Gearing</u> section of the *NI-Motion Help* for information about using <u>superimposed</u> moves/registration applications. Also, refer to the <u>Rotating Knife</u> section of the *NI-Motion Help* example code that includes superimposed moves.



Note Enabling a high-speed capture input when the input is already active captures the position immediately and sets the status bit. If you change the mode with Configure High-Speed Capture after you enable high-speed capture for this axis, the new mode does not take effect until the next time Enable High-Speed Capture is called.



Note Enabling a high-speed capture when a <u>breakpoint is enabled</u> may cause the breakpoint to be missed. If you are using a breakpoint on the same axis on which you are performing the high-speed capture, ensure that both the <u>breakpoint</u> and the <u>high-speed</u> <u>capture</u> are configured and enabled before starting the move.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- If your application includes a high-speed capture that takes place after a <u>Find Reference</u> move that finds an index, you must enable the high-speed capture *after* the find reference move. If your application includes a buffered high-speed capture, you must also <u>configure the buffer</u> after the find reference move.
- NI 73xx controllers do not support multiple simultaneous highspeed captures per axis. NIMC_SECOND_HS_CAPTURE is not supported.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

• The NI SoftMotion Controller supports multiple simultaneous highspeed captures per axis.

To configure the first high-speed capture line on an axis, use the constant for that axis. For example, if the first high-speed capture is on axis 1, the function call appears as

```
status = flex_enable_hs_capture ( boardID, NIMC_AXIS1, TRUE);
```

To configure the second high-speed capture line on an axis, use the second high-speed capture constants NIMC_SECOND_HS_CAPTURE1 through NIMC_SECOND_HS_CAPTURE30. For example, if the second high-speed capture is on axis 1, the function call appears as

```
status = flex_enable_hs_capture ( boardID,
NIMC_SECOND_HS_CAPTURE1, TRUE);
```

• The NI SoftMotion Controller for Copley Controls' Accelnet and Xenus CANopen drives does not support multiple simultaneous high-speed captures per axis. NIMC_SECOND_HS_CAPTURE is not supported.

flex_read_cap_pos

Device Compatibility

Read Captured Position

Usage

status = flex_read_cap_pos(u8 boardID, u8 axisOrEncoder, u8 returnVector);

Purpose

Reads a captured position value from an axis or encoder.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrEncoder	u8	axis or encoder to read
returnVector	u8	destination for the return data

Parameter Discussion

axisOrEncoder is the axis or encoder to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_ENCODER1 through NIMC_ENCODER30. On motion controllers that support fewer than thirty axes, reading non-existent axes or encoders returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no returnVector is required.

Using This Function

This function returns the value in the single high-speed capture register of the axis or encoder selected. This value was captured when an enabled high-speed capture input went active.

For buffered high-speed capture, the captured positions are stored in a buffer. Use <u>Read Buffer</u> to read the captured positions.

High-speed capture functionality is performed by the encoder resources themselves. When this function is sent to an axis, the value returned is actually from the mapped encoder resource.

Refer to the <u>Enable High-Speed Capture</u> and <u>Read High-Speed Capture</u> <u>Status</u> functions for more information about the high-speed capture inputs and typical applications.



Note For closed-loop stepper motion systems in which the encoder counts per revolution differs from the steps per revolution, the captured position is in encoder counts.

flex_read_cap_pos_rtn

Device Compatibility

Read Captured Position Return

Usage

status = flex_read_cap_pos_rtn(u8 boardID, u8 axisOrEncoder, i32*
capturedPosition);

Purpose

Reads a captured position value from an axis or encoder.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrEncoder	u8	axis or encoder to read
capturedPosition	i32*	position value captured

Parameter Discussion

axisOrEncoder is the axis or encoder to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_ENCODER1 through NIMC_ENCODER30. On motion controllers that support fewer than thirty axes, reading non-existent axes or encoders returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

capturedPosition is the position value captured when the corresponding high-speed capture input went active.

Using This Function

This function returns the value in the single high-speed capture register of the axis or encoder selected. This value was captured when an enabled high-speed capture input went active.

For buffered high-speed capture, the captured positions are stored in a buffer. Use <u>Read Buffer</u> to read the captured positions.

High-speed capture functionality is performed by the encoder resources themselves. When this function is sent to an axis, the value returned is actually from the mapped encoder resource.

Refer to the <u>Enable High-Speed Capture</u> and <u>Read High-Speed Capture</u> <u>Status</u> functions for more information about the high-speed capture inputs and typical applications.



Note For closed-loop stepper motion systems in which the encoder counts per revolution differs from the steps per revolution, the captured position is in encoder counts.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

• NI 73xx controllers do not support multiple simultaneous highspeed captures per axis. NIMC_SECOND_HS_CAPTURE is not supported.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

• The NI SoftMotion Controller supports multiple simultaneous highspeed captures per axis

To read the captured position on an axis, use the constant for that axis. For example, if the first high-speed capture is on axis 1, the function call appears as

```
status = flex_read_cap_pos_rtn (boardID, NIMC_AXIS1,
&capturedPosition);
```

To read the captured position on the secondary high-speed capture line, use the second high-speed capture constants NIMC_SECOND_HS_CAPTURE1 through NIMC_SECOND_HS_CAPTURE30. For example, if the second high-speed capture is on axis 1, the function call appears as

```
status = flex_read_cap_pos_rtn (boardID,
NIMC_SECOND_HS_CAPTURE1, &capturedPosition);
```

• The NI SoftMotion Controller for Copley Controls' Accelnet and Xenus CANopen drives does not support multiple simultaneous high-speed captures per axis. NIMC_SECOND_HS_CAPTURE is not supported.

flex_read_hs_cap_status

Device Compatibility

Read High-Speed Capture Status

Usage

status = flex_read_hs_cap_status(u8 boardID, u8 axisOrEncoder, u8
returnVector);

Purpose

Reads the high-speed position capture status for all axes or encoders.

Note You cannot use this function to read the status of buffered high-speed captures. Use <u>Check Buffer</u> to get the status of the buffer.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrEncoder	u8	axis or encoder selector
returnVector	u8	destination for the return data

Parameter Discussion

axisOrEncoder is the axis or encoder selector. For multi-axis status, use NIMC_AXIS_CTRL. For multi-encoder status, use NIMC_ENCODER_CTRL. Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function allows you to see if a position capture has occurred or is pending. When configured as a single high-speed capture, the corresponding status bit is set when an input trigger becomes active, as configured in <u>Configure High-Speed Capture</u>. The status bit, along with the captured encoder position, is latched until the input trigger is re-enabled. When the position capture occurs, its status bit is set to True (1). Refer to the <u>Read Captured Position</u> function for information about retrieving the captured position value.

Example

Executing this function with **axisOrEncoder** = NIMC_AXIS_CTRL returns **highSpeedCaptureStatus** = 0x000C, which corresponds to the following bitmap:

D15	•••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
XXX		XXX	XXX	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	XXX
0		0	0	0	0	0	0	0	1	1	0	0

On encoders mapped to axes 2 and 4, high-speed captures have occurred, but all other captures are pending or were never enabled.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- The high-speed capture circuitry is also used during a Find Reference Index execution. When an index is found successfully, the capture status for the corresponding encoder and axis is set to True as a side effect.
- Executing the Find Reference function to find an index position automatically leaves the corresponding high-speed capture input disabled after the index is found.

Read High-Speed Capture Status Return

Usage

status = flex_read_hs_cap_status_rtn(u8 boardID, u8 axisOrEncoder, u16* highSpeedCaptureStatus);

Purpose

Reads the high-speed position capture status for all axes or encoders.

Note You cannot use this function to read the status of buffered high-speed captures. Use <u>Check Buffer</u> to get the status of the buffer.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description					
boardID	u8	assigned by Measurement & Automation Explorer (MAX)					
axisOrEncoder	u8	axis or encoder selector					
highSpeedCaptureStatus	u16*	bitmap of high-speed capture status					

Parameter Discussion

axisOrEncoder is the axis or encoder selector. For multi-axis status, use NIMC_AXIS_CTRL. For multi-encoder status, use

NIMC_ENCODER_CTRL. Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

highSpeedCaptureStatus is the bitmap of capture status for all axes or all encoders.

When reading high-speed capture status for axes (**axisOrEncoder** = NIMC_AXIS_CTRL):

D15	••••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis		Axis	0									
15		10	9	8	7	6	5	4	3	2	1	

For D1 through D15:

1 = Capture occurred

0 = Capture pending or disabled

When reading high-speed capture status for encoders (**axisOrEncoder** = NIMC_ENCODER_CTRL):

D15		D10	D9	D8	D7	D6	D5	D4	D3	D2	
Enc 15	••••	Enc 10	Enc 9	Enc 8	Enc 7	Enc 6	Enc 5	Enc 4	Enc 3	Enc 2	ł

For D1 through D15:

1 = Capture occurred

0 = Capture pending or disabled

Using This Function

This function allows you to see if a position capture has occurred or is pending. When configured as a single high-speed capture, the corresponding status bit is set when an input trigger becomes active, as configured in <u>Configure High-Speed Capture</u>. The status bit, along with the captured encoder position, is latched until the input trigger is re-enabled. Refer to the <u>Read Captured Position</u> function for information about retrieving the captured position value.

Example

Executing the Read High-Speed Capture Status function with **axisOrEncoder** = NIMC_AXIS_CTRL returns **highSpeedCaptureStatus** = 0x000C, which corresponds to the following bitmap:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D
Axis	A													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

On encoders mapped to axes 2 and 3, high-speed captures have occurred, but all other captures are pending or were never enabled.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- The high-speed capture circuitry is also used during Find Index execution. When an index is found successfully, the capture status for the corresponding encoder and axis is set to True as a side effect.
- Executing the Find Reference Index function automatically leaves the corresponding high-speed capture input disabled after the index is found.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

• The NI SoftMotion Controller supports multiple simultaneous highspeed captures per axis.

To read the first high-speed capture line on an axis, use the constant for that axis. For example, if the first high-speed capture is on axis 1, the function call appears as

```
status = flex_read_hs_cap_status_rtn (boardID, NIMC_AXIS1,
&highSpeedCaptureStatus);
```

To read the second high-speed capture line on an axis, use the second high-speed capture constants

NIMC_SECOND_HS_CAPTURE1 through

NIMC_SECOND_HS_CAPTURE30. For example, if the second high-speed capture is on axis 1, the function call appears as

```
status = flex_read_hs_cap_status_rtn (boardID,
NIMC_SECOND_HS_CAPTURE1, &highSpeedCaptureStatus);
```

• The NI SoftMotion Controller for Copley Controls' Accelnet and Xenus CANopen drives does not support multiple simultaneous high-speed captures per axis. NIMC_SECOND_HS_CAPTURE is not supported.

- Executing the Find Reference Index function does not affect the capture status.
- To obtain the high-speed capture status and secondary high-speed capture status of axes sixteen through thirty, use the <u>Read per Axis</u> <u>Status Return</u> function.

Find Reference

You can use these functions to initialize the motion system and establish a repeatable reference position.

Typical closed-loop motion systems use incremental feedback to keep track of position. At power-up, this position is meaningless until a zero reference position is established. Open-loop stepper systems must also be initialized at power-up.

NI-Motion provides two built-in functions, Find Reference and Wait Reference, to accomplish these tasks. Find Reference performs search sequences to find and stop on a specific edge of the home input, find the next instance of the encoder index, find the center of travel between forward and reverse limits, or find the forward/reverse limit. Wait Reference returns when an initiated find has completed. In this way a repeatable reference position that is accurate to one encoder count is established.

Find Reference Check Reference Wait Reference Read Reference Status Load Reference Parameter Get Reference Parameter flex_find_reference

Device Compatibility

Find Reference

Usage

status = flex_find_reference(u8 boardID, u8 axisOrVectorSpace, u16 axisOrVSMap, u8 searchType);
Purpose

Executes a search operation to find a reference position: <u>home</u>, <u>index</u>, <u>center</u>, <u>forward limit</u>, <u>reverse limit</u>, or <u>run sequence</u>.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space selector
axisOrVSMap	u16	bitmap of axes or vector spaces to find references for
searchType	u8	reference position to find

Parameter Discussion

axisOrVectorSpace can select an axis (NIMC_AXIS1 through NIMC_AXIS30), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously searching on multiple axes or vector spaces, the **axisOrVSMap** parameter indicates which axes or vector spaces are involved. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

axisOrVSMap is the bitmap of axes or vector spaces to search on. It is only required when multiple axes or vector spaces are selected with the **axisOrVectorSpace** parameter. NI-Motion ignores additional axes and vector spaces if you select non-existent axes.

When starting axes (**axisOrVectorSpace** = NIMC_AXIS_CTRL):

D15	••••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis		Axis	0									
15		10	9	8	7	6	5	4	3	2	1	

For D1 through D15:

1 = Start axis

0 = Do not start axis

When starting vector spaces (**axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL):

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS 15	 VS 10	VS 9	VS 8	VS 7	VS 6	VS 5	VS 4	VS 3	VS 2	VS 1	0

For D1 through D15:

1 = Start vector space

0 = Do not start vector space

To start a find on a single axis or vector space, set **axisOrVectorSpace** to the specified axis or vector space. The **axisOrVSMap** bitmap is ignored.

To start a find on multiple axes, set **axisOrVectorSpace** to

NIMC_AXIS_CTRL and the **axisOrVSMap** bitmap defines the axes to start. Similarly, to start a find on multiple vector spaces, set **axisOrVectorSpace** to NIMC_VECTOR_SPACE_CTRL and the **axisOrVSMap** bitmap defines the vector spaces to start.



Note It is not possible to combine a Find Reference on an axis and a vector space in a single use of this function. To accomplish this, create a single axis vector space and then execute a multivector space Find Reference.

Search Type

Value	Definition	Description
0	Find Home	Executes a Find Home on the given set of axes.
1	Find Index	Executes a Find Index on the given set of axes.
2	Find Center	Finds the center of travel on a given set of axes.
3	<u>Find Forward</u> Limit	Finds the forward limit on a given set of axes.
4	<u>Find Reverse</u> Limit	Finds the reverse limit on a given set of axes.
5	Run Sequence	Runs the loaded Find Sequence.

Using This Function

This function returns immediately after starting the given find operation. Use this function in conjunction with <u>Wait Reference</u> to determine when the find operation completes and if the find was successful.

You can manually stop a search operation using MAX or by calling the <u>Stop Motion</u> function. When encountering a limit or home input a search operation performs a halt stop and then continues based on the **Search Type**. An unexpected limit condition during a find stops the sequence and generates a modal error. Refer to <u>Errors and Error Handling</u> for information about errors.



Note You must set unused limit and home inputs to their inactive state using MAX or with the <u>Enable Limits</u> and <u>Enable Home</u> <u>Inputs</u> functions.

Note Forward is defined as the direction of increasing position. The Forward and Reverse Limits must be located at the proper ends of travel for Find Reference to function properly.

You can access settings for Find Reference through MAX, <u>Load</u> <u>Reference Parameter</u>, <u>Get Reference Parameter</u>, <u>Set u32</u> and <u>Get u32</u>. Refer to the section about each type for a detailed description on how these settings affect a given **Search Type**. Most of these settings are on a per-**Search Type** basis.



Note If you start a coordinate (vector) space, Find Reference actually starts the axes in the coordinate space individually, so it is not considered a vector move.

Search Types

- Find Home
- Find Index
- Find Center
- Find Forward and Reverse Limits
- Run Sequence

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- If your application includes a high-speed capture that takes place after a <u>Find Reference</u> move that finds an index, you must enable the high-speed capture *after* the find reference move. If your application includes a buffered high-speed capture, you must also <u>configure the buffer</u> after the find reference move.
- When performing a reset position as part of a Find Reference operation, add at least one of the following to the move complete criteria to ensure that the motor is not moving when the reset position executes:
 - Add a delay after the profile complete status is true to allow sufficient time between the output and reset position for the motor to stop. Use <u>Configure Move Complete Criteria</u> to configure a delay.
 - Use the in-position signal as a criteria for move complete. The NI PCI-7390 controller uses this signal by default. To add this signal to the move complete criteria for controllers other than the NI PCI-7390, use <u>Configure Motion I/O Map</u> to map the signal to a pin and then use <u>Write Trajectory Data</u> to add in-position to the move complete criteria. In addition, configure the in-position range on the drive to a small enough number to ensure that the target position is reached. Refer to your drive documentation for information about configuring the in-position range on the drive.

NI SoftMotion Controller Considerations

The following includes considerations you must make when you are using this function with the NI SoftMotion Controller:

You cannot execute this function on multiple axes using the NIMC_AXIS_EX_CTRL resource parameter with axes sixteen through thirty.

flex_wait_reference

Device Compatibility

Wait Reference

Usage

status = flex_wait_reference(u8 boardID, u8 axisOrVectorSpace, u16 axisOrVSMap, u32 timeout, u32 pollingInterval, u16* found);

Purpose

Waits for a search sequence initiated by <u>Find Reference</u> to complete and returns the status. Wait Reference also can be used to query the status of a search.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space selector
axisOrVSMap	u16	bitmap of axes or vector spaces
timeout	u32	timeout in milliseconds
pollingInterval	u32	the distance in time between the polling of the find
found	u16*	returns true if the reference is found

Parameter Discussion

axisOrVectorSpace selects an axis (NIMC_AXIS1 through NIMC_AXIS30), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously waiting on multiple axes or vector spaces, the **axisOrVSMap** parameter indicates which axes or vector spaces are involved. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector</u> <u>Spaces</u> for axis and vector space resource IDs.

axisOrVSMap is the bitmap of axes or vector spaces to wait on. It is only required when multiple axes or vector spaces are selected with the **axisOrVectorSpace** parameter. NI-Motion ignores additional axes and vector spaces if you select non-existent axes.

When starting axes (axisOrVectorSpace = NIMC_AXIS_CTRL):

D15 .	•••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis . 15		Axis 10	Axis 9	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	0

For D1 through D15:

1 = Start axis

0 = Do not start axis

When starting vector spaces (**axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL):

D15		D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS 15	••••	VS 10	VS 9	VS 8	VS 7	VS 6	VS 5	VS 4	VS 3	VS 2	VS 1	0

For D1 through D15:

1 = Start vector space

0 = Do not start vector space

timeout specifies how long, in milliseconds, the Wait Reference call waits before returning. If the timeout is reached before the search completes, the Wait Reference returns. In this manner, Wait Reference can be used to check the status of a search by passing a timeout of 0. By default, the timeout is 4294967295 or 0xFFFFFFF. This value is handled differently,

signifying an infinite wait time.

pollingInterval is the distance in time between the polling of the find reference activity.

found is a returned Boolean value that indicates the success of the find it is waiting for. It does not matter if different finds are being executed on different axes. When all finds are done on all the axes or vector spaces specified, **found** is true (1) if all of the finds were successful. If any find did not complete successfully, **found** is false (0).

Using This Function

This function returns when the find on the indicated axes finishes or the timeout is reached, whichever occurs first. There is no need to specify what type of find to wait for because the controller knows what searches are being done on what axes. This function waits for and reports on the success of the current or last executed find on an axis, vector space, or a given set of axes or vector spaces as a whole.

If a <u>Find Reference</u> is being executed with the Run Sequence option, the Wait Reference function does not require an **axisOrVectorSpace** or **axisOrVSMap**. When waiting on a Run Sequence, the Wait Reference function returns when the entire sequence has finished. The **found** Boolean returns True only if all of the operations in the sequence were successful.



Note You cannot execute any other processes while the Wait Reference function is running. Use <u>Check Reference</u> or <u>Read</u> <u>Reference Status</u> if you want to run other processes simultaneously.

Example

You initiated a Find Home on Axis 1, and Find Center on Axis 2 and Axis 3, and you want to wait for all find sequences to finish before proceeding with the rest of the program. Simply call Wait Reference with the following parameters:

axisOrVectorSpace set to Axis Control (NIMC_AXIS_CTRL) axisMap set to 00001110 (axis 1, 2, & 3 enabled)

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

You cannot execute this function on multiple axes using the NIMC_AXIS_EX_CTRL resource parameter with axes sixteen through thirty.

flex_check_reference

Device Compatibility

Check Reference

Usage

status = flex_check_reference(u8 boardID, u8 axisOrVectorSpace, u16 axisOrVSMap, u16* found, u16* finding);

Purpose

Checks the status of a search sequence initiated by <u>Find Reference</u>.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space selector
axisOrVSMap	u16	bitmap of axes or vector spaces to check
found	u16*	returns true if the reference is found
finding	u16*	returns true if the reference find is ongoing

Parameter Discussion

axisOrVectorSpace selects an axis (NIMC_AXIS1 through NIMC_AXIS30), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously checking multiple axes or vector spaces, the **axisOrVSMap** parameter indicates which axes or vector spaces to check. On motion controllers that support fewer than thirty axes, configuring non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

axisOrVSMap is the bitmap of axes or vector spaces to check. It is only required when multiple axes or vector spaces are selected with the **axisOrVectorSpace** parameter. NI-Motion ignores additional axes and vector spaces if you select non-existent axes.

When starting axes (**axisOrVectorSpace** = NIMC_AXIS_CTRL):

D15 .	•••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Axis . 15		Axis 10	Axis 9	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1	0

For D1 through D15:

1 = Start axis

0 = Do not start axis

When starting vector spaces (**axisOrVectorSpace** = NIMC_VECTOR_SPACE_CTRL):

D15		D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
VS 15	••••	VS 10	VS 9	VS 8	VS 7	VS 6	VS 5	VS 4	VS 3	VS 2	VS 1	0

For D1 through D15:

1 = Start vector space

0 = Do not start vector space

To wait on a single axis or vector space, set **axisOrVectorSpace** to the specified axis or vector space. The **axisOrVSMap** bitmap is ignored.

To wait on multiple axes, the **axisOrVectorSpace** selector is set to NIMC_AXIS_CTRL and the **axisOrVSMap** bitmap defines the axes to

start. Similarly, to wait on multiple vector spaces, **axisOrVectorSpace** is set to NIMC_VECTOR_SPACE_CTRL and the **axisOrVSMap** bitmap defines the vector spaces to start.

M

Note It is not possible to combine a Check Reference on an axis and a vector space in a single use of this function. To accomplish this, create a single axis vector space and then execute a multicoordinate space Check Reference.

found is a returned Boolean value that indicates the success of the search or searches the function is waiting for. It does not matter if different searches are being executed on different axes. When all searches are complete on all the axes or vector spaces specified, **found** is true if all of the searches were successful. If any search did not complete successfully, **found** is false.

finding is a returned Boolean value that indicates the status of the search. **finding** returns true if the find reference is actively searching for the reference, and false if the find reference has stopped.

Using This Function

This function indicates if the reference on the indicated axes is found. There is no need to specify what type of find to wait for because the controller knows what searches are being done on what axes. This function waits for and reports on the success of the current or last executed find on an axis, vector space, or a given set of axes or vector spaces as a whole.

If a <u>Find Reference</u> is executed with the <u>Run Sequence</u> option the Check Reference function does not require an **axisOrVectorSpace** or **axisOrVSMap**. When waiting on a Run Sequence, the Check Reference function returns the status of the entire sequence. The **found** Boolean returns True only if and when all of the operations in the sequence are successful.

Wait until the **finding** parameter is false before checking the status of the **found** parameter. The status of the **found** parameter is undefined when **finding** is true. The <u>Wait Reference</u> function does this for you, but you cannot execute any other functions while Wait Reference is running.

Example

You initiated a <u>Find Home</u> on Axis 1, and <u>Find Center</u> on Axis 2 and Axis 3, and you want to check if all axes have found their reference. Simply call Check Reference with the following parameters:

axisOrVectorSpace: Axis Control (NIMC_AXIS_CTRL)

axisOrVSMap: 00001110 (axis 1, 2, & 3 enabled)

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

You cannot execute this function on multiple axes using the NIMC_AXIS_EX_CTRL resource parameter with axes sixteen through thirty.

flex_read_reference_status

Device Compatibility

Read Reference Status

Usage

status = flex_read_reference_status(u8 boardID, u8 axisOrVectorSpace, u16 axisOrVSMap, u16 attribute, u8 returnVector);

Purpose

Reads the currently selected reference status for the given set of axes or coordinate (vector) space.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space selector
axisOrVSMap	u16	bitmap of axes or vector spaces to start
attribute	u16	type of reference to read
returnVector	u8	destination for the return data

Parameter Discussion

axisOrVectorSpace can select an axis (NIMC_AXIS1 through NIMC_AXIS30), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously reading multiple axes or vector spaces, the **axisOrVSMap** parameter indicates which axes or vector spaces are involved. On motion controllers that support fewer than thirty axes, reading non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

axisOrVSMap is the bitmap of axes or vector spaces to read. It is only required when multiple axes or vector spaces are selected with the **axisOrVectorSpace** parameter. NI-Motion ignores additional axes and vector spaces if you select non-existent axes.

attribute is the type of reference on which to read the status.

Attribute	Number	Description
NIMC_HOME_FOUND	0x0	returns True if the last Find Home executed successfully
NIMC_INDEX_FOUND	0x1	returns True if the last Find Index executed successfully
NIMC_CENTER_FOUND	0x2	returns True if the last Find Center executed successfully
NIMC_FORWARD_LIMIT_FOUND	0x3	returns True if the last Find Forward Limit executed successfully
NIMC_REVERSE_LIMIT_FOUND	0x4	returns True if the last Find Reverse Limit executed successfully
NIMC_REFERENCE_FOUND	0x5	returns True is the last

You can choose the following attributes:

		Find Reference executed successfully
NIMC_CURRENT_SEQUENCE_PHASE	0x6	returns the current phase of a sequence that is currently executing
NIMC_FINDING_REFERENCE	0x7	returns True if a Find Reference is currently executing

returnVector indicates the desired destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and do not return data (0). Refer to <u>Input and</u> <u>Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function reads the currently selected reference status for the given set of axes or coordinate (vector) space.

If multiple axes in a coordinate space are selected using the axis bitmap, all axes must have successfully found their last reference for NIMC_HOME_FOUND, NIMC_INDEX_FOUND, NIMC_CENTER_FOUND, NIMC_FORWARD_LIMIT_FOUND, NIMC_REVERSE_LIMIT_FOUND, and NIMC_REFERENCE_FOUND to be true.

If any of the selected axes are currently executing a Find Reference, NIMC_FINDING_REFERENCE is true. Wait until NIMC_FINDING_REFERENCE is false before checking the status of any other attribute. The status of the NIMC_HOME_FOUND, NIMC_INDEX_FOUND, NIMC_CENTER_FOUND, NIMC_FORWARD_LIMIT_FOUND, NIMC_REVERSE_LIMIT_FOUND, and NIMC_REFERENCE_FOUND attributes are undefined when NIMC_FINDING_REFERENCE is true. The <u>Wait Reference</u> function does this for you, but you cannot execute any other functions while Wait Reference is running. flex_read_reference_status_rtn Device Compatibility

Read Reference Status Return

Usage

status = flex_read_reference_status_rtn(u8 boardID, u8 axisOrVectorSpace, u16
axisOrVSMap, u16 attribute, u16* value);

Purpose

Reads the currently selected reference status for the given set of axes or coordinate (vector) space.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrVectorSpace	u8	axis or vector space selector
axisOrVSMap	u16	bitmap of axes or vector spaces to start
attribute	u16	type of reference to read
value	u16*	is the output for attribute .

Parameter Discussion

axisOrVectorSpace can select an axis (NIMC_AXIS1 through NIMC_AXIS30), vector space (NIMC_VECTOR_SPACE1 through NIMC_VECTOR_SPACE15), multiple axes (NIMC_AXIS_CTRL), or multiple vector spaces (NIMC_VECTOR_SPACE_CTRL). When simultaneously reading multiple axes or vector spaces, the **axisOrVSMap** parameter indicates which axes or vector spaces are involved. On motion controllers that support fewer than thirty axes, reading non-existent axes or vector spaces returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Vector Spaces</u> for axis and vector space resource IDs.

axisOrVSMap is the bitmap of axes or vector spaces to read. It is only required when multiple axes or vector spaces are selected with the **axisOrVectorSpace** parameter. NI-Motion ignores additional axes and vector spaces if you select non-existent axes.

attribute is the type of reference on which to read the status.

Attribute	Number	Description
NIMC_HOME_FOUND	0x0	returns True if the last <u>Find Home</u> executed successfully
NIMC_INDEX_FOUND	0x1	returns True if the last Find Index executed successfully
NIMC_CENTER_FOUND	0x2	returns True if the last Find Center executed successfully
NIMC_FORWARD_LIMIT_FOUND	0x3	returns True if the last Find Forward Limit executed successfully
NIMC_REVERSE_LIMIT_FOUND	0x4	returns True if the last Find Reverse Limit executed successfully
NIMC_REFERENCE_FOUND	0x5	returns True is the last

You can choose the following attributes:

		Find Reference executed successfully
NIMC_CURRENT_SEQUENCE_PHASE	0x6	returns the current phase of a sequence that is currently executing
NIMC_FINDING_REFERENCE	0x7	returns True if a Find Reference is currently executing

value is the output for attribute
Using This Function

This function reads the currently selected reference status for the given set of axes or coordinate (vector) space.

If multiple axes in a coordinate space are selected using the axis bitmap, all axes must have successfully found their last reference for NIMC_HOME_FOUND, NIMC_INDEX_FOUND, NIMC_CENTER_FOUND, NIMC_FORWARD_LIMIT_FOUND, NIMC_REVERSE_LIMIT_FOUND, and NIMC_REFERENCE_FOUND to be true.

If any of the selected axes are currently executing a Find Reference, NIMC_FINDING_REFERENCE is true.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

You cannot execute this function on multiple axes using the NIMC_AXIS_EX_CTRL resource parameter with axes sixteen through thirty.

Load Reference Parameter

Usage

status = flex_load_reference_parameter(u8 boardID, u8 axis, u8 findType, u16 attribute, f64 value);

Purpose

Loads the value for the specified find reference parameter.

Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description						
boardID	u8	assigned by Measurement & Automation Explorer (MAX)						
axis	u8	axis to configure						
findType	u8	sets the type of find reference search for which to load the attribute						
attribute	u16	the attribute to load						
value	f64	the input for the attribute you are loading						

Parameter Discussion

axis is the axis to configure with this function. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

findType sets the type of find reference search for which to load the attribute. Valid inputs are as follows:

- NIMC_FIND_HOME_REFERENCE
- NIMC_FIND_INDEX_REFERENCE
- NIMC_FIND_CENTER_REFERENCE
- NIMC_FIND_FORWARD_LIMIT_REFERENCE
- NIMC_FIND_REVERSE_LIMIT_REFERENCE

attribute is the attribute to load. The following table includes valid **attributes**.

Attribute	Number
NIMC_INITIAL_SEARCH_DIRECTION	0x0
NIMC_FINAL_APPROACH_DIRECTION	0x1
NIMC_EDGE_TO_STOP_ON	0x2
NIMC_SMART_ENABLE	0x3
NIMC_ENABLE_RESET_POSITION	0x4
NIMC_OFFSET_POSITION	0x5
NIMC_PRIMARY_RESET_POSITION	0x6
NIMC_SECONDARY_RESET_POSITION	0x7
NIMC_APPROACH_VELOCITY_PERCENT	0x8
NIMC_SEQUENCE_SEARCH_ORDER	0x9
NIMC_ENABLE_SEARCH_DISTANCE	0xA
NIMC_SEARCH_DISTANCE	0xB
NIMC_PHASE_A_REFERENCE_STATE	0xC
NIMC_PHASE_B_REFERENCE_STATE	0xD

value is the input for the attribute you are loading. Valid values for each

attribute are listed in the following section.

Using This Function

Use this function to load the parameters for <u>Find Reference</u>. The following are valid values for each **attribute**:

- Initial Search Direction—Enter true to search in the reverse direction. Enter false (default) to search in the forward direction.
- **Final Approach Direction**—Valid values are true or false (default). The following table lists the behavior of the Final Approach Direction parameter for each type of find:

Find Type	Final Approach Direction is True	Final Approach Direction is False
Find Home	reverse approach	forward approach
Find Center Find Forward Limit Find Reverse Limit	opposite to direction of travel into limit	same as the direction of travel into limit
Find Index	N/A	N/A

- Note For Find Center, this value refers to how the controller behaves when approaching and marking the position of the limits to find their center.
- Edge to Stop On—Applies only to Find Home. If true, reverse edge. If false (default), forward edge.
- **Approach Velocity Percent**—The percent of the loaded velocity that a given axis uses to perform final adjustments. Refer to the *Remarks* section for valid values.

Find Index uses this % velocity throughout the entire search routine.

• **Smart Enable**—If true, appropriately enables or disables the limits and home input before executing a find. The limits and home input

are returned to their original state after the find operation is complete. To determine what state **Smart Enable** sets the limits or home input to, reference the appropriate **Search Type** description. If false, you must manually enable or disable the limits or home input.

- Enable Reset Position—If true, after successfully locating the given reference, it resets primary and secondary positions to values indicated by Primary Reset Position and Secondary Reset Position. The reset is performed after the offset move if one is requested. If the find operation fails, the position is not reset. If false, no reset is performed.
- Offset Position—Performs an offset move of a given distance, in counts (steps), after successfully locating the given reference, but before resetting the position if a reset is requested. Valid inputs range from -2³¹-1 to 2³¹-1. An offset of 0 is equivalent to not performing an offset move. If the find operation fails, the offset move does not occur.
- **Primary Reset Position**—The position to reset the primary position to. Will only reset if **Enable Reset Position** is true. Valid values range from $-2^{31}-1$ to $2^{31}-1$.
- Secondary Reset Position—The position to reset the secondary position to. Will only reset if **Enable Reset Position** is true. Valid values range from $-2^{31}-1$ to $2^{31}-1$.
- Sequence Search Order—The order number for a find sequence.
 Valid values range from 0 to 2¹⁶–1. Refer to the <u>Run Sequence</u> description for more information.
- Enable Search Distance—If true, Find Reference uses the value loaded for Search Distance when searching for an index. If false, Find Reference uses a set distance of 1 1/16th revolution. The default is false.
- **Search Distance**—The distance Find Reference searches for an index before giving up. Valid values range from 0 to 2³¹–1 counts.
- **Phase A Reference State**—The logical state of encoder phase A when an index is found. Use this in conjunction with the encoder polarity to configure the criteria for index status. Enter 1 for inactive or 0 for active.

• **Phase B Reference State**—The logical state of encoder phase B when an index is found. Use this in conjunction with the encoder polarity to configure the criteria for index status. Enter 1 for inactive or 0 for active.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- For the NI 73xx controllers, NIMC_APPROACH_VELOCITY_PERCENT is a single-precision floating-point value from 0.4% to 150%. This value directly scales the programmed velocity. The default value is 20%.
- The resolution of NIMC_APPROACH_VELOCITY_PERCENT is approximately 0.4%.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- NIMC_SECONDARY_RESET_POSITION is not supported by the NI SoftMotion Controller.
- NIMC_PHASE_A_REFERENCE_STATE and NIMC_PHASE_B_REFERENCE_STATE are not supported on CANopen devices.
- NIMC_PRIMARY_RESET_POSITION must be set to 0 when you are using the NI SoftMotion Controller for the Copley Controls CANopen drives, Accelnet and Xenus.
- The NI SoftMotion Controller supports the full 64-bit floating point range for NIMC_APPROACH_VELOCITY_PERCENT.
- The NI SoftMotion Controller requires that **Search Distance**, and thus, **Enable Search Distance** is always true (1).

Get Reference Parameter

Usage

status = flex_get_reference_parameter(u8 boardID, u8 axis, u8 findType, u16 attribute, f64* value);

Purpose

Gets the value for the specified find reference parameter.



Parameters

Name	Туре	Description					
boardID	u8	assigned by Measurement & Automation Explorer (MAX)					
axis	u8	axis to configure					
findType	u8	sets the type of find reference search for which to get the attribute					
attribute	u16	the attribute to get					
value	f64*	the value output for attribute					

Parameter Discussion

axis is the axis to configure with this function. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

findType sets the type of find reference search for which to get the attribute. Valid inputs are as follows:

- NIMC_FIND_HOME_REFERENCE
- NIMC_FIND_INDEX_REFERENCE
- NIMC_FIND_CENTER_REFERENCE
- NIMC_FIND_FORWARD_LIMIT_REFERENCE
- NIMC_FIND_REVERSE_LIMIT_REFERENCE

attribute is the attribute to get. The following table lists valid attributes:

Attribute	Number
NIMC_INITIAL_SEARCH_DIRECTION	0x0
NIMC_FINAL_APPROACH_DIRECTION	0x1
NIMC_EDGE_TO_STOP_ON	0x2
NIMC_SMART_ENABLE	0x3
NIMC_ENABLE_RESET_POSITION	0x4
NIMC_OFFSET_POSITION	0x5
NIMC_PRIMARY_RESET_POSITION	0x6
NIMC_SECONDARY_RESET_POSITION	0x7
NIMC_APPROACH_VELOCITY_PERCENT	0x8
NIMC_SEQUENCE_SEARCH_ORDER	0x9
NIMC_ENABLE_SEARCH_DISTANCE	0xA
NIMC_SEARCH_DISTANCE	0xB
NIMC_PHASE_A_REFERENCE_STATE	0xC
NIMC_PHASE_B_REFERENCE_STATE	0xD

value is the output for the attribute you are getting. Valid values for each **attribute** are listed in the following section.

Using This Function

Use this function to get the loaded parameters for <u>Find Reference</u>. The following are valid values for each **attribute**:

- Initial Search Direction—If true, search reverse. If false (default), search forward.
- **Final Approach Direction**—Valid values are true or false (default). The following table lists the behavior of the Final Approach Direction parameter for each type of find:

Find Type	Final Approach Direction is True	Final Approach Direction is False
Find Home	reverse approach	forward approach
Find Center Find Forward Limit Find Reverse Limit	opposite to direction of travel into limit	same as the direction of travel into limit
Find Index	N/A	N/A

- Note For Find Center, this value refers to how the controller behaves when approaching and marking the position of the limits to find their center.
- Edge to Stop On—Applies only to Find Home. If true, reverse edge. If false (default), forward edge.
- **Approach Velocity Percent**—The percent of the loaded velocity that a given axis uses to perform final adjustments. Refer to the *Remarks* section for valid values.

Find Index uses this % velocity throughout the entire search routine.

• **Smart Enable**—If true, appropriately enables or disables the limits and home input before executing a find. The limits and home input

are returned to their original state after the find operation is complete. To determine what state **Smart Enable** sets the limits or home input to, reference the appropriate **Search Type** description. If false, you must manually enable or disable the limits or home input.

- Enable Reset Position—If true, after successfully locating the given reference, it resets primary and secondary positions to values indicated by Primary Reset Position and Secondary Reset Position. The reset is performed after the offset move if one is requested. If the find operation fails the position is not reset. If false, no reset is performed.
- Offset Position—Performs an offset move of a given distance, in counts (steps), after successfully locating the given reference, but before resetting the position if a reset is requested. Valid inputs range from -2³¹-1 to 2³¹-1. An offset of 0 is equivalent to not performing an offset move. If the find operation fails the offset move does not occur.
- **Primary Reset Position**—The position to reset the primary position to. This parameter resets the primary position only if **Enable Reset Position** is true. Valid values range from $-2^{31}-1$ to $2^{31}-1$.
- Secondary Reset Position—The position to reset the secondary position to. Will only resets the secondary position only if Enable Reset Position is true. Valid values range from -2³¹-1 to 2³¹-1.
- Sequence Search Order—The order number for a find sequence.
 Valid values range from 0 to 2¹⁶–1. Refer to the <u>Run Sequence</u> description for more information.
- Enable Search Distance—If true, Find Reference uses the value loaded for Search Distance when searching for an index. If false (default), Find Reference uses a set distance of 1 1/16th revolution.
- **Search Distance**—The distance Find Reference searches for an index before giving up. Valid values range from 0 to 2³¹–1.
- **Phase A Reference State**—The logical state of encoder phase A when an index is found. Use this in conjunction with the encoder polarity to configure the criteria for index status. Valid values are 1 for inactive or 0 for active.

• **Phase B Reference State**—The logical state of encoder phase B when an index is found. Use this in conjunction with the encoder polarity to configure the criteria for index status. Valid values are 1 for inactive or 0 for active.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- For the NI 73xx controllers, NIMC_APPROACH_VELOCITY_PERCENT is a single-precision floating-point value from 0.4% to 150%. This value directly scales the programmed velocity. The default value is 20%.
- The resolution of NIMC_APPROACH_VELOCITY_PERCENT is approximately 0.4%.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- NIMC_SECONDARY_RESET_POSITION is not supported by the NI SoftMotion Controller.
- NIMC_PHASE_A_REFERENCE_STATE and NIMC_PHASE_B_REFERENCE_STATE are not supported on CANopen devices.
- NIMC_PRIMARY_RESET_POSITION must be set to 0 when you are using the NI SoftMotion Controller for the Copley Controls CANopen drives, Accelnet and Xenus.
- The NI SoftMotion Controller supports the full 64-bit floating point range for NIMC_APPROACH_VELOCITY_PERCENT.
- The NI SoftMotion Controller requires that **Search Distance**, and thus, **Enable Search Distance** is always true (1).

Analog & Digital I/O

You can use the analog and digital I/O functions to control the generalpurpose analog and digital I/O resources on the NI motion controller. These resources include up to 32 bits of general-purpose digital I/O, PWM outputs, RTSI lines, and any extra encoders, ADC channels, and DAC outputs that are not mapped to an axis.

The 32 bits of digital I/O are available on the Digital I/O Connector. These bits are organized into 8-bit ports that you can configure as inputs or outputs on a port-wise basis, or on a bitwise basis. Each bit has individually programmable polarity that you can configure as active low or active high. You can use the general-purpose digital I/O for system integration applications including operator panel switch inputs and outputs, relay and solenoid activation, trigger I/O between other controllers and/or instruments in the system, and so on.

You can use encoders, ADC channels, and DAC outputs that are not mapped to an axis for general-purpose I/O. Typical uses for encoder inputs include velocity monitoring, masters for master-slave gearing, and digital potentiometer applications.

You can use unused ADC inputs and DAC outputs for any analog I/O that is within their specifications. Typical analog input applications include analog joysticks, potentiometers, force, pressure, level and strain sensors, and so on. Analog output applications vary from heater element control to laser intensity modulation.

Set I/O Port Direction

Set I/O Port Polarity Set I/O Port MOMO Read I/O Port Enable ADCs Read ADCs Set ADC Range Load DAC Enable Encoders Read Encoder Position Reset Encoder Position Configure Encoder Filter Configure Encoder Polarity Configure PWM Output Load PWM Duty Cycle Select Signal

Configure Encoder Filter

Usage

status = flex_configure_encoder_filter(u8 boardID, u8 axisOrEncoder, u16
frequency);

Purpose

Selects the maximum count frequency for an encoder channel by configuring its digital filter.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
axisOrEncoder	u8	axis or encoder to configure
frequency	u16	maximum count frequency selector

Parameter Discussion

axisOrEncoder is the axis or encoder to configure. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_ENCODER1 through NIMC_ENCODER30. On motion controllers that support fewer than thirty axes, configuring non-existent axes or encoders returns error 70006 (NIMC_badResourceIDOrAxisError). For configuring encoders mapped to axes, you can call this function on the axis or directly on its mapped encoder. Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

frequency Value	Maximum Count Frequency	Max Count Frequency Constant
0	25.6 MHz	NIMC_ENCODER_FILTER_25_6MHz
1	12.8 MHz	NIMC_ENCODER_FILTER_12_8MHz
2	6.4 MHz	NIMC_ENCODER_FILTER_6_4MHz
3	3.2 MHz	NIMC_ENCODER_FILTER_3_2MHz
4	1.6 MHz (default)	NIMC_ENCODER_FILTER_1_6MHz
5	800 kHz	NIMC_ENCODER_FILTER_800KHz
6	400 kHz	NIMC_ENCODER_FILTER_400KHz
7	200 kHz	NIMC_ENCODER_FILTER_200KHz
8	100 kHz	NIMC_ENCODER_FILTER_100KHz
9	50 kHz	NIMC_ENCODER_FILTER_50KHz
10	25 Hz	NIMC_ENCODER_FILTER_25KHz

frequency selects the maximum count frequency for the specified encoder.

Using This Function

Setting the maximum allowable count frequency for an encoder is useful for reducing the effect of noise on the encoder lines. Noise on the encoder lines can be interpreted as extra encoder counts. By setting the frequency to the lowest possible setting required for the motion application, you can ensure the highest degree of accuracy in positioning. In choosing the appropriate value, you must take into account the counts per revolution of the encoder and the maximum velocity for the axis in question.

For example, with a 20,000 counts per revolution encoder and a maximum velocity of 3,000 RPM (50 revolutions per second), the encoder signal could be as high as 1,000,000 counts per second. A frequency value of 4, which would correspond to a maximum count frequency of 1.6 MHz would be appropriate in this case.

If you never call this function, a default value of 4 (1.6 MHz) is used.

Tip When the encoder filter is changed, reset the position to prevent an erroneous count.

flex_configure_encoder_polarity

Device Compatibility

Configure Encoder Polarity

Usage

i32 status = flex_configure_encoder_polarity (u8 boardID, u16 indexPolarityMap, u16 phaseAPolarityMap, u16 phaseBPolarityMap);

Purpose

Configures the encoder **Phase A**, **Phase B**, and **Index** line polarities.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
indexPolarityMap	u16	sets the polarity for encoder Index
phaseAPolarityMap	u16	sets the polarity for encoder Phase A
phaseBPolarityMap	u16	sets the polarity for encoder Phase B

Parameter Discussion

indexPolarityMap sets the polarity for encoder **Index** to active low or active high.

D8	D7	D6	D5	D4	D3	D2	D1
Encoder							
8	7	6	5	4	3	2	1

D1 through D8:

1 =Active low

0 = Active high

phaseAPolarityMap sets the polarity for encoder **Phase A** to active low or active high.

D8	D7	D6	D5	D4	D3	D2	D1
Encoder							
8	7	6	5	4	3	2	1

D1 through D8:

1 =Active low

0 =Active high

phaseBPolarityMap sets the polarity for encoder **Phase B** to active low or active high.

D8	D7	D6	D5	D4	D3	D2	D1
Encoder							
8	1	6	5	4	3	2	1

D1 through D8:

1 =Active low

0 = Active high

Using This Function

The Configure Encoder Polarity function configures the polarities for the encoder **Phase A**, **Phase B**, and **Index** lines. Configure encoder polarities before calling the <u>Enable Encoders</u> function.

You can configure the encoder polarity as active low or active high. When configured as active low, the input is active when there is a low signal on the input pin. Conversely, active high means that the input is active when there is a high signal on the input pin.

Note Setting Phase A and Phase B incorrectly causes the feedback to appear to move in the opposite direction. For example, if Phase A or Phase B is the wrong polarity when moving in the forward direction, the position appears to move in the reverse direction. The opposite occurs when moving in the reverse direction. If both are set incorrectly, motion occurs in the correct direction but the index might not work correctly.

flex_configure_pwm_output

Device Compatibility
Configure PWM Output

Usage

status = flex_configure_pwm_output(u8 boardID, u8 PWMOutput, u16 enable, u16 clock);

Purpose

Enables and disables PWM outputs, and sets the PWM clock frequency.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
PWMOutput	u8	PWM Output
enable	u16	enable/disable for PWM Output
clock	u16	clock selector

Parameter Discussion

PWMOutput selects the PWM Output to configure (1 or 2).

enable enables or disables the specified PWM Output. When enabled, the clock parameter determines the clock frequency used for the PWM output.

- 1 = enabled
- 0 = disabled

clock specifies the clock frequency for the PWM output. The base clock frequency for the PWM outputs is 24.81 MHz on the 7340 and 7330 controllers, 24.58 MHz on the 7350 controller, and 20.48 on 7334/42/44 controllers. This base clock frequency is divided down depending on the clock value selected. The following table lists the PWM clock frequency settings:

Clock Divider Value	7334/42/44 Frequency	7330/40 Frequency	7350 Frequency	PWM Frequency S Constant
512	40.00 kHz	48.46 kHz	48.01 kHz	NIMC_PWM_FREQ_SCA
1K	20.00 kHz	24.23 kHz	24.00 kHz	NIMC_PWM_FREQ_SCA
2K	10.00 kHz	12.11 kHz	12.00 kHz	NIMC_PWM_FREQ_SCA
4K	5.00 kHz	6.06 kHz	6.00 kHz	NIMC_PWM_FREQ_SCA
8K	2.50 kHz	3.03 kHz	3.00 kHz	NIMC_PWM_FREQ_SCA
16K	1.25 kHz	1.51 kHz	1.50 kHz	NIMC_PWM_FREQ_SCA (5)
33K	625.00 Hz	757.14 Hz	750.09 Hz	NIMC_PWM_FREQ_SCA (6)
External Clock/ 256 Hz	External Clock/ 256 Hz	External Clock/ 256 Hz	External Clock/ 256 Hz	NIMC_PWM_FREQ_EXT (7)
65K	312.50 Hz	378.57 Hz	375.05 Hz	NIMC_PWM_FREQ_SCA (8)
131K	156.25 Hz	189.29 Hz	187.52 Hz	NIMC_PWM_FREQ_SCA (9)
262K	78.13 Hz	94.64 Hz	93.76 Hz	NIMC_PWM_FREQ_SCA

				(10)
524K	39.06 Hz	47.32 Hz	46.88 Hz	NIMC_PWM_FREQ_SCA (11)
1048K	19.53 Hz	23.66 Hz	23.44 Hz	NIMC_PWM_FREQ_SCA (12)
2097K	9.77 Hz	11.83 Hz	11.72 Hz	NIMC_PWM_FREQ_SCA (13)
4194K	4.88 Hz	5.92 Hz	5.86 Hz	NIMC_PWM_FREQ_SCA (14)
External Clock/ 32,768 Hz	External Clock/ 32,768 Hz	External Clock/ 32,768 Hz	External Clock/ 32,768 Hz	NIMC_PWM_FREQ_EXT (15)

Using This Function

The PWM outputs on the NI motion controller are digital pulse-train outputs that have a frequency specified by the clock parameter of this function and a duty cycle specified by the <u>Load PWM Duty Cycle</u> function. These outputs can be used to control devices that require a PWM input, such as a laser whose intensity is controlled by a PWM signal, or can be used to generate isolated analog outputs by passing the PWM output through an optocoupler, and then filtering the digital pulse train to produce an analog output voltage.

When you configure a PWM output, the clock frequency applies to both PWM outputs. If you configure one PWM output for a clock value of 3, and then the second PWM output for a clock value of 4, the value of 4 applies to both PWM outputs.

The only exception is when the clock settings for the two PWM outputs are 0 and 8, 1 and 9, 2 and 10, and so on, in which case each output has a different frequency. This is because clock values 0–7 and 8–15 are paired. Clock values 0 and 8 use the common time base of 10.24 MHz (12.8 MHz on the 7350). The frequency of clock value 0 is found by dividing the time base by 256, while clock value 8 is found by dividing the time base by 32768. If you have an NI 7344 controller and you switch channel B to clock value 10, the time base changes to 2.56 MHz. This change also switches channel A to 2.56/256 = 10 kHz. Therefore, if the value in one set changes, it switches the other channel to the pairing clock value. If the two values are in the same group, the two channels have the same frequency.

To use an external clock (clock values of 7 or 15), connect the external clock signal to the PCLK input on the Digital I/O connector.

flex_enable_adcs

Device Compatibility

Enable ADCs

Usage

status = flex_enable_adcs(u8 boardID, u8 reserved, u16 ADCMap);

Purpose

Enables one or more of the unmapped ADC channels.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
reserved	u8	unused input
ADCMap	u16	bitmap of ADCs to enable

Parameter Discussion

reserved is an unused input. The input value is ignored.

ADCMap is the bitmap of ADC channels to enable:

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	۵
0	 0	0	0	ADC 8	ADC 7	ADC 6	ADC 5	ADC 4	ADC 3	ADC 2	1

D0 through D7:

1 = ADC channel enabled (default)

0 = ADC channel disabled

Using This Function

This function enables one or more independent ADC channels for use as general-purpose analog inputs. The motion controller returns an error if you attempt to enable or disable and ADC that is being used as an axis feedback when the axis is enabled. These feedback channels are automatically enabled/disabled when you enable or disable their corresponding axis with the Enable Axis function. You must first disable the axis using analog feedback, then enable or disable ADCs. Bit locations corresponding to mapped ADC channels are ignored.

The NI-Motion Analog-to-Digital Converter (ADC) multiplexes between channels with a scan rate of approximately 40 μ s for the 7340 motion controller, and 25 μ s for the 7350 motion controller. Therefore, the time between samples for a specific ADC channel is as follows:

7330/40:

ADC sample time = $40 \mu s$ /channel x (number of enabled channels)

7350:

ADC sample time = $25 \mu s/channel x$ (number of enabled channels)

By default, all channels are enabled at power up. You must disable unused channels to increase the scan rate and decrease the sample time.

The 40 μs /channel scan rate is fast enough to support analog feedback at the fastest PID update rates as long as no additional ADC channels are enabled.

Example

To enable ADC channels 1, 3, 5, and 7 on the 7350 motion controller, call the Enable ADCs function with **ADCMap** = 0x0055, which corresponds to the following bitmap:

D15	••••	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	[
0	••••	0	0	0	ADC 8	ADC 7	ADC 6	ADC 5	ADC 4	ADC 3	ADC 2	ŀ
0	••••	0	0	0	0	1	0	1	0	1	0	1

Under normal conditions, because ADC channels 2, 4, 6, and 8 are set to zero (0) they are disabled when you execute this function. However, if

ADC channel 2 is already being used as feedback for axis 2, the disable (0) for ADC 2 is ignored, resulting in a modal error and the following bitmap of enabled ADCs:

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	0
0	 0	0	0	ADC 8	ADC 7	ADC 6	ADC 5	ADC 4	ADC 3	ADC 2	A
0	 0	0	0	0	1	0	1	0	1	1	1

In this example there are five ADCs enabled, so the sample time for each ADC channel is as follows:

ADC sample time = $40 \mu s/channel \times 5 = 200 \mu s$

The ADC sample time puts a limit on the fastest PID update rate practically achievable. You can set a faster PID update rate with the Enable Axis function, but the PID loop does not truly operate at that faster rate because the ADC channels used as feedback are not being sampled fast enough.

flex_enable_encoders

Device Compatibility

Enable Encoders

Usage

status = flex_enable_encoders(u8 boardID, u16 encoderMap);

Purpose

Enables one or more of the unmapped encoder resources.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
encoderMap	u16	bitmap of encoders to enable

Parameter Discussion

encoderMap is the bitmap of encoder resources to enable:

D15	•••	D10	D9	D8	D7	D6	D5	D4	D3	D2	C
Enc 15		Enc 10	Enc 9	Enc 8	Enc 7	Enc 6	Enc 5	Enc 4	Enc 3	Enc 2	E

D1 through D15:

1 = Encoder enabled

0 = Encoder disabled (default)

Using This Function

This function enables one or more independent encoder channels for use as general-purpose encoder inputs. It has no effect on encoders that are mapped to axes and being used for axis feedback. These feedback encoders are automatically enabled/disabled when their corresponding axis is enabled or disabled with the <u>Enable Axis</u> function. Bit locations corresponding to mapped encoders are ignored, as are encoder resources that do not exist. For example, encoders 5 through 8 are ignored if you are using a controller that supports a maximum of four axes.

Typical uses for independent encoder inputs include velocity monitoring, masters for master-slave gearing, and digital potentiometer applications.

Example

To enable encoders 3 and 4 on an NI motion controller, call the Enable Encoders function with **encoderMap** = 0x0018, which corresponds to the following bitmap:

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	C
Enc 15	 Enc 10	Enc 9	Enc 8	Enc 7	Enc 6	Enc 5	Enc 4	Enc 3	Enc 2	E
0	 0	0	0	0	0	0	1	1	0	0

Normally, because encoders 1 and 2 are set to zero (0), they is disabled by this function execution. However, if encoder 2 is already being used as feedback for axis 2 and axis 1 is not being used, the disable (0) for Enc 2 is ignored resulting in the following bitmap of enabled encoders:

D15	 D10	D9	D8	D7	D6	D5	D4	D3	D2	С
Enc 15	 Enc 10	Enc 9	Enc 8	Enc 7	Enc 6	Enc 5	Enc 4	Enc 3	Enc 2	E
0	 0	0	0	0	0	0	1	1	1	0

There is a limit on the number of enabled encoders supportable at the faster update rates. Attempting to enable too many encoders generates an error. Refer to the <u>Enable Axis</u> function for more information about update rate limitations.

flex_load_dac

Device Compatibility

Load DAC

Usage

status = flex_load_dac(u8 boardID, u8 DAC, i16 outputValue, u8 inputVector);

Purpose

Loads an output value to an unmapped DAC resource.

Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
DAC	u8	DAC to control
outputValue	i16	value sent to the DAC
inputVector	u8	source of the data for this function

Parameter Discussion

DAC is the DAC to control. Valid values are NIMC_DAC1 through NIMC_DAC30. On motion controllers that support fewer than thirty axes, configuring non-existent DACs returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>DAC Outputs</u> for DAC resource IDs.

outputValue is the value sent to the DAC. The parameter range is - 32,768 to +32,767, corresponding to the full ± 10 V output range.



Note DAC torque limits and offsets do not apply when directly loading a DAC.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to Input and Return Vectors for more detailed information.

Using This Function

This function is used to send a value directly to an unmapped DAC resource. DACs not mapped as servo axis outputs are available for general-purpose analog out applications.



Caution You must not execute this function on a DAC mapped to an axis. Doing so causes the DAC output to glitch momentarily before returning to axis control.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following includes considerations you must make when you are using this function with the NI SoftMotion Controller:

If the axis is enabled, the control loop has ownership of the drive command output, and the value that this function sends is ignored.

flex_load_pwm_duty

Device Compatibility

Load PWM Duty Cycle

Usage

status = flex_load_pwm_duty(u8 boardID, u8 PWMOutput, u16 dutyCycle, u8
inputVector);

Purpose

Sets the duty cycle for a PWM output.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
PWMOutput	u8	PWM Output
dutyCycle	u16	duty cycle for PWM Output
inputVector	u8	source of the data for this function

Parameter Discussion

PWMOutput selects the PWM Output to control (1 or 2).

dutyCycle is a value between 0 and 255 that specifies the amount of time the PWM output is high.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

The **dutyCycle** determines the amount of time the PWM output is high. A **dutyCycle** of 0 corresponds to a 0 V output, and a **dutyCycle** of 255 corresponds to a pulse train that is high for 255/256 = 99.6% of the time. Use the <u>Configure PWM Output</u> function to set the frequency of the PWM output signal.

You can set the duty cycle before or after configuring a PWM output. By default, **dutyCycle** is 0, so if you call the Configure PWM Output function to configure a PWM output, the output is low until you set **dutyCycle** differently. If you set the **dutyCycle** first, the PWM output reflects this **dutyCycle** immediately after calling the Configure PWM Output function.

flex_read_adc16

Device Compatibility

Read ADCs

Usage

status = flex_read_adc16(u8 boardID, u8 ADC, u8 returnVector);

Purpose

Reads the converted value from an ADC input channel.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
ADC	u8	ADC channel to read
returnVector	u8	destination for the return data
Parameter Discussion

ADC is the Analog-to-Digital Converter channel to read. Valid ADC resources are NIMC_ADC1 through NIMC_ADC30. On motion controllers that support fewer than thirty axes, reading non-existent ADCs returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>ADC Channels</u> for ADC resource IDs.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and don't return data (0). Refer to <u>Input and Return</u> <u>Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns the converted voltage from any of the analog input channels. You can only read values from channels that have been either directly enabled by the <u>Enable ADCs</u> function or automatically enabled by being mapped to an enabled axis.

For an ADC channel mapped to an axis, this function returns the actual ADC value. In contrast, the <u>Read Position</u> function executed on the owner axis returns an ADC value that has been offset by a reset value stored when the <u>Reset Position</u> function was executed. ADC channels are never internally reset so their DC values are preserved.

flex_read_adc16_rtn

Device Compatibility

Read ADCs Return

Usage

status = flex_read_adc16_rtn(u8 boardID, u8 ADC, i32* ADCValue);

Purpose

Reads the converted value from an ADC input channel.

Parameters

Name	Туре	Description						
boardID	u8	assigned by Measurement & Automation Explorer (MAX)						
ADC	u8	ADC channel to read						
ADCValue	i32*	the converted analog value						

Parameter Discussion

ADC is the Analog-to-Digital Converter channel to read. Valid ADC resources are NIMC_ADC1 through NIMC_ADC30. On motion controllers that support fewer than thirty axes, reading non-existent ADCs returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>ADC Channels</u> for ADC resource IDs.

ADCValue is the signed 16-bit (7350) or 12-bit (all others) value from the ADC channel. The voltage range is set through the <u>Set ADC Range</u> function.

	Analog Values						
Range Values	7350	7330/40	NI SoftMotion Controller for CANopen— Xenus				
0 to 5	0 to +65,535	0 to +4,095	—				
–5 to +5	–32,768 to +32,767	–2,048 to +2,047					
0 to +10	0 to +65,535	0 to +4,095					
–10 to +10 (default)	-32,768 to +32,767	-2,048 to +2,047	-2,048 to +2,047				

Using This Function

This function returns the converted voltage from any of the analog input channels. You can only read values from channels that have been either directly enabled by the <u>Enable ADCs</u> function or automatically enabled by being mapped to an enabled axis.

For an ADC channel mapped to an axis, this function returns the actual ADC value. In contrast, the <u>Read Position</u> function executed on the owner axis returns an ADC value that has been offset by a reset value stored when the <u>Reset Position</u> function was executed. ADC channels are never internally reset so their DC values are preserved.

flex_read_encoder

Device Compatibility

Read Encoder Position

Usage

status = flex_read_encoder(u8 boardID, u8 axisOrEncoder, u8 returnVector);

Purpose

Reads the position of an encoder.

Parameters

Name	Туре	Description				
boardID	u8	assigned by Measurement & Automation Explorer (MAX)				
axisOrEncoder	u8	axis or encoder to read				
returnVector	u8	destination for the return data				

Parameter Discussion

axisOrEncoder is the axis or encoder to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_ENCODER1 through NIMC_ENCODER30. On motion controllers that support fewer than thirty axes, reading non-existent axes or encoders returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function returns the quadrature count value of the encoder selected. The encoder must be enabled, either directly through the <u>Enable</u> <u>Encoders</u> function or automatically, by being mapped to an enabled axis.

The Read Encoder Position function is typically used to read the value of an encoder that is not part of an axis. This encoder could be a master encoder used for master-slave gearing or an independent position or velocity sensor.

For reading encoders mapped to axes, you can call this function on the axis or directly on its mapped encoder. For servo axes, both approaches return the same value as the <u>Read Position</u> function. On stepper axes however, this function can return additional useful information.

During axis setup, you can operate the closed-loop stepper axis in openloop mode and use this function to directly measure the counts per revolution and steps per revolution for the axis. These values must be loaded in advance for subsequent closed-loop operation. Refer to the Load Counts/Steps per Revolution function for more information.

You also can use this function to return a finer reading of position in cases where the encoder resolution greatly exceeds the step resolution of the closed-loop stepper axis.

flex_read_encoder_rtn

Device Compatibility

Read Encoder Position Return

Usage

status = flex_read_encoder_rtn(u8 boardID, u8 axisOrEncoder, i32*
encoderCounts);

Purpose

Reads the position of an encoder.

Parameters

Name	Туре	Description				
boardID	u8	assigned by Measurement & Automation Explorer (MAX)				
axisOrEncoder	u8	axis or encoder to read				
encoderCounts	i32*	encoder position in quadrature counts				

Parameter Discussion

axisOrEncoder is the axis or encoder to read. Valid values are NIMC_AXIS1 through NIMC_AXIS30 or NIMC_ENCODER1 through NIMC_ENCODER30. On motion controllers that support fewer than thirty axes, reading non-existent axes or encoders returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> and <u>Encoders</u> for axis and encoder resource IDs.

encoderCounts is the encoder position is quadrature counts.

Using This Function

This function returns the quadrature count value of the encoder selected. The encoder must be enabled, either directly through the <u>Enable</u> <u>Encoders</u> function or automatically, by being mapped to an enabled axis.

The Read Encoder Position function is typically used to read the value of an encoder that is not part of an axis. This encoder could be a master encoder used for master-slave gearing or an independent position or velocity sensor.

For reading encoders mapped to axes, you can call this function on the axis or directly on its mapped encoder. For servo axes, both approaches return the same value as the <u>Read Position</u> function. On stepper axes however, this function can return additional useful information.

During axis setup, you can operate the closed-loop stepper axis in openloop mode and use this function to directly measure the counts per revolution and steps per revolution for the axis. These values must be loaded in advance for subsequent closed-loop operation. Refer to the Load Counts/Steps per Revolution function for more information.

You also can use this function to return a finer reading of position in cases where the encoder resolution greatly exceeds the step resolution of the closed-loop stepper axis.

flex_read_port

Device Compatibility

Read I/O Port

Usage

status = flex_read_port(u8 boardID, u8 port, u8 returnVector);

Purpose

Reads the logical state of the bits in an I/O port.

Parameters

Name	Туре	Description					
boardID u8		assigned by Measurement & Automation Explorer (MAX)					
port	u8	general-purpose I/O port to read					
returnVector	u8	destination for the return data					

Parameter Discussion

port is the general-purpose I/O port. Valid values are NIMC_IO_PORT1 through NIMC_IO_PORT8 for 7330/40/50 motion controllers, NIMC_DIGITAL_INPUT_PORT1 through NIMC_DIGITAL_INPUT_PORT4 or NIMC_DIGITAL_OUTPUT_PORT1 through NIMC_DIGITAL_OUTPUT_PORT4 for the PCI-7390 motion controller, or the RTSI software port (NIMC_RTSI_PORT) to read. On motion controllers that support fewer than eight axes, reading non-existent ports returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to 73xx Controller General-Purpose I/O Port IDs and NI SoftMotion Controller General-Purpose I/O Port IDs for I/O port resource IDs and constants.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.

 $\overline{\mathbb{N}}$

Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function reads the logical state of the bits in the general-purpose I/O port selected. You can execute this function at any time to monitor the signals connected to an input port. Reads of ports configured as outputs return the last value written to the port with the <u>Set I/O Port MOMO</u> function.

The PCI-7390 has dedicated direction I/O ports. Refer to <u>73xx Controller</u> <u>General-Purpose I/O Port IDs</u> for the appropriate general-purpose I/O port digital input and output resource constants and IDs to use for **port**.

For example, to read the state of input port 3, **port** is NIMC_DIGITAL_INPUT_PORT3. To read the value previously set on output port 2, **port** is NIMC_DIGITAL_OUTPUT_PORT2.

Note This function reads the logical state (On or Off, True or False) of the bits in a port. The polarity of the bits in the port determines if an On state is active high or active low. Refer to the Set I/O Port Polarity function for more information.

When reading the RTSI port, the value read is the latched data, so you can detect active pulses on the RTSI bus. After reading the latched data value, the function resets the latch. Use the <u>Set I/O Port Polarity</u> function to specify the polarity, and therefore the active state for latching.



Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

flex_read_port_rtn

Device Compatibility

Read I/O Port Return

Usage

status = flex_read_port_rtn(u8 boardID, u8 port, u16* portData);

Purpose

Reads the logical state of the bits in an I/O port.



Parameters

Name	Туре	Description						
boardID	u8	assigned by Measurement & Automation Explorer (MAX)						
port	u8	general-purpose I/O port to read						
portData	u16*	bitmap of the logical state of the I/O port						

Parameter Discussion

port is the general-purpose I/O port. Valid values are NIMC_IO_PORT1 through NIMC_IO_PORT8 for 7330/40/50 motion controllers, NIMC_DIGITAL_INPUT_PORT1 through NIMC_DIGITAL_INPUT_PORT4 or NIMC_DIGITAL_OUTPUT_PORT1 through NIMC_DIGITAL_OUTPUT_PORT4 for the PCI-7390 motion controller, NIMC_DIGITAL_INPUT_PORT1 through NIMC_DIGITAL_INPUT_PORT30 or NIMC_DIGITAL_OUTPUT_PORT1 through NIMC_DIGITAL_OUTPUT_PORT1 through NIMC_DIGITAL_OUTPUT_PORT30 for the NI SoftMotion Controller, or RTSI software port (NIMC_RTSI_PORT) to control. Refer to 73xx Controller General-Purpose I/O Port IDs and NI SoftMotion Controller

<u>General-Purpose I/O Port IDs</u> for I/O port resource IDs and constants.

portData is the bitmap of the logical state of the I/O port.

D15		D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
XXX	••••	XXX	XXX	XXX	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

For D0 through D7:

1 = I/O bit True (On)

0 = I/O bit False (Off)

Using This Function

This function reads the logical state of the bits in the general-purpose I/O port selected. You can execute this function at anytime to monitor the signals connected to an input port. Reads of ports configured as outputs return the last value written to the port with the <u>Set I/O Port MOMO</u> function.



Note This function reads the logical state (On or Off, True or False) of the bits in a port. The polarity of the bits in the port determines if an On state is active high or active low. Refer to the <u>Set I/O Port Polarity</u> function for more information.

When reading the RTSI port, the value read is the latched data, so you can detect active pulses on the RTSI bus. After reading the latched data value, the function resets the latch. Use the <u>Set I/O Port Polarity</u> function to specify the polarity, and therefore the active state for latching.



Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following includes considerations you must make when you are using this function with a 73xx controller:

The PCI-7390 has dedicated direction I/O ports. Refer to <u>73xx Controller</u> <u>General-Purpose I/O Port IDs</u> for the appropriate general-purpose I/O port digital input and output resource constants and IDs to use for **port**.

For example, to read the state of input port 3, **port** is NIMC_DIGITAL_INPUT_PORT3. To read the value previously set on output port 2, **port** is NIMC_DIGITAL_OUTPUT_PORT2.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

 The NI SoftMotion Controller has dedicated direction I/O ports. Refer to <u>NI SoftMotion Controller General-Purpose I/O Port IDs</u> for the appropriate general-purpose I/O port digital input and output resource constants and IDs to use for **port**.

For example, to read the state of input port 3, **port** is NIMC_DIGITAL_INPUT_PORT3. To read the value previously set on output port 2, **port** is NIMC_DIGITAL_OUTPUT_PORT2.

• The RTSI port is not supported by the NI SoftMotion Controller.

flex_reset_encoder

Device Compatibility

Reset Encoder Position

Usage

status = flex_reset_encoder(u8 boardID, u8 encoder, i32 position, u8
inputVector);

Purpose

Resets the position of an unmapped encoder to the specified value.
Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
encoder	u8	encoder to reset
position	i32	reset value for encoder
inputVector	u8	source of the data for this function

Parameter Discussion

encoder is encoder to reset. Valid values are NIMC_ENCODER1 through NIMC_ENCODER30. On motion controllers that support fewer than thirty axes, configuring non-existent encoders returns error 20006 (NIMC_badBosourceIDOrAvisError). Pefor to Encoders for encoder

70006 (NIMC_badResourceIDOrAxisError). Refer to Encoders for encoder resource IDs.

position is the reset value for the encoder resource. You can reset position to any value in the total position range of $-(2^{31})$ to $+(2^{31}-1)$.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to Input and Return Vectors for more detailed information.

Using This Function

The Reset Encoder Position function resets the position of the selected encoder. You can reset position to zero or to any value in the 32-bit position range. Normally, this function is only used on independent encoders that are not mapped to axes. For encoders mapped to axes, you must use the <u>Reset Position</u> function instead.



Note Attempting to reset an encoder that is mapped to an axis generates an error.

Encoder position can be reset at any time. However, it is recommended that you reset position only while the encoder is stopped. A encoder reset while it is moving does not have a repeatable reference position.



Note Non-zero reset values are useful for defining a position reference offset.

Enabled breakpoints are automatically disabled when you execute a Reset Position or Reset Encoder Position function on the corresponding axis. flex_select_signal

Device Compatibility

Select Signal

Usage

status = flex_select_signal(u8 boardID, u16 destination, u16 source);

Purpose

Specifies the source and destination for various motion signals, including trigger inputs, high-speed capture circuits, breakpoint outputs, RTSI lines, and RTSI software ports.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
destination	u16	destination of signal
source	u16	source of signal

Parameter Discussion

destination is the destination of the signal coming from source.

source is the source of the signal to route to destination. For a destination value of NIMC_HS_CAPTURE1 through NIMC_HS_CAPTURE8, the valid source values are as follows:

source	Comments
NIMC_RTSI0 through NIMC_RTSI7	RTSI lines 0 through 7
NIMC_PXI_STAR_TRIGGER	PXI star trigger line
NIMC_TRIGGER_INPUT	Trigger input for the corresponding axis

Note You must route signals from the RTSI lines *before* you <u>enable high-speed capture</u>.

For a destination value of NIMC_RTSI0 through NIMC_RTSI7 (Value 0-7), the valid source values are as follows:

source	Comments
NIMC_BREAKPOINT1 through NIMC_BREAKPOINT8	Breakpoint outputs
NIMC_RTSI_SOFTWARE_PORT	Corresponding bit in RTSI software port
NIMC_DONT_DRIVE	Sets RTSI pin back to input state
NIMC_PHASE_A1 through NIMC_PHASE_A8	'A' phase of encoder 1 through 8
NIMC_PHASE_B1 through NIMC_PHASE_B8	'B' phase of encoder 1 through 8
NIMC_INDEX1 through NIMC_INDEX8	Index signal of encoder 1 through 8

Using This Function

When the destination is NIMC_RTSI0 through NIMC_RTSI7 or NIMC_PXI_STAR_TRIGGER, the motion controller drives the RTSI line as an output. When the destination is NIMC_HS_CAPTURE1 through NIMC_HS_CAPTURE8, the RTSI line serves as an input for the high-speed capture circuitry. This function is used to set the direction of the RTSI lines. The RTSI lines can always be read using the Read I/O Port function, regardless of the way they are currently configured.

To manually set the state of the RTSI lines, set the source for the specified RTSI line to be NIMC_RTSI_SOFTWARE_PORT. Use the <u>Set I/O</u> <u>Port MOMO</u> function to set the state of the lines.

The signal seen on the RTSI 4 pin is a high pulse of 120 to 150 ns duration. The action specified in the <u>Enable Breakpoint Output</u> function only applies to the breakpoint output pin on the motion I/O connector, not to RTSI pins.

Example 1

To use the signal coming in on RTSI pin 3 to trigger the high-speed capture on encoder/axis 1, call Select Signal as follows:

flex_select_signal (boardID, NIMC_HS_CAPTURE1, NIMC_RTSI3)

The polarity of the high-speed capture input is specified by the <u>Configure</u> <u>High-Speed Capture</u> function.

Example 2

To output the breakpoint signal for axis 2 on RTSI pin 4, call Select Signal as follows:

flex_select_signal (boardID, NIMC_RTSI4, NIMC_BREAKPOINT2)

Example 3

To drive RTSI pin 5 with the corresponding bit (bit 5) of the RTSI software port, call Select Signal as follows:

flex_select_signal (boardID, NIMC_RTSI5, NIMC_RTSI_SOFTWARE_PORT)

To set the state of the RTSI software port, use the <u>Set I/O Port MOMO</u> function.

Example 4

When writing to the RTSI software port by using the <u>Set I/O Port MOMO</u> function, only those RTSI lines that have been configured to control by the RTSI software port are affected. To set the RTSI line back to an input, call Select Signal as follows:

flex_select_signal (boardID, NIMC_RTSI5, NIMC_DONT_DRIVE)

flex_set_adc_range

Device Compatibility

Set ADC Range

Usage

status = flex_set_adc_range(u8 boardID, u8 ADC, u16 range);

Purpose

Sets the voltage range for the analog to digital converters, on a perchannel basis.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
ADC	u8	ADC channel to configure
range	u16	the voltage range for the specified ADC

Parameter Discussion

ADC is the analog-to-digital converter channel to configure. Valid ADC resources are NIMC_ADC1 through NIMC_ADC30. On motion controllers that support fewer than thirty axes, configuring non-existent ADC channels returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to ADC Channels for ADC resource IDs.

range specifies the input voltage range over which the ADC converts input voltages to digital values. Voltages outside of the range clamps at the extremes, which are -2048 or +2047 (NI 7330/40) and -32,768 to +32,767 (NI 7350) for the -5 to +5 V and -10 to +10 V ranges, and 0 to 4,096 (NI 7330/40) and 0 to +65,535 (NI 7350) for the 0 to +5 V and 0 to +10 V ranges. You can choose from the following values for the range:

Danga Valuas	Binary Values					
Rallye values	7350	7330/40				
0 to 5	0 to +65,535	0 to +4,095				
–5 to +5	-32,768 to +32,767	-2,048 to +2,047				
0 to +10	0 to +65,535	0 to +4,095				
-10 to +10 (default)	-32,768 to +32,767	-2,048 to +2,047				

The constants listed previously are defined in the NI-Motion header files MotnCnst.h (for C/C++ users) and motncnst.bas (for Visual Basic users).

Using This Function

If you do not call this function, the range defaults to -10 to +10 V. If you know that the input voltage falls within a more restrictive range, you can effectively increase the resolution of the measurements by selecting an appropriate range from the previous list.



Note With the 7350 motion controller, changing the range of an ADC changes all other ADCs to the same range. Other NI motion controllers can set the range independent of other ADCs.

For example, if you are using a 7340 motion controller, the input signal ranges from –3 to +3 V, and you select the –5 to +5 V range, the 4,096 discrete values for the ADC is 2.44 mV apart instead of the 4.88 mV apart when using the –10 to +10 V range. If you are using a 7350 motion controller, the input signal ranges from –3 to +3 V, and you select the –5 to +5 V range, the 65,536 discrete values for the ADC is 152 μ V apart instead of the 305 μ V apart when using the –10 to +10 V range.

ADC ranges cannot be changed on any ADC while an axis that is using analog feedback is enabled. You must first disable the axis using analog feedback, then change the range. Refer to the <u>Read ADC</u> and <u>Enable ADCs</u> functions for more information.

flex_set_port_direction

Device Compatibility

Set I/O Port Direction

Usage

status = flex_set_port_direction(u8 boardID, u8 port, u16 directionMap);

Purpose

Sets the direction of a general-purpose I/O port as input or output.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
port	u8	general-purpose I/O port to control
directionMap	u16	port direction control

Parameter Discussion

port is the general-purpose I/O port (NIMC_IO_PORT1 through NIMC_IO_PORT8) to control. Refer to <u>73xx Controller General-Purpose</u> I/O Port IDs for general-purpose I/O port resource IDs.

directionMap is the bitmap of directions for the bits in the I/O port:

D15	••••	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
XXX		XXX	XXX	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

For D0 through D7:

1 = Input (default)

0 = Output

Using This Function

This function configures the bits in a general-purpose I/O port as input or output. After setting the direction, use the <u>Read I/O Port</u> function to read the port, the <u>Set I/O Port MOMO</u> function to write to the port, and the <u>Set I/O Port Polarity</u> function to set the polarity of each bit in the port to active high or active low.



Note The direction of bits in the RTSI software port (NIMC_RTSI_PORT) is controlled with the <u>Select Signal</u> function.

flex_set_port

Device Compatibility

Set I/O Port MOMO

Usage

status = flex_set_port(u8 boardID, u8 port, u8 mustOn, u8 mustOff, u8
inputVector);

Purpose

Sets an I/O port value using the MustOn/MustOff protocol.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
port	u8	general-purpose I/O port to control
mustOn	u8	bitmap of I/O port bits to force on
mustOff	u8	bitmap of I/O port bits to force off
inputVector	u8	source of the data for this function

Parameter Discussion

port is the general-purpose I/O port. Valid values are NIMC_IO_PORT1 through NIMC_IO_PORT8 for 7330/40/50 motion controllers,

NIMC_DIGITAL_OUTPUT_PORT1 through

NIMC_DIGITAL_OUTPUT_PORT4 for the PCI-7390 motion controller, NIMC_DIGITAL_OUTPUT_PORT1 through

NIMC_DIGITAL_OUTPUT_PORT30 for the NI SoftMotion Controller, or RTSI software port (NIMC_RTSI_PORT) to control. Refer to <u>73xx</u> <u>Controller General-Purpose I/O Port IDs</u> and <u>NI SoftMotion Controller</u> <u>General-Purpose I/O Port IDs</u> for I/O port resource IDs and constants.

mustOn is the bitmap of I/O port bits to force on:

D7	D6	D5	D4	D3	D2	D1
mustOn 7	mustOn 6	mustOn 5	mustOn 4	mustOn 3	mustOn 2	must

For D0 through D7:

1 = I/O bit forced to logical On

0 = I/O bit left unchanged (default)

mustOff is the bitmap of I/O port bits to force off:

D7	D6	D5	D4	D3	D2	D1
mustOff 7	mustOff 6	mustOff 5	mustOff 4	mustOff 3	mustOff 2	mus

For D0 through D7:

1 = I/O bit forced to logical Off

0 = I/O bit left unchanged (default)

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to Input and Return Vectors for more detailed information.

Using This Function

This function sets the logical state of bits in the general-purpose I/O port selected.

Using the MustOn/MustOff protocol allows you to set or reset individual bits without affecting other output bits in the port. This gives you tri-state control over each bit: On, Off or Unchanged. A one (1) in a bit location of the **MustOn** bitmap turns the bit On, while a one (1) in the corresponding location of the **MustOff** bitmap turns the bit Off. A zero (0) in either bitmap has no effect, so leaving both the **MustOn** and **MustOff** bits at zero is effectively a hold and the state of the bit is unchanged. If you set both the **MustOn** and **MustOff** bits to one (1), it is interpreted as a **MustOn** condition and the bit is turned On.



Note This function sets the logical state of a bit On or Off (True or False). The polarity of the bits in the port determines if an On state is active high or active low. Refer to the <u>Set I/O Port Polarity</u> function for more information.

Note When executing this function in a program, there may be a delay of up to 0.5 ms before the output lines are physically updated. The program flow continues, even though the output may not have been updated yet.



Note If the motion controller is programmed to use the port for output when it has been configured for input, the action is ignored.

The Set I/O Port MOMO function allows individual control of generalpurpose output bits without requiring a shadow value or a read of the port to remember the state of other bits not being set or reset with the function.

Example

 $\mathbf{\widehat{v}}$ **Tip** This section applies to all NI motion controllers.

In I/O port 2, to set bits 1 and 3 On, bits 0 and 5 Off and to leave the other bits (2, 4, 6, and 7) unchanged, call this function with the following parameters:

port = NIMC_IO_PORT2

mustOn = 0x0A, where the value 0x0A corresponds to the following bitmap:

D7	D6	D5	D4	D3	D2	D1
mustOff 7	mustOff 6	mustOff 5	mustOff 4	mustOff 3	mustOff 2	mus
0	0	0	0	1	0	1

mustOff = 0x21, where the value 0x21 corresponds to the following bitmap:

D7	D6	D5	D4	D3	D2	D1
mustOff 7	mustOff 6	mustOff 5	mustOff 4	mustOff 3	mustOff 2	mus
0	0	1	0	0	0	0

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- A line configured for input is not affected by this function.
- You can always write to the RTSI software port, but the actual RTSI lines on the physical RTSI port are only affected if the RTSI line has been configured properly by using the <u>Select Signal</u> function. By default, none of the RTSI lines are configured to output their corresponding bits in the RTSI software port; you must configure each RTSI line individually using the Select Signal function, rather than the <u>Set I/O Port Direction</u> function.
- The PCI-7390 has dedicated direction I/O ports. Refer to <u>73xx</u> <u>Controller General-Purpose I/O Port IDs</u> for the appropriate general-purpose I/O port digital input and output resource constants and IDs to use for **port**.

For example, to set the state of output port 2, **port** is NIMC_DIGITAL_OUTPUT_PORT2. To set the state of input port 3, **port** is NIMC_DIGITAL_INPUT_PORT3.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- Refer to <u>General-Purpose I/O Ports</u> for information about how the NI SoftMotion Controller supports digital input and output lines for each axis.
- For **inputVector**, the NI SoftMotion Controller supports only the immediate vector (0XFF).
- The NI SoftMotion Controller has dedicated direction I/O ports. Refer to <u>NI SoftMotion Controller General-Purpose I/O Port IDs</u> for the appropriate general-purpose I/O port digital input and output resource constants and IDs to use for **port**.

For example, to set the state of output port 2, **port** is NIMC_DIGITAL_OUTPUT_PORT2. To set the state of input port 3, **port** is NIMC_DIGITAL_INPUT_PORT3.

• The RTSI port is not supported by the NI SoftMotion Controller.

flex_set_port_pol

Device Compatibility

Set I/O Port Polarity

Usage

status = flex_set_port_pol(u8 boardID, u8 port, u16 portPolarityMap);

Purpose

Sets the bit polarity in a general-purpose I/O port.



Parameters

Name Type		Description			
boardID	u8	assigned by Measurement & Automation Explorer (MAX)			
port	u8	general-purpose I/O port to control			
portPolarityMap	u16	bitmap of active polarities			

Parameter Discussion

port is the general-purpose I/O port. Valid values are NIMC_IO_PORT1 through NIMC_IO_PORT8 for 7330/40/50 motion controllers, NIMC_DIGITAL_INPUT_PORT1 through NIMC_DIGITAL_INPUT_PORT4 or NIMC_DIGITAL_OUTPUT_PORT1 through NIMC_DIGITAL_OUTPUT_PORT4 for the PCI-7390 motion controller, NIMC_DIGITAL_INPUT_PORT1 through NIMC_DIGITAL_INPUT_PORT30 or NIMC_DIGITAL_OUTPUT_PORT1 through NIMC_DIGITAL_OUTPUT_PORT1 through NIMC_DIGITAL_OUTPUT_PORT30 for the NI SoftMotion Controller, or RTSI software port (NIMC_RTSI_PORT) to control. Refer to <u>73xx</u>

<u>Controller General-Purpose I/O Port IDs</u> and <u>NI SoftMotion Controller</u> <u>General-Purpose I/O Port IDs</u> for I/O port resource IDs and constants.

portPolarityMap is the bitmap of active polarities for the I/O port:

D15		D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	••••	0	0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

For D0 through D7:

1 = active low/active open (default)

0 = active high/active closed
Using This Function

This function sets the polarity (active state) of the general-purpose I/O port on an individual bit basis.

When configured as active low, the input or output is active when there is a low signal on the pin. Conversely, active high means that the input or output is active when there is a high signal on the pin.

Configuring an active state of active open or active closed does not correspond to the level of the signal on the input or output pin. Instead, an active open state means that the input or output is active when current is not flowing through the optocoupled input. Conversely, an active closed state means that the input or output is active when current is flowing through the optocoupled input.

Typically, ports and their pins are configured for direction and polarity at initialization. After configuration, you can then read or write logical states (True or False, On or Off) to ports without worrying about the physical states of signals on the port pins.

 $\overline{\mathbb{N}}$

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

• The PCI-7390 has dedicated direction I/O ports. Refer to <u>73xx</u> <u>Controller General-Purpose I/O Port IDs</u> for the appropriate general-purpose I/O port digital input and output resource constants and IDs to use for **port**.

For example, to configure the polarity of the output port on axis 2, **port** is NIMC_DIGITAL_OUTPUT_PORT2. To configure the polarity of the input port on axis 3, **port** is NIMC_DIGITAL_INPUT_PORT3.

• The polarity also defines the latching behavior for the RTSI port. To detect short pulses on RTSI lines, the hardware latches activegoing signals and holds that state until the port is read. For example, if you configure a bit for active low polarity, a transition from high to low is latched until read, even if the signal goes high again. If the signal starts low, it also is latched until read, even if the signal is high when you read the bit.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

• The NI SoftMotion Controller has dedicated direction I/O ports. Refer to <u>NI SoftMotion Controller General-Purpose I/O Port IDs</u> for the appropriate general-purpose I/O port digital input and output resource constants and IDs to use for **port**.

For example, to configure the polarity of the output port on axis 2, **port** is NIMC_DIGITAL_OUTPUT_PORT2. To configure the polarity of the input port on axis 3, **port** is NIMC_DIGITAL_INPUT_PORT3.

• The RTSI port is not supported by the NI SoftMotion Controller.

Error & Utility

You can use the error handling functions and utility functions to get information about a motion controller. Refer to <u>Errors and Error Handling</u> for more information about errors.

Get Motion Board Information

<u>Get u32</u>

<u>Set u32</u>

Read Error Message

Get Last Error

Get Error Description

Device Compatibility

Get Error Description

Usage

status = flex_get_error_description(u16 descriptionType, i32 errorCode, u16 commandID, u16 resourceID, i8* charArray, u32* sizeOfArray);

Purpose

Gets an error, command, and/or resource description string as an ASCII character array.

Parameters

Name	Туре	Description	
descriptionType u16		type of description selector	
errorCode	i32	error code	
commandID	u16	command ID number	
resourceID	u16	resource ID number	
charArray	[i8]*	character array	
sizeOfArray	u32*	size of character array	

Parameter Discussion

descriptionType is the selector for the type of description string to return.

descriptionType Constant	descriptionType Value
NIMC_ERROR_ONLY	0
NIMC_FUNCTION_NAME_ONLY	1
NIMC_RESOURCE_NAME_ONLY	2
NIMC_COMBINED_DESCRIPTION	3

errorCode is an error code from a function return status or the error code returned from the Read Error Message function.

commandID is the command ID of a function.

resourceID is the resource ID of an axis, vector space, encoder, ADC, DAC, or other resource.

charArray is an array of ASCII characters containing the error, command, and/or resource description string. This function places all or part of the selected string in charArray, if sizeOfArray is greater than zero (> 0).

sizeOfArray is the number of characters in the description plus one for the NULL string terminator. As an input, this I/O parameter specifies the size of the allocated array. If **sizeOfArray** and/or **charArray** is NULL or zero (0), the required size of the array (not including the NULL terminator) is returned in the **sizeOfArray** parameter as an output.

Using This Function

This function returns the selected description string as an ASCII character array. You must allocate space for this array on the host computer before calling this function. You can use this function to generate a string for displaying a function name, a resource name, an error code description, or a complete error description string in response to an error code returned as a function status or the result of calling the <u>Read Error Message</u> function.

Not all input parameters are required for each description type. The following parameters are required to return an accurate description string:

descriptionType	errorCode	commandID	resourcel
NIMC_ERROR_ONLY	required	not required	not required
NIMC_FUNCTION_NAME_ONLY	not required	required	not required
NIMC_RESOURCE_NAME_ONLY	not required	required	required
NIMC_COMBINED_DESCRIPTION	required	required	required

Because resource IDs are not unique (for example, axis 1 and program 1 both are resource 1), the command ID is required to set the context and allow this function to generate the proper resource name string.

If NULL (or 0) is passed in either the **charArray** or **sizeOfArray** parameters, the required size of the array (not including the NULL terminator) is returned in the **sizeOfArray** parameter. You can use this feature when you want to allocate only the memory necessary to hold the description string. This function is then called twice: one time to get the required array size, and again to actually retrieve the description.

The number of characters required for the character array is always one more than the actual number of characters in the controller name due to the NULL terminator at the end of the string. If **sizeOfArray** is smaller than the actual description string, this function returns a partial string with the last three characters replaced by "..." to indicate that the string is not complete.

Example

After executing a <u>Find Reference Index</u> sequence on axis 1, a modal error is detected. A call to the Read Error Message function returns the following set of parameters:

commandID = 334 resourceID = 0x01 errorCode = -70124

To generate an error description string for display, call the Get Error Description function with these parameters, plus a **descriptionType**, **sizeOfArray** = 0 and **charArray** = NULL. When the function returns, **sizeOfArray** has the size of the description in it. Allocate memory for a character array of size **sizeOfArray** + 1. Call the Get Error Description function a second time passing in the same parameters as before except **sizeOfArray** is the value of **sizeOfArray** + 1 returned by the first function call, and **charArray** points to the character array just allocated. This function returns the following strings, depending upon the **descriptionType** selected:

descriptionType Constant	String
NIMC_ERROR_ONLY	Error –70124 (NIMC_findIndexError); Find Index sequence did not find the index successfully
NIMC_FUNCTION_NAME_ONLY	Find Index (flex_find_index)
NIMC_RESOURCE_NAME_ONLY	Axis 0x01
NIMC_COMBINED_DESCRIPTION	Error –70124 (NIMC_findIndexError) occurred in Find Index (flex_find_index) on Axis 0x01; Find Index sequence did not find the index successfully

flex_get_last_error

Device Compatibility

Get Last Error

Usage

status = flex_get_last_error(u8 boardID, u16* commandID, u16* resourceID, i32* errorCode);

Purpose

Gets detailed information about the last error generated by a high-level NI-Motion function in the course of executing other NI-Motion functions.

Parameters

Name	Туре	Description		
boardID	u8	assigned by Measurement & Automation Explorer (MAX)		
commandID	u16*	command ID number		
resourceID	u16*	resource ID number		
errorCode	i32*	error code		

Parameter Discussion

errorCode is the error code value, as defined in the file motnErr.h in the NI-Motion include directory.

commandID is the command ID of the function that generated the error.

resourceID is the resource (e.g. axis, vector space, ADC, etc.) that generated the error.

Using This Function

If the <u>Initialize</u> function returns an error, you can call this function to determine the specific NI-Motion function and resource that generated the error. You can then call the <u>Get Error Description</u> function, which formats a string for you with the function name, resource ID and error description.

Device Compatibility

Get Motion Board Information

Usage

status = flex_get_motion_board_info(u8 boardID, u32 informationType, u32*
informationValue);

Purpose

Gets information about the properties and features of the motion controller.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description	
boardID	u8	assigned by Measurement & Automation Explorer (MAX)	
informationType	u32	type of information you want to retrieve	
informationValue	u32*	retrieved information	

Parameter Discussion

informationType is the selector for the type of controller information you want. Legal values are defined as constants in the NI-Motion header files MotnCnst.h (for C/C++ users) and motncnst.bas (for Visual Basic users) and are listed in the following section.

informationValue is the returned information of the type you selected.

The following table shows possible return values for each of the valid **informationType** values.

informationType	informationValue	
NIMC_BOARD_FAMILY (1100)	NIMC_NI_MOTION (0)	
NIMC_BOARD_TYPE (1120)	PCI_7344 (28) PCI_7340 (61) PXI_7344 (27) PXI_7340 (62) PCI_7342 (37) PCI_7330 (63) PCI_7334 (32) PXI_7330 (64) PXI_7342 (36) PCI_7390 (71) PXI_7350 (35) FW_7344 (41) PCI_7350 (34) SOFTMOTION PXI_7334 (25)	
NIMC_BUS_TYPE (1130)	NIMC_PCI_BUS (1) NIMC_PXI_BUS (2) NIMC_1394_BUS (3) NIMC_CAN_BUS (4)	
NIMC_NUM_AXES (1510)	Number of axes on the control	
NIMC_FIRMWARE_VERSION (3020)	Version-build code (MMmmbbl Note To read this information, you must convert it to hexadecima format.	
NIMC_DSP_VERSION (3030)	Version-build code (MMmmbbl	
NIMC_FPGA_VERSION (3040)	Version-build code (MMmmbbl	
NIMC_FPGA2_VERSION (3050)	Version-build code (MMmmbbl	
NIMC_CONTROLLER_SERIAL_NUMBER (2040)	Controller serial number	

NIMC_LOCAL_OR_REMOTE (2050)	NIMC_TRUE (1) (remote) NIMC_FALSE (0) (local)
NIMC_DRIVER_VERSION (2060)	Version-build code (MMmmbbl

Using This Function

This function returns selected information about NI motion controllers including controller type and family, bus type, number of axes, and so on.

NI-Motion also has four information types for retrieving the version numbers and release dates of the firmware segments loaded in the onboard Flash ROM. All firmware segments are field upgradable using the **Update Firmware** option in Measurement & Automation Explorer. Versions are returned in a version-build code format:

Version-build code = MMmmbbbb, where *MM* = the major version number, *mm* = the minor version number, and *bbbb* = the build number.

You can use this information to verify that the NI motion controller has the latest firmware downloaded on it.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- The NI SoftMotion Controller does not support the following **informationValue** returned values:
 - NIMC_FIRMWARE_VERSION
 - NIMC_DSP_VERSION
 - NIMC_FPGA_VERSION
 - NIMC_FPGA2_VERSION

Get Motion Board Name

Usage

status = flex_get_motion_board_name(u8 boardID, i8* charArray, u32*
sizeOfArray);

Purpose

Gets the motion controller name as an ASCII character array.

Parameters

Name	Туре	Description		
boardID	u8	assigned by Measurement & Automation Explorer (MAX)		
charArray	[i8]*	character array		
sizeOfArray	u32*	size of character array		

Parameter Discussion

sizeOfArray is the number of characters in the controller name plus one for the NULL string terminator. As an input, this I/O parameter specifies the size of the allocated array. If **sizeOfArray** is insufficient, a NIMC_insufficientSizeError is returned as the status of the function, and the required character array size (including space for the NULL terminator) is returned in the **sizeOfArray** parameter. If **sizeOfArray** is sufficient, a NIMC_noError is returned as the status of the function, the name is copied into the character array, and the number of bytes copied (plus one for the NULL terminator) is returned in the **sizeOfArray**

charArray is an array of ASCII characters containing the name of the controller. The NI-Motion software places the name of the controller, referenced by **boardID**, in the character array, if there is sufficient space. You must allocate space for this array before calling this function.

Using This Function

This function returns the name of the motion controller as an ASCII character array. You must allocate space for this array on the host computer before calling this function.

If NULL (or 0) is passed in the **charArray** parameter, the size of a character array required to hold the controller name is returned in the **sizeOfArray** parameter. You can use this feature when you want to allocate only the memory necessary to hold the controller name. This function is then called twice: one time to get the required array size, and again to actually retrieve the name.

The number of characters required for the character array is always one more than the actual number of characters in the controller name due to the NULL terminator at the end of the string. For example, the controller name PXI-7344 is eight characters long, so you must provide a 9-byte character array to hold this name. **sizeOfArray** must be nine or greater as an input, and upon successful copy of the controller name, a value of nine is placed in **sizeOfArray**.

flex_getu32

Device Compatibility

Get u32

Usage

status = flex_getu32(u8 boardID, u8 resource, u16 attribute, u32* value);

Purpose

Gets the general software settings.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
resource	u8	resource selector
attribute	u16	type of data you want to retrieve
value	u32*	retrieved data

Parameter Discussion

resource specifies the resource from which you are retrieving information.

attribute is the selector for the type of information you want. Legal values are defined as constants in the NI-Motion header files MotnCnst.h (for C/C++ users) and motncnst.bas (for Visual Basic users) and are listed in the following section.

value is the returned data of the type you selected.

Using This Function

This function returns selected data about software settings. This function can retrieve values that are stored as integers.

Name	Attribute	Description	Attribute Range
NIMC_BP_WINDOW	0x0200	After the axis encounters a breakpoint, the axis must move outside the breakpoint window before it can rearm for another breakpoint. This is used to prevent oscillation from sending multiple breakpoint outputs from the same breakpoint position. The default is 0.	0 to 255
NIMC_PROGRAM_ AUTOSTART	0x0300	Use this attribute to return the onboard program that is enabled for auto start.	
NIMC_GEARING_ ENABLED_STATUS	0x0301	Use this attribute to read the gearing enable status on an individual axis or on axes one through eight using the axis control resource. Reading the status on axes one through eight returns a bitmap where bit 0 is the status for axis 1	For each bit in the bitmap: 1 = gearing enabled 0 = gearing disabled
	and so on. Reading the status on an individual axis returns a bitmap that is masked for that axis.		
--	---	--	
--	---	--	

flex_read_error_msg_rtn

Device Compatibility

Read Error Message Return

Usage

status = flex_read_error_msg_rtn(u8 boardID, u16* commandID, u16*
resourceID, i32* errorCode);

Purpose

Reads the most recent modal error from the Error Message Stack.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
commandID	u16*	command ID number
resourceID	u16*	resource ID number
errorCode	i32*	error code

Parameter Discussion

commandID is the command ID of the function that caused the error. **resourceID** is the resource ID involved in the error. **errorCode** is the code for the error condition.

Using This Function

This function retrieves the most recent modal error from the controller and returns the command ID and resource ID that caused the error, along with the error code.



Note See <u>Error Codes</u> for a description of error codes and possible causes.

When a modal error occurs, the command ID, resource ID, and error code are automatically stored in a last-in-first-out stack and the Error Message (Err Msg) bit in the Communication Status Register is set to indicate that one or more errors are present on the stack.

Modal errors are defined as errors that are not detected at the time of function execution. Refer to <u>Errors and Error Handling</u> for a complete description.

These errors can occur for a number of reasons including: bad command ID, bad axis, vector space or resource ID, data out of range, function not valid in the present operating mode, and so on. A common source of modal errors is improperly constructed function calls stored in an onboard program. When the program is run, the errors generate modal error messages.

Because the error messages are stored in a last-in-first-out (LIFO) buffer, the most recent error is available immediately. You can read older errors with additional calls to this function. When the stack is empty, the Error Message (Err Msg) bit in the Communication Status Register is reset.

You can get a string description of the error by using the <u>Get Error</u> <u>Description</u> function.

Normally, if the application program is functioning correctly, errors are not generated. Reading the error messages from the board is useful during debugging and for handling special conditions.

Example

An application program running on the host computer monitors the Communication Status Register to check for errors. If the Error Message bit is set, the program sends a Read Error Message function to the controller and then reacts to the error information returned. Depending upon the type of error and/or the function and resource involved, the appropriate action is taken. You can check the Error Message bit again to see if any previous errors were missed.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- The **commandID** read in is always zero (0).
- The errors stack is always in FIFO mode.

flex_setu32

Device Compatibility

Set u32

Usage

status = flex_setu32(u8 boardID, u8 resource, u16 attr, u32 value);

Purpose

Sets the general software settings.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
resource	u8	resource selector
attr	u16	type of information you want to set
value	u32	new value to which attribute is set

Parameter Discussion

resource is the resource that the attribute you selected is set to.

attribute is the selector for the type of data you want to set. Legal values are defined as constants in the NI-Motion header files MotnCnst.h (for C/C++ users) and motncnst.bas (for Visual Basic users) and are listed in the following section.

value is the new value to which the data you selected is set.

Using This Function

The Set u32 function sets selected software settings as specified by attribute. This function can set values that are stored as integers.

The following table includes the attributes you can set, their descriptions, possible values, and the devices they work on.

Attribute	Resource	Constant	Description
NIMC_BP_ WINDOW	Axis or encoder	0x0200	After the axis encounters a b the axis must move outside t breakpoint window before it c for another breakpoint. This i prevent oscillation from send multiple breakpoint outputs fr same breakpoint position. Th is 0.
			Note The breakpoint v must be smaller than th breakpoint modulus or modal error is generate the breakpoint window than the loaded breakp modulus or period.
NIMC_PULL_IN_ WINDOW	Axis	0x0400	For closed-loop stepper axes around the target position the determines whether or not to more pull-in moves. Once with area, the motion controller st pull-in tries. The default is 1.
NIMC_PULL_IN_ TRIES	Axis	0x0401	The maximum number of tim closed-loop stepper axis tries the target position. It will try t within the range specified by NIMC_PULL_IN_WINDOW c reaches this maximum. The (3.
NIMC_STOP_TYPE_	Axis	0x0403	Changes the behavior of the

ON_SWITCH			 is performed when a limit swi home switch is encountered. setting applies to all axes. Caution When selecti other than NIMC_HAL? make sure that there is space between the limi and the physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate and the physical end of the axis to decelerate and the physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis to decelerate. do so could cause physical end of the axis decelerates are a site in NIMC_HALT_STOP (1), the axis performs an ir full torque/stop (halt still torque/stop (halt still torque/stop (and and actininhibit/output)
NIMC_STEP_ DUTY_CYCLE	Axis	0x0600	Duty cycle of the step output function sets the duty cycle to 25% or 50%.
NIMC_DECEL_ STOP_JERK	Axis	0x0500	 When this attribute is set to 1 turned off when any of the fo stop conditions are met: calling Stop Motion wit NIMC_DECEL_STOP a stopType encountering a softwai encountering a hardwa home switch with the S attribute NIMC_STOP_TYPE_OI set to NIMC_DECEL_S

I	1	The default value of 0 means
		The default value of o means
		deceleration uses the s-curve
		value defined in Load S-Curv

Onboard Programming

You can use Onboard Programming functions to load, execute, and save onboard programs. NI-Motion offers a set of programming functions and features that allow you to write and control autonomous programs that are completely independent from the host computer. You can execute up to 10 onboard programs simultaneously. The size and number of programs is completely flexible. It is ultimately limited by the 32 total memory objects in the Object Registry or by total available memory, whichever is reached first. Refer to <u>Begin Program Storage</u> for more information. Onboard programs run in a time-sliced manner. Refer to the <u>Onboard Programs</u> section of the *NI-Motion Help* for more information.

Onboard programs support basic math and data operation functions on general-purpose variables. Onboard programs also offer event-based functions such as Jump on Event and Wait on Event, which allow you to control program execution. Programs can even start and stop other programs.

Implementing part or all of the motion application as an onboard program or programs offloads the host computer from handling these real-time events. Onboard programs also can isolate the application from the host computer non-real-time operating system. Only bus power is required to correctly execute an onboard program after it is started.

Programs can be run from RAM or optionally saved to non-volatile Flash ROM. Saved programs are therefore available for execution at any future time, even after power cycles. Use the <u>Read Object Registry</u> function to determine the size of an onboard program. Refer to <u>Buffered Operations</u> for information about the total amount RAM and ROM on each controller.

Onboard Programming functions include <u>Object Management</u> and <u>Data</u> <u>Operations</u> functions. You can use Onboard Programming functions to begin and end program storage and to control program execution. You can use Object Management functions to organize, annotate, and save program objects to ROM. Data Operations functions include basic math functions and data operations.

Several functions are typically run from the host to control the onboard program execution, such as the Run Program, Pause/Resume, and Stop Program functions. When run as a host command, these functions interrupt the onboard program and change the program status accordingly.



Note The Onboard Programming functions are compatible with NI 7340/44 and NI 7350 controllers only.

Begin Program StorageEnd Program StorageRun ProgramPause/Resume ProgramStop ProgramInsert LabelJump to Label on ConditionSet User Status MOMORead Program StatusWait on EventLoad Program DelayLoad Program Time Slice

flex_begin_store

Device Compatibility

Begin Program Storage

Usage

status = flex_begin_store(u8 boardID, u8 program);

Purpose

Begins a program storage session.

Parameters

Name	Туре	Description	
boardID	u8	assigned by Measurement & Automation Explorer (MAX)	
program	u8	program number	

Parameter Discussion

program is the program number. Valid program numbers are 0x01 through 0xFF (1 through 255).

Using This Function

This function initiates program storage in RAM. After they are begun, all subsequent functions are stored in an object buffer and not executed until the program is run with the <u>Run Program</u> function. This memory storage continues until you execute the <u>End Program Storage</u> function. You can store only one program at a time.

The size and number of programs is completely flexible. It is ultimately limited by the 32 total memory objects in the Object Registry or by total available memory, whichever is reached first.

The 7350 and 7340 have the following onboard memory:

	RAM	ROM
7340	1 64k sector	2 64k sectors
7350	2 64k sectors	4 64k sectors

You can run programs from either RAM or ROM, but you cannot split programs between the two, and you cannot split programs between the two 64 KB ROM sectors. With an average command size of 10 bytes, a single program can be as large as 6,400 commands. For example, the 7350 and 7340 controllers can execute 10 programs simultaneously, five from RAM and five from ROM, with each program up to 1,280 commands long.



Note Attempting to store more than 32 programs generates an error. Similarly, an error is generated if you run out of memory during program storage. Both of these cases are extremely unlikely.

Do not store onboard programs while other NI-Motion functions are running. Doing so can result in incorrect program storage.

If you run two onboard program functions with the same program number, the second onboard program overrides the first. No error message is generated when this happens. flex_end_store

Device Compatibility

End Program Storage

Usage

status = flex_end_store(u8 boardID, u8 program);

Purpose

Ends a program storage session.

Parameters

Name	Туре	Description	
boardID	u8	assigned by Measurement & Automation Explorer (MAX)	
program	u8	program number	

Parameter Discussion

program is the program number. Valid program numbers are 0x01 through 0xFF (1 through 255).

Using This Function

This function ends memory storage of the program. All subsequent functions are executed normally. You can save a program to non-volatile memory (ROM) using the <u>Object Memory Management</u> function.

This function may take longer than 62 ms to process, so it is not guaranteed to be compatible with real-time execution.

flex_insert_program_label

Device Compatibility

Insert Program Label

Usage

status = flex_insert_program_label(u8 boardID, u16 labelNumber);

Purpose

Inserts a label in a program.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
labelNumber	u16	arbitrary label number

Parameter Discussion

labelNumber is any arbitrary label number from 1 to 65,535.
Using This Function

This function marks a location in the sequence of a program. The label number identifies this location and uses it in the <u>Jump on Event</u> function. Label numbers are arbitrary and do not have to follow a numerical sequence. flex_jump_on_event

Device Compatibility

Jump on Event

Usage

status = flex_jump_on_event(u8 boardID, u8 resource, u16 condition, u16 mustOn, u16 mustOff, u16 matchType, u16 labelNumber);

Purpose

Inserts a conditional jump in a program.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
resource	u8	axis control or other resource
condition	u16	qualifying condition for the jump
mustOn	u16	bitmap of bits that must be True
mustOff	u16	bitmap of bits that must be False
matchType	u16	selector for type of match required
labelNumber	u16	label number to jump to

Parameter Discussion

resource is the axis control or other resource involved in the condition. **condition** is the qualifying condition for the jump.

condition Constant	condition Value	,
		re
NIMC_CONDITION_LESS_THAN	0	N//
NIMC_CONDITION_EQUAL	1	N//
NIMC_CONDITION_LESS_THAN_OR_EQUAL	2	N//
NIMC_CONDITION_GREATER_THAN	3	N//
NIMC_CONDITION_NOT_EQUAL	4	N//
NIMC_CONDITION_GREATER_THAN_OR_EQUAL	5	N//
NIMC_CONDITION_TRUE	6	N//
NIMC_CONDITION_HOME_FOUND	7	N//
NIMC_CONDITION_INDEX_FOUND	8	N//
NIMC_CONDITION_HIGH_SPEED_CAPTURE	9	0 (; 0x2 (en
NIMC_CONDITION_POSITION_BREAKPOINT	10	0 (; 0x2 (en
Reserved	11	N//
NIMC_CONDITION_VELOCITY_THRESHOLD	12	N//
NIMC_CONDITION_MOVE_COMPLETE	13	N//
NIMC_CONDITION_PROFILE_COMPLETE	14	N//
NIMC_CONDITION_BLEND_COMPLETE	15	0 (i
NIMC_CONDITION_MOTOR_OFF	16	N//
NIMC_CONDITION_HOME_INPUT_ACTIVE	17	N//
NIMC_CONDITION_LIMIT_INPUT_ACTIVE	18	N//
NIMC_CONDITION_SOFTWARE_LIMIT_ACTIVE	19	N//
NIMC_CONDITION_PROGRAM_COMPLETE	20	prc

21

mustOn is the bitmap of bits that must be True to satisfy the condition.

XXX		XXX	XXX	mustOn 8	mustOn 7	mustOn 6	mustOn 5	mustOn
D15	••••	D10	D9	D8	D7	D6	D5	D4

For D0 through D8:

1 = Bit must be True

0 = Don't care (default)

mustOff is the bitmap of bits that must be False to satisfy the condition.

D15	••••	D10	D9	D8	D7	D6	D5	D4
XXX		XXX	XXX	mustOff 8	mustOff 7	mustOff 6	mustOff 5	mustOff

For D0 through D8:

1 = Bit must be False

0 = Don't care (default)

matchType selects the type of match required for the bitmap.

matchType Constant	matchType Value
NIMC_MATCH_ALL	0
NIMC_MATCH_ANY	1

NIMC_MATCH_ANY means that a match of any bit (logical OR) is sufficient to satisfy the condition, while NIMC_MATCH_ALL requires a complete pattern match (logical AND) of all bits.

labelNumber is the arbitrary label number to jump to. Valid label numbers are from 1 to 65,535.

Using This Function

This function controls the flow of execution in a stored program by defining a conditional jump to any label within the program. In addition to condition codes set as the result of a previous data operations function, you can test virtually any instantaneous status of axes or resources to decide if you should execute a jump.

There are two distinct groups of conditions. The first group, conditions 0 through 6, test the result of the most recent logical, mathematical or data transfer operations function. Refer to the <u>Data Operations Functions</u> for more information about mathematical or data transfer operations. These condition codes test if the result of the latest logical, mathematical, or data transfer function is less than zero, equal to zero, less than or equal to zero, greater than zero, not equal to zero, or greater than or equal to zero. For these conditions, the **resource**, **mustOn**, **mustOff**, and **matchType** parameters are not required and their values are ignored.

Note You can program unconditional jumps by setting the condition to NIMC_CONDITION_TRUE (6).

The second group, conditions 7 and above, test a specific multi-axis, multi-vector space, multi-encoder, program, motion I/O, or general-purpose I/O status. Where applicable, you can select the specified resource with the resource parameter.

NIMC_CONDITION_PROGRAM_COMPLETE is similar to the first condition group in that **mustOn**, **mustOff**, and **matchType** parameters are not required and their values are ignored. You set resource equal to the specified program number to test. The balance of the conditions in this group test status bitmaps and function similar to each other as described in the remainder of this section.

The **mustOn**, **mustOff**, and **matchType** parameters work together to define a bitmap of True and False bits that must be matched to satisfy the condition. The **matchType** parameter allows you to select between an OR match, where any matching bit is sufficient, and an AND match, where all status bits must match the True/False bitmap defined by **mustOn** and **mustOff**.

Using the MustOn/MustOff protocol gives you tri-state control over each match bit: True, False or Don't care. A one (1) in a bit location of the

MustOn bitmap sets the match bit to True, while a one (1) in the corresponding location of the MustOff bitmap resets the match bit to False. A zero (0) in either bitmap has no affect, so leaving both the MustOn and MustOff bits at zero defines the bit as Don't care. If you set both the MustOn and MustOff bits to one (1), it is interpreted as a MustOn condition and the match bit is set to True.

The NIMC_CONDITION_LIMIT_INPUT_ACTIVE and NIMC_CONDITION_SOFTWARE_LIMIT_ACTIVE conditions create a combined status bitmap where if either the forward or reverse limit is active, the bit is True.

Example

To perform a conditional jump to label 99 if either axis 3 is move complete or axis 4 is still moving (move not complete), call the Jump to Label on Condition function with the following parameters:

condition = NIMC_CONDITION_MOVE_COMPLETE (13)

mustOn = 0x08,	which corresp	ponds to the	following	bitmap:
			3	

D15		D10	D9	D8	D7	D6	D5	D4
XXX	•••	XXX	XXX	mustOn 8	mustOn 7	mustOn 6	mustOn 5	mustOn
0		0	0	0	0	0	0	0

mustOff = 0x20, which corresponds to the following bitmap:

D15	•••	D10	D8	D8	D7	D6	D5	D4
XXX	••••	XXX	XXX	mustOn 8	mustOn 7	mustOff 6	mustOff 5	mustOf
0		0	0	0	0	0	0	1

matchType = NIMC_MATCH_ANY (1)

labelNumber = 99

In this example, the move complete status of axes 1 and 2 are do not care and the **matchType** is set to match either axis 3 move complete (On) or axis 4 move not complete (Off).

flex_load_delay

Device Compatibility

Load Program Delay

Usage

status = flex_load_delay(u8 boardID, u32 delayTime);

Purpose

Loads a delay into a program sequence.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
delayTime	u32	delay time in milliseconds

Parameter Discussion

delayTime is the specified delay in milliseconds. The range is from 1 to 2^{31} -1 ms.

Using This Function

This function suspends program execution for the number of milliseconds loaded. Program execution resumes after the delay. Delays can be as short as one or two milliseconds or as long as hundreds of hours.

Load Program Time Slice

Usage

status = flex_load_program_time_slice(u8 boardID, u8 program, u16 timeSlice, u8 inputVector);

Purpose

Specifies the minimum time an onboard program has to run per watchdog period.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
program	u8	program number
timeSlice	u16	execution time per watchdog period
inputVector	u8	source of the data for this function

Parameter Discussion

program is the program number. Valid program numbers are 0x01 through 0xFF.

timeSlice is the execution time for the onboard program per watchdog period. Default is 2 ms.

inputVector indicates the source of the data for this function. Available input vectors include immediate (0xFF), variable (0x01 through 0x78), or indirect variable (0x81 through 0xF8). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

Load Program Time Slice specifies the minimum time an onboard program has to be run per watchdog period.



Note This function can only be run as a part of onboard program; the host cannot change the time slice of a program directly.

Some things to remember when changing the time slice of an onboard program:

- A total of 20 ms is allowed for all running onboard programs.
- Every onboard program loads with a default 2 ms time slice unless Load Program Time Slice is executed at the beginning of the onboard program. The default value of 2 ms is calculated based on maximum 10 onboard programs running simultaneously with equal time slice.
- You can assign different time slices for each stored onboard program as long as the total of time slice for the running onboard program does not exceed 20 ms at any given time.

For example, suppose you have the following onboard programs currently stored:

- 1. Onboard program 1, time slice = 10 ms
- 2. Onboard program 2, time slice = 10 ms
- 3. Onboard program 3, time slice = 5 ms
- 4. Onboard program 4, time slice = 4 ms
- 5. Onboard program 5, time slice = 1 ms

You could run programs 1 and 2 simultaneously without error, because the total time slices of the two running programs is 20 ms. If you then start program 3, an NIMC_invalidTimeSliceError modal error occurs because the new total time slice is 25 ms. The available time slice for an onboard program also depends on the order in which the programs are run, because a user-defined time slice takes effect only when the onboard program actually runs, and not when it is stored.

6. You can use this function to configure the performance of the onboard programs. The higher the time slice, the longer the

onboard program runs per period, and the more work the program can do. However, as one program uses more processing power, the response times of other running onboard programs decreases. You can adjust the time slices to maintain the performance of other onboard programs by running the motion system under a longer watchdog period.

Note Increasing the time slice of every onboard program using an extended watchdog period puts higher constraints on the motion system, because more time must be spent running the onboard programs.

flex_pause_prog

Device Compatibility

Pause/Resume Program

Usage

status = flex_pause_prog(u8 boardID, u8 program);

Purpose

Pauses a running program or resumes execution of a paused program.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
program	u8	program number

Parameter Discussion

program is the program number. Valid program numbers are 0x01 through 0xFF (1 through 255).

Using This Function

This function suspends execution of a running program or resumes execution of a paused program.

A program can pause or resume another program and also can pause (but not resume) itself.

 $\overline{\mathbb{N}}$

Note Pausing a program does not affect a move already started and in progress. It does not implement a <u>Stop Motion</u> function.

Any run-time (modal) error in a program automatically pauses the program in addition to generating the error message. Refer to the <u>Read</u> <u>Error Message</u> function for information about errors and error handling.

A program also can automatically pause if you execute a Stop Motion function from the host computer on an axis or axes under control of the onboard program. In these cases, the program pauses when it attempts to execute a <u>Start Motion</u> or <u>Blend Motion</u> function on the stopped axes. This automatic pause also applies when the stop is due to a limit, home, software limit, or following error condition.

You can effectively single-step through an onboard program by having the program pause after every function, and then resuming the program from the host computer.

System time does not pause when you pause a program. Pausing merely stops the next function from executing. This means that a delay counter from Load Program Delay keeps counting down even while the onboard program is paused. For example, if you load both a delay of five seconds and a pause of 10 seconds, you have a total of only 10 seconds before execution of the onboard program resumes.

flex_read_program_status

Device Compatibility

Read Program Status

Usage

status = flex_read_program_status(u8 boardID, u8 program, u8 returnVector);

Purpose

Reads the status of an onboard program.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
program	u8	program number
returnVector	u8	destination for the return data

Parameter Discussion

program is the program number. Valid program numbers are 0x01 through 0xFF (0 through 255).

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

You can use this function to determine the state of an onboard program. You can only read the state of other onboard programs from an onboard program.

Read Program Status Return

Usage

status = flex_read_program_status_rtn(u8 boardID, u8 program, u16*
programStatus);
Purpose

Reads the status of an onboard program.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
program	u8	program number
programStatus	u16*	status of specified program

Parameter Discussion

program is the program number. Valid program numbers are 0x01 through 0xFF (0 through 255).

programStatus is the status of the specified program. Possible values are as follows:

Value	Definition
0	NIMC_PROGRAM_DONE
1	NIMC_PROGRAM_PLAYING
2	NIMC_PROGRAM_PAUSED
3	NIMC_PROGRAM_STORING

Using This Function

You can use this function to determine the state of an onboard program. You can only read the state of other onboard programs from an onboard program. flex_run_prog

Device Compatibility

Run Program

Usage

status = flex_run_prog(u8 boardID, u8 program)

Purpose

Runs a previously stored program.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
program	u8	program number

Parameter Discussion

program is the program number. Valid program numbers are 0x01 through 0xFF (1 through 255).

Using This Function

This function initiates execution of the functions stored in the selected program. You can run programs out of either RAM or ROM. You can run up to ten (10) programs simultaneously.

A program can run another program but you cannot have a program run itself. Attempting to store a recursive Run Program function in a program generates an error and does not store the function. flex_set_status_momo

Device Compatibility

Set User Status MOMO

Usage

status = flex_set_status_momo(u8 boardID, u8 mustOn, u8 mustOff);

Purpose

Controls the user status bits in the Move Complete Status (MCS) register.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
mustOn	u8	bitmap of user status bits to force True
mustOff	u8	bitmap of user status bits to force False

Parameter Discussion

mustOn is the bitmap of user status bits to force True:

D7	D6	D5	D4	D3	D2	D1	D0
mustOn Sts15	mustOn Sts14	mustOn Sts13	0	0	0	0	0

D5 through D7:

1 = User status bit forced True

0 = User status bit unchanged (default)

mustOff is the bitmap of user status bits to force False:

D7	D6	D5	D4	D3	D2	D1	D0
mustOff Sts15	mustOff Sts14	mustOff Sts13	0	0	0	0	0

D5 through D7:

1 = User status bit forced False

0 = User status bit unchanged (default)

Using This Function

This function controls the upper three bits in the Move Complete Status (MCS) register using the MustOn/MustOff protocol. You can use this function in programs to report special conditions back to the host computer by setting and resetting one or more of these bits. Refer to the Read Move Complete Status function for more information about using the MCS register for high-speed polling.

Using the MustOn/MustOff protocol allows you to set or reset individual user status bits without affecting the other user status bits. This gives you tri-state control over each bit: True, False, or Unchanged. A one (1) in a bit location of the **MustOn** bitmap sets the user status bit high, while a one (1) in the corresponding location of the **MustOff** bitmap resets the user status bit low. A zero (0) in either bitmap has no affect, so leaving both the **MustOn** and **MustOff** bits at zero is effectively a hold, and the state of the user status bit is unchanged. If you set both the **MustOn** and **MustOff** bits to one (1), it is interpreted as a **MustOn** condition and the user status bit is set high.

Example

After a conditional jump in a program, you want the program to flag the host with a success code. This can be accomplished by storing the Set User Status MOMO with **mustOn** = 0xA0 and **mustOff** = 0x40. This forces user status bits 13 and 15 True and user status bit 14 low.

A subsequent poll of the MCS register returns **moveCompleteStatus** = 0xC07E, which corresponds to the following bitmap:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4
Sts 15	Sts 14	Sts 13	XXX	XXX	XXX	XXX	MC 8	MC 7	MC 6	MC 5	MC 4
1	0	1	0	0	0	0	0	0	1	1	1

flex_stop_prog

Device Compatibility

Stop Program

Usage

status = flex_stop_prog(u8 boardID, u8 program);

Purpose

Stops a running program.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
program	u8	program number

Parameter Discussion

program is the program number. Valid program numbers are 0x01 through 0xFF (1 through 255).

Using This Function

This function terminates execution of a running program. You cannot resume a stopped program but you can re-run the program from the beginning.

A program can stop another program but you cannot have a program stop itself. Attempting to store a recursive Stop Program function in a program generates an error and does not store the function.



Note Stopping a program does not affect a move already started and in progress. It does not implement a <u>Stop Motion</u> function.

flex_wait_on_event

Device Compatibility

Wait on Event

Usage

status = flex_wait_on_event(u8 boardID, u8 resource, u16 waitType, u16 condition, u16 mustOn, u16 mustOff, u16 matchType, u16 timeOut, u8 returnVector);

Purpose

Inserts a conditional wait in a program.

Parameters

Name	Туре	Description			
boardID	u8	assigned by Measurement & Automation Explorer (MAX)			
resource	u8	axis control or other resource			
waitType	u16	selector for type of wait			
condition	u16	qualifying condition to end the wait			
mustOn	u16	bitmap of bits that must be True			
mustOff	u16	bitmap of bits that must be False			
matchType	u16	selector for type of match required			
timeOut	u16	imeout value in 100 millisecond increments			
returnVector	u8	destination for the return data			

Parameter Discussion

resource is the axis control or other resource involved in the condition.

waitType is the selector for the type of wait to perform.

waitType Constant	waitType Value
NIMC_WAIT	0
NIMC_WAIT_OR	1

NIMC_WAIT_OR allows you to combine multiple, unrelated wait conditions into one wait where the program is waiting for condition 1 OR condition 2 OR condition 3 and so on.

condition is the qualifying condition to end the wait.

condition Constant	condition Value	Valid r
NIMC_CONDITION_HOME_FOUND	7	NIMC_AXI
NIMC_CONDITION_INDEX_FOUND	8	NIMC_AXI
NIMC_CONDITION_HIGH_SPEED_CAPTURE	9	NIMC_AXI or NIMC_ENC (0x20)
NIMC_CONDITION_POSITION_BREAKPOINT	10	NIMC_AXI or NIMC_ENC (0x20)
Reserved	11	N/A
NIMC_CONDITION_VELOCITY_THRESHOLD	12	NIMC_AXI
NIMC_CONDITION_MOVE_COMPLETE	13	NIMC_AXI
NIMC_CONDITION_PROFILE_COMPLETE	14	NIMC_AXI
NIMC_CONDITION_BLEND_COMPLETE	15	NIMC_AXI
NIMC_CONDITION_MOTOR_OFF	16	NIMC_AXI
NIMC_CONDITION_HOME_INPUT_ACTIVE	17	NIMC_AXI
NIMC_CONDITION_LIMIT_INPUT_ACTIVE	18	NIMC_AXI

NIMC_CONDITION_SOFTWARE_LIMIT_ACTIVE	19	NIMC_AXI
NIMC_CONDITION_PROGRAM_COMPLETE	20	program
NIMC_CONDITION_IO_PORT_MATCH	21	I/O port 1–{
NIMC_CENTER_FOUND	22	NIMC_AXI
NIMC_FORWARD_LIMIT_FOUND	23	NIMC_AXI
NIMC_REVERSE_LIMIT_FOUND	24	NIMC_AXI

Note Conditions 0 through 6 are not applicable to waits and generate an error.

mustOn is the bitmap of bits that must be True to satisfy the condition.

D15	••••	D10	D9	D8	D7	D6	D5	D4
XXX		XXX	XXX	mustOn 8	mustOn 7	mustOn 6	mustOn 5	mustOn

For D0 through D8:

- 1 = Bit must be True
- 0 = Don't care (default)

mustOff is the bitmap of bits that must be False to satisfy the condition:

D15	••••	D10	D9	D8	D7	D6	D5	D4
XXX	•••	XXX	XXX	mustOff 8	mustOff 7	mustOff 6	mustOff 5	mustOff

For D0 through D8:

- 1 = Bit must be False
- 0 = Don't care (default)

matchType selects the type of match required for the bitmap:

matchType Constant	matchType Value
NIMC_MATCH_ALL	0
NIMC_MATCH_ANY	1

NIMC_MATCH_ANY means that a match of any bit (logical OR) is sufficient to satisfy the condition while NIMC_MATCH_ALL requires a complete pattern match (logical AND) of all bits.

timeOut is the wait timeout value in 100 millisecond increments. The range is 0 to 65,535. If you specify a timeout of 65,535, it is treated as an infinite timeout and waits until the condition is met.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function controls the flow of execution in a stored program. It suspends program execution and waits until the specified condition is met or the timeout expires. When the condition is met, program execution is resumed with the next function after the Wait on Event.

If the timeout expires before the condition is met, an error is generated and the program goes into the paused state. Refer to the <u>Pause/Resume</u> <u>Program</u> function for information about resuming a paused program.

If you set a timeout of zero, the condition must already be true or an error is generated. You can wait on virtually any instantaneous status of axes, encoders, programs, motion I/O, or general-purpose I/O. Where applicable, you can select the specified resource with the resource parameter.

When waiting on a program with the

NIMC_CONDITION_PROGRAM_COMPLETE condition, **mustOn**, **mustOff**, and **matchType** parameters are not required and their values are ignored. You set resource equal to the specified program number to wait on. The balance of the conditions test status bitmaps and function similar to each other as described in the remainder of this section.

The **mustOn**, **mustOff**, and **matchType** parameters work together to define a bitmap of True and False bits that must be matched to satisfy the condition. Depending on the type of event, the **mustOn** and **mustOff** bitmaps are either one based or zero based. If the event is tied to an I/O port resource the bitmaps start at bit 0 to correspond with line 0 on an I/O port. If the event is tied to an axis-based resource, such as an encoder or an axis, the bitmaps start at bit 1 to correspond with axis 1. The **matchType** parameter allows you to select between an OR match, where any matching bit is sufficient, and an AND match, where all status bits must match the True/False bitmap defined by MustOn and MustOff.

Using the MustOn/MustOff protocol gives you tri-state control over each match bit: True, False or Don't care. A one (1) in a bit location of the MustOn bitmap sets the match bit to True, while a one (1) in the corresponding location of the MustOff bitmap resets the match bit to False. A zero (0) in either bitmap has no affect, so leaving both the MustOn and MustOff bits at zero defines the bit as Don't care. If you set both the MustOn and MustOff bits to one (1), it is interpreted as a

MustOn condition and the match bit is set to True.

The NIMC_CONDITION_LIMIT_INPUT_ACTIVE and NIMC_CONDITION_SOFTWARE_LIMIT_ACTIVE conditions create a combined status bitmap where if either the forward or reverse limit is active, the bit is True.

When the **returnVector** is set to anything other than zero (0), the condition code and status bitmap that satisfied the condition are returned to the destination specified, either to a variable or the host computer, as two 16-bit words (u16).

Waits are one of the most powerful and useful features on the NI motion controller. While a program is suspended waiting for a condition, NI-Motion is not wasting CPU cycles on it. The preemptive multitasking realtime operating system (RTOS) on the NI motion controller suspends the task until the condition is met or the timeout expires. This feature allows up to 10 programs to be running simultaneously with little impact on function execution performance.

To perform a conditional wait on two unrelated conditions, store the Wait on Condition function twice—the first with **waitType** = NIMC_WAIT_OR and the second with **waitType** = NIMC_WAIT.

Note Two sequential Wait on Condition functions both with waitType = NIMC_WAIT emulates a Wait AND, because both wait conditions must evaluate successfully before program execution is resumed. However, both wait conditions do not have to occur at the same time. The first wait could pass before the second wait is reached. When the second wait passes, the first wait may or may not be True.

Example

In program one, you want to wait until axes 1 through 3 have found home or until program two is complete. To accomplish this, store a Wait on Event function with the following parameters:

waitType = NIMC_WAIT_OR

condition = NIMC_CONDITION_HOME_FOUND (7) **mustOn** = 0x0E, which corresponds to the following bitmap:

D15	••••	D10	D9	D8	D7	D6	D5	D4

XXX	••••	XXX	XXX	mustOn 8	mustOn 7	mustOn 6	mustOn 5	mustOn
0		0	0	0	0	0	0	0

Note Because you are waiting on axes, the bitmap uses bits one through eight.

mustOff = 0x00, which corresponds to the following bitmap:

D15	•••	D10	D9	D8	D7	D6	D5	D4
XXX		XXX	XXX	mustOff 8	mustOff 7	mustOff 6	mustOff 5	mustOff
0	••••	0	0	0	0	0	0	0

Note Because you are waiting on axes, the bitmap uses bits one through eight.

matchType = NIMC_MATCH_ALL (0)
timeOut = 100 (timeout after 10 s)
returnVector = 0 (throw the status away)

Immediately follow this with a second Wait on Condition function with the following parameters:

```
resource = 2 (for program two)
waitType = NIMC_WAIT
condition = NIMC_CONDITION_PROGRAM_COMPLETE (20)
timeOut = 100 (timeout after 10 s)
returnVector = 0 (throw the status away)
```

In this example, the home found status of axis 4 is do not care.

Object Management

You can use Object Management functions to organize, annotate, and save program or buffer objects to flash ROM. These advanced functions are primarily used for applications that require non-volatile program storage. You can run programs out of RAM and create and use buffers in RAM without using any of these functions.

Object Memory Management

Load Memory Object Description

Read Memory Object Description

Read Object Registry

flex_load_description

Device Compatibility

Load Memory Object Description

Usage

status = flex_load_description(u8 boardID, u8 object, i8* description);

Purpose

Loads a ASCII text description for a program or buffer object.
Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
object	u8	program or other memory object
description	[i8]*	ASCII character array describing the object

Parameter Discussion

object is a program or buffer object stored in onboard RAM.

description is an ASCII character array of up to 32 characters that describes the object.

Using This Function

This function loads a text description for a program or buffer object. The ASCII text description is useful as a quick reminder of the contents or purpose of an object stored in memory.



Note This function must be executed while the object is still in RAM. After the object is saved to ROM, its description cannot be changed.

The description is limited to 32 characters; extra characters are ignored. You can retrieve the stored description with the <u>Read Memory Object</u> <u>Description</u> function.

Device Compatibility

Object Memory Management

Usage

status = flex_object_mem_manage(u8 boardID, u8 object, u16 operation);

Purpose

Saves, deletes, or frees programs or buffers in RAM and ROM.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
object	u8	program or other memory object
operation	u16	operation to perform on the object

Parameter Discussion

object is a program or buffer stored in onboard RAM or ROM.

operation is the operation to perform on the memory object.

operation Constant	operation Value
NIMC_OBJECT_SAVE	0
NIMC_OBJECT_DELETE	1
NIMC_OBJECT_FREE	2

Using This Function

N

This function is used to save to ROM, delete from ROM, or free from RAM, a program or buffer object. Objects saved to non-volatile Flash ROM are available for use at any future time, even after power cycles.

To save an object to ROM, call this function with operation = NIMC_OBJECT_SAVE. The object is copied to ROM and exists in both RAM and ROM until the next power cycle, when the RAM image is erased.

To remove an object from ROM, call this function with operation = NIMC_OBJECT_DELETE. The object is deleted from both ROM and RAM if it still exists in RAM.

After you have saved an object to ROM, you can free up its space in RAM by calling this function with operation = NIMC_OBJECT_FREE. This has no effect on the copy in ROM but deletes the image in RAM, making more memory available for storing additional programs or other objects.

Note You cannot save or delete an object while any program is running or any buffer is in use. Also, you cannot free a program or buffer while it is running. In addition, you cannot save or delete an object when any motor is moving. Attempting to execute this function in these cases generates an error.

You can delete a buffer that is active, such as a high-speed capture buffer that is waiting for a trigger, but NI-Motion returns an error in this case.

 \bigcirc Tip Saving or deleting an object takes 2 to 4 seconds.

The 7340/7330 controller has 64 KB of RAM plus 128 KB of ROM (divided into two 64 KB sectors) for program and buffer storage. You can run programs from either RAM or ROM, but you cannot split programs between the two, and you cannot split programs between the two 64 KB ROM sectors. With an average command size of 10 bytes, a single program can be as large as 6,400 commands. As another example, the 7340/7330 controller can simultaneously execute 10 programs, five from RAM and five from ROM, with each program up to 1,280 commands long.

Like programs, buffers can be stored in RAM or ROM. The maximum

buffer size is somewhat less than 64 KB (16,000 samples), because some of each sector is used for record keeping.

This function may take longer than 62 ms to process, so it is not guaranteed to be compatible with real-time execution.

flex_read_description_rtn

Device Compatibility

Read Memory Object Description

Usage

status = flex_read_description_rtn(u8 boardID, u8 object, i8* description);

Purpose

Reads the ASCII text description for a program or buffer object.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
object	u8	program or other memory object
description	[i8]*	ASCII character array describing the object

Parameter Discussion

object is a program or buffer object stored in onboard RAM or ROM. **description** is an ASCII character array of up to 32 characters that describes the object.

Using This Function

This function returns the ASCII text description for a program or buffer object. The ASCII text description, previously loaded with the <u>Load</u> <u>Memory Object Description</u> function, is useful as a quick reminder of the contents or purpose of an object stored in memory.

flex_read_registry_rtn

Device Compatibility

Read Object Registry

Usage

status = flex_read_registry_rtn(u8 boardID, u8 index, REGISTRY*
registryRecord);

Purpose

Reads a data record for a memory object from the Object Registry.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
index	u8	registry record number
registryRecord	REGISTRY*	data record containing information about the memory object

Parameter Discussion

index is the registry record number. The range for index is 0 to 31.

registryRecord is the data record containing object information in the following structure:

```
struct {
    u16 device; // Object number
    u16 type; // Object type. If 1, a program. If 2, a buffer.
    u32 pstart; // Start address in RAM or ROM
    u32 size; // Size of the object in words
} REGISTRY;
```

Object type tells you the type of object stored:

Object type	Value
NIMC_OBJECT_TYPE_PROGRAM	1
NIMC_OBJECT_TYPE_BUFFER	2

Size is in number of 16-bit words. For buffers, the size value returned is twice the size of the buffer as you configured it with the <u>Configure Buffer</u> function, where **bufferSize** is specified in 32-bit words.

If pstart < 0x100000, then the object is in Flash (ROM), otherwise, the object is in RAM.

Using This Function

This function returns a registry record for an object from the object registry. The object registry contains information about all objects stored in memory. You can store up to 32 objects in RAM and/or ROM. Each time an object is stored, a new record is created to keep track of it.



Note If an object exists both in RAM and ROM, this function reports the object as existing in RAM. To see the object in ROM, you must first free the object from RAM.

On 7330 and 7340 motion controllers, objects are in RAM if **pstart** \geq 0x100000, and in ROM if **pstart** < 0x100000. On the 7350 controller, objects are in RAM if **pstart** \geq 0x200000, and in ROM if **pstart** < 0x200000.

Registry records are referenced by index and each call to this function returns information about the referenced object. The index is not the same as the object number. You can use up to 255 unique object numbers (0x01 through 0xFF) but only 32 objects can be stored in memory at one time.

Data Operations

Data Operation functions include the available math functions on generalpurpose variables. Variables can be loaded, added, multiplied, ANDed, and so on before being used as data in a motion control function.

General-purpose variables are 32 bits long and can be used either signed (i32) or unsigned (u32). All Data Operation functions operate on 32-bit values and return 32-bit values. You must be careful to avoid overflow and underflow conditions. For example, multiplying two 32-bit variables and returning the result to a 32-bit variable might overflow and wrap around.

Smaller sized data is right aligned within a 32-bit variable. Bitwise logical functions always assume this alignment and return similarly aligned results.

Many NI-Motion functions can take input data from a general-purpose variable by pointing to the variable with the input vector parameter. Similarly, all read functions can return data to a general-purpose variable by using the return vector parameter, refer to <u>Input and Return Vectors</u>.

All data operation functions set condition codes (less than, equal to or greater than zero) depending on the result of the operation. Your program can test these conditions with the <u>Jump on Event</u> function. Executing a data operations functions with a return vector of zero (0) tells the program to set the condition code and then throw the resulting data away. In this way, you can use all the data operations functions as tests for conditional branching.

You can use indirect variables as variable inputs or return vectors with all data operation functions.

Add Variables Subtract Variables Multiply Variables Divide Variables AND Variables OR Variables Exclusive OR Variables Invert Variable Logical Shift Variable Load Constant to Variable Read Variable flex_add_vars

Device Compatibility

Add Variables

Usage

status = flex_add_vars(u8 boardID, u8 variable1, u8 variable2, u8 returnVector);

Purpose

Adds the values in the two variables and returns the result.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
variable1	u8	first operand
variable2	u8	second operand
returnVector	u8	destination for the result

Parameter Discussion

variable1 is the variable holding the first operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

variable2 is the variable holding the second operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function adds the values in the two variables and returns the result to the destination specified by the **returnVector**.

V1 + V2 = Result

where: V1 is the value at **variable1**. V2 is the value at **variable2**. *Result* is stored in the location pointed to by **returnVector**.

The result can be returned to a new variable or one of the two input variables, returned to the host computer, or thrown away. In all cases the condition codes are set according to the resulting value. This value is compared with 0, and the appropriate condition code is set: GREATER THAN, LESS THAN, or EQUAL to zero.



Note Be careful when adding two large values. The result could overflow and wrap around. No error generates when an overflow occurs.

flex_and_vars

Device Compatibility

AND Variables

Usage

status = flex_and_vars(u8 boardID, u8 variable1, u8 variable2, u8 returnVector);

Purpose

Performs a bitwise AND of the values in the two variables and returns the result.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
variable1	u8	first operand
variable2	u8	second operand
returnVector	u8	destination for the result

Parameter Discussion

variable1 is the variable holding the first operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

variable2 is the variable holding the second operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function performs a bitwise logical AND of the values in the two variables and returns the result to the destination specified by the **returnVector**.

V1 AND V2 = Result

where: V1 is the value at **variable1**. V2 is the value at **variable2**. *Result* is stored in the location pointed to by **returnVector**.

The result can be returned to a new variable or one of the two input variables, returned to the host computer, or thrown away. In all cases the EQUAL condition code is set True if the result equals zero (all bits low) and False if any bit is set. The GREATER THAN and LESS THAN codes are also set but can be confusing after logical bitwise operations.

Example

If the values in variable1 and variable2 are 0x0000 1234 and 0x0000 EEEE, respectively, the result of the bitwise AND is 0x0000 0224 which is NOT EQUAL to zero.
flex_div_vars

Device Compatibility

Divide Variables

Usage

status = flex_div_vars(u8 boardID, u8 variable1, u8 variable2, u8 returnVector);

Purpose

Divides the value in the first variable by the value in the second variable and returns the result.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
variable1	u8	dividend
variable2	u8	divisor
returnVector	u8	destination for the result

Parameter Discussion

variable1 is the variable holding the first operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

variable2 is the variable holding the second operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function divides the value in the first variable by the value in the second variable and returns the result to the destination specified by the **returnVector**.

V1 / V2 = Result

where: V1 is the value at **variable1**. V2 is the value at **variable2**. Result is stored in the location pointed to by **returnVector**.

The result can be returned to a new variable or one of the two input variables, returned to the host computer, or thrown away. In all cases the condition codes are set according to the resulting value. This value is compared with 0, and the appropriate condition code is set: GREATER THAN, LESS THAN, or EQUAL to zero.



Note This function does an integer divide and the remainder is lost.

Note If V2 = 0, this function returns an invalid function data modal error (-70078).

flex_load_var

Device Compatibility

Load Constant to Variable

Usage

status = flex_load_var(u8 boardID, i32 value, u8 variable1);

Purpose

Loads a constant value into a variable.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
value	i32	value to load into the variable
variable1	u8	variable to load

Parameter Discussion

value is the value to load into the variable.

variable1 is the variable to load. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

Using This Function

This function loads a constant value into the selected variable.

The condition codes are set according to the loaded value, GREATER THAN, LESS THAN, or EQUAL to zero.

flex_lshift_var

Device Compatibility

Logical Shift Variable

Usage

status = flex_lshift_var(u8 boardID, u8 variable1, i8 logicalShift, u8
returnVector);

Purpose

Performs a logical shift on the value in a variable and returns the result.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
variable1	u8	variable holding the value to shift
logicalShift	i8	number of bits to shift
returnVector	u8	destination for the result

Parameter Discussion

variable1 is the variable holding the value to shift. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

logicalShift is the number of bits to shift. A positive **logicalShift** value shifts **variable1** to the left and a negative value shifts **variable1** to the right. The shift range is -31 through +31 bits.

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function performs a logical shift on the value in the selected variable and returns the result to the destination specified by the **returnVector**.

For positive logicalShift values:

V1 « Logical Shift = Result

For negative logicalShift values:

V1 » Logical Shift = Result

where:

V1 is the value at **variable1**.

Result is stored in the location pointed to by **returnVector**.

The result can be returned to a new variable or one of the two input variables, returned to the host computer, or thrown away. In all cases the condition codes are set according to the resulting value. This value is compared with 0, and the appropriate condition code is set: GREATER THAN, LESS THAN, or EQUAL to zero.

This function actually performs an arithmetic rather than logical shift if the variable is a signed 32-bit value (i32). Negative values are sign-extended when shifted to the right. You can use this function to perform division or scaling of signed or unsigned numbers. In this case the function effectively performs the following:

V1 × 2^(Logical Shift) = Result

Example 1

If the value in **variable1** is 0x0000 F002 and **logicalShift** = -1, this function returns 0x00007801.

Example 2

If the value in **variable1** is 0xFFFF F002 and **logicalShift** = -1, this function returns 0xFFFFF801. The sign of the value is preserved by sign-extension.

flex_mult_vars

Device Compatibility

Multiply Variables

Usage

status = flex_mult_vars(u8 boardID, u8 variable1, u8 variable2, u8
returnVector);

Purpose

Multiplies the values in the two variables and returns the result.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
variable1	u8	first operand
variable2	u8	second operand
returnVector	u8	destination for the result

Parameter Discussion

variable1 is the variable holding the first operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

variable2 is the variable holding the second operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function multiplies the values in the two variables and returns the result to the destination specified by the **returnVector**.

V1 × V2 = Result

where: V1 is the value at **variable1**. V2 is the value at **variable2**. *Result* is stored in the location pointed to by **returnVector**.

The result can be returned to a new variable or one of the two input variables, returned to the host computer, or thrown away. In all cases the condition codes are set according to the resulting value. This value is compared with 0, and the appropriate condition code is set: GREATER THAN, LESS THAN, or EQUAL to zero.



Note Be careful when multiplying two large values. The result can overflow and wrap around. An error is not generated when an overflow occurs.

flex_not_var

Device Compatibility

Invert Variable

Usage

status = flex_not_var(u8 boardID, u8 variable1, u8 returnVector);

Purpose

Performs a bitwise inversion (NOT) on the value in a variable and returns the result.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
variable1	u8	variable to invert
returnVector	u8	destination for the result

Parameter Discussion

variable1 is the location of the variable to invert. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function performs a bitwise logical NOT on the value in the selected variable and returns the result to the destination specified by the **returnVector**.

~(V1) = Result

where: V1 is the value at **variable1**. *Result* is stored in the location pointed to by **returnVector**.

The result can be returned to a new variable or to the input variable, returned to the host computer, or thrown away. In all cases the EQUAL condition code is set True if the result equals zero (all bits low) and False if any bit is set. The GREATER THAN and LESS THAN codes are also set but can be confusing after logical bitwise operations.

Example

If the value in variable1 is 0x0000 5A5A, the result of the bitwise NOT is 0xFFFF A5A5. The EQUAL condition code is set to False.

flex_or_vars

Device Compatibility

OR Variables

Usage

status = flex_or_vars(u8 boardID, u8 variable1, u8 variable2, u8 returnVector);

Purpose

Performs a bitwise OR of the values in the two variables and returns the result.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
variable1	u8	first operand
variable2	u8	second operand
returnVector	u8	destination for the result

Parameter Discussion

variable1 is the variable holding the first operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

variable2 is the variable holding the second operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function performs a bitwise logical OR of the values in the two variables and returns the result to the destination specified by the **returnVector**.

V1 OR V2 = Result

where: V1 is the value at **variable1**. V2 is the value at **variable2**. *Result* is stored in the location pointed to by **returnVector**.

The result can be returned to a new variable or one of the two input variables, returned to the host computer, or thrown away. In all cases the EQUAL condition code is set True if the result equals zero (all bits low) and False if any bit is set. The GREATER THAN and LESS THAN codes are also set but can be confusing after logical bitwise operations.

Example

If the values in **variable1** and **variable2** are 0x5A5A 1234 and 0x8282 0000, respectively, the result of the bitwise OR is 0xDADA 1234, which is NOT EQUAL to zero.
flex_read_var

Device Compatibility

Read Variable

Usage

status = flex_read_var(u8 boardID, u8 variable1, u8 returnVector);

Purpose

Reads the value of a variable and returns the result.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
variable1	u8	variable to read
returnVector	u8	destination for the return data

Parameter Discussion

variable1 is the variable to read. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.



Note The suffix _rtn on the function indicates that the data must be returned to the host. When this calling convention is used, no **returnVector** is required.

Using This Function

This function reads the value of the selected variable and returns it to the destination specified by the **returnVector**.

When the value of **returnVector** is set to **Return to Host (0xFF)**, this function returns the value to the return data buffer.

The condition codes are set according to the value read: GREATER THAN, LESS THAN, or EQUAL to zero.

flex_read_var_rtn

Device Compatibility

Read Variable Return

Usage

status = flex_read_var_rtn(u8 boardID, u8 variable1, i32* value);

Purpose

Reads the value of a variable and returns the result.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
variable1	u8	variable to read
value	i32*	value of the variable

Parameter Discussion

variable1 is the variable to read. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

value is the value of the variable.

Using This Function

This function reads the value of the selected variable and returns it to **value**.

The condition codes are set according to the value read: GREATER THAN, LESS THAN, or EQUAL to zero.

flex_sub_vars

Device Compatibility

Subtract Variables

Usage

status = flex_sub_vars(u8 boardID, u8 variable1, u8 variable2, u8 returnVector);

Purpose

Subtracts the value of second variable from the value of the first variable and returns the result.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
variable1	u8	first operand
variable2	u8	second operand
returnVector	u8	destination for the result

Parameter Discussion

variable1 is the variable holding the first operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

variable2 is the variable holding the second operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function subtracts the value of second variable from the value of the first variable and returns the result to the destination specified by the **returnVector**.

V1 – V2 = Result

where: V1 is the value at **variable1**. V2 is the value at **variable2**. *Result* is stored in the location pointed to by **returnVector**.

The result can be returned to a new variable or one of the two input variables, returned to the host computer, or thrown away. In all cases the condition codes are set according to the resulting value. This value is compared with 0, and the appropriate condition code is set: GREATER THAN, LESS THAN, or EQUAL to zero.

This function is often used to compare two values prior to executing a conditional jump with the <u>Jump on Event</u> function. In this case, the result is typically thrown away by setting **returnVector** = 0.

flex_xor_vars

Device Compatibility

Exclusive OR Variables

Usage

status = flex_xor_vars(u8 boardID, u8 variable1, u8 variable2, u8 returnVector);

Purpose

Performs a bitwise Exclusive OR (XOR) of the values in the two variables and returns the result.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
variable1	u8	first operand
variable2	u8	second operand
returnVector	u8	destination for the result

Parameter Discussion

variable1 is the variable holding the first operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

variable2 is the variable holding the second operand. Valid inputs are variables (0x01 through 0x78) and indirect variables (0x81 through 0xF8).

returnVector indicates the destination for the return data generated by this function. Available return vectors include return data to the host (0xFF), to a variable (0x01 through 0x78), to an indirect variable (0x81 through 0xF8), and return no data (0). Refer to <u>Input and Return Vectors</u> for more detailed information.

Using This Function

This function performs a bitwise logical XOR of the values in the two variables and returns the result to the destination specified by the **returnVector**.

V1 XOR V2 = Result

where:
V1 is the value at variable1.
V2 is the value at variable2.
Result is stored in the location pointed to by returnVector.

The result can be returned to a new variable or one of the two input variables, returned to the host computer, or thrown away. In all cases the EQUAL condition code is set True if the result equals zero (all bits low) and False if any bit is set. The GREATER THAN and LESS THAN codes are also set but can be confusing after logical bitwise operations.

Advanced

You can use the advanced functions to control the communications between the host computer and NI motion controller. These functions allow you to check the status of communications, clear the RDB, and manage the low-level communications to the controller. You typically do not have to use any of these functions because the default configuration is correct for almost all applications. These functions are available to handle special applications.

The advanced functions include specialized <u>read</u> and <u>write</u> functions for quickly viewing axis or coordinate data or status information, configuring I/O lines, and so on.

The advanced functions also include advanced <u>methods</u> and two utility functions that are regularly used but are different from the rest of the NI-Motion API in that they are not typically included in application code: Clear Power Up Status and Save Default Parameters.

Read Communication Status

Flush Return Data Buffer

Clear Power Up Status

Save Default Parameters

Reset Default Parameters

Enable Auto Start

Enable Shutdown

Enable 1394 Watchdog

Read Board Temperature

flex_load_axis_configuration_parameter Decompatible

Load Axis Configuration Parameter

Usage

status = flex_load_axis_configuration_parameter(i32 boardID, i32 axis, TnimcAxisConfigurationParameter attribute, NIMC_DATA* value);

Purpose

Loads data on a per axis basis for an axis configuration object on the motion controller.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	i32	assigned by Measurement & Automation Explorer (MAX)
axis	i32	axis to configure
attribute	TnimcAxisConfigurationParameter	attribute to set
value	NIMC_DATA	the value for the attribute you are loading

Parameter Discussion

axis is the axis to configure with this function. Valid values are NIMC_AXIS1 through NIMC_AXIS30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

attribute is the attribute to load. The following are valid attributes:

• **TnimcAxisConfigurationParameterEnable**—Set **boolData** in the NIMC_DATA structure to NIMC_TRUE to enable the axis, and NIMC_FALSE to disable the axis.

value is the value for the attribute you are loading in the following structure:

struct{

i32 longData; u8 boolData; f64 doubleData;

```
} NIMC_DATA;
```

Based on the attribute the correct member of $\ensuremath{\operatorname{NIMC_DATA}}$ needs to be set.

Using This Function

Use this function to load the data on a per axis basis for the Axis Configuration object on the motion controller.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx controller:

• Because the number of axes enabled depends on the control loop rate, use the Enable Axis function to enable/disable an axis for 73xx controllers.

flex_clear_pu_status

Device Compatibility

Clear Power Up Status

Usage

status = flex_clear_pu_status(u8 boardID);

Purpose

Clears the Power-Up status bit and boots up the controller, making it ready to accept functions.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
Using This Function

When the NI motion controller is reset by a power cycle, watchdog timeout, or other means, the controller is suspended in a Power-Up state and a Power-Up status bit in the Communications Status Register (CSR) is set. Use this function to clear the Power-Up status bit and ready the controller for motion control communications.

You cannot execute most of the other motion control functions until the Power-Up status bit has been cleared by using this function. This lockout ensures that you are aware of the occurrence of an unexpected reset, as in the case of a watchdog timeout.

You can include this function one time at the beginning of an initialization routine. Do not include it in other routines to avoid the possibility of restarting an application unexpectedly after a power cycle or watchdog timeout. The <u>Initialize Controller</u> function automatically calls this function.

Note If you are running a NI-Motion startup application on ETS (PharLap) systems, the first function in your application must be either the Clear Power Up Status or Initialize Controller function. Calling any other function first may result in error –70012 (NIMC_badBoardIDError) because the driver may not have loaded before a NI-Motion function is called.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

When the 73*xx* motion controller is in the Power-Up state, the Move Complete Status (MCS) register contains a power-up code that describes why the controller is in the Power-Up state. To access this code, execute the <u>Read Move Complete Status</u> function.

Code	Reset Type	Cause
0x80	Bus reset	Normal PC power cycle
0x40	Power-Up reset	Normal PC power cycle
0x20	Watchdog timeout	Fatal internal error
0x08	Shutdown	Shutdown (E-Stop) input active; refer to the <u>Enable Shutdown</u> function
0x02	Software reset	Firmware download

The following table describes the power-up codes:

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

The power-up codes do not apply to the NI SoftMotion Controller.

flex_enable_auto_start

Device Compatibility

Enable Auto Start

Usage

status = flex_enable_auto_start(u8 boardID, u8 enableOrDisable, u8
programToExecute);

Purpose

Allows you to automatically run a program when the controller powers up.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
enableOrDisable	u8	enable or disable auto start
programToExecute	u8	program number to execute

Parameter Discussion

enableOrDisable enables or disables the auto start feature. Set this to NIMC_TRUE (1) to enable auto start and NIMC_FALSE (0) to disable auto start.

programToExecute is the onboard program the controller executed if the auto start feature is enabled. This must be a valid program number (1—255), that is stored to FLASH using the <u>Object Memory Management</u> function.

Using This Function

This function configures the controller to automatically start an onboard program on power up. After auto start is enabled, the controller automatically executes the onboard program specified when the controller is powered on. The onboard program to be executed must be saved to FLASH using the <u>Object Memory Management</u> function before the controller is powered down. If the controller does not find a valid program that it can load, NIMC_autoStartFailedError is generated. If the onboard program is removed from FLASH memory, the auto start functionality is disabled.



Note This function writes to onboard FLASH memory and hence it is not safe to execute when motors are in motion. Doing so generates a NIMC_wrongModeError.

This function may take longer than 62 ms to process, so it is not guaranteed to be compatible with real-time execution.

flex_enable_shutdown

Device Compatibility

Enable Shutdown

Usage

status = flex_enable_shutdown(u8 boardID);

Purpose

Enables the emergency shutdown functionality of the controller.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)

Using This Function

This function enables the controller to react to the shutdown (E-Stop) input. When the shutdown input transitions from low to high, the controller goes into a shutdown state. The following actions take place in the shutdown state:

- All the axes are killed. On servo axes, the control loop is disabled and the output DACs are zeroed, allowing frictional forces alone to stop the motion. On stepper axes, the stepper pulse generation is stopped. On both axis types, there is no trajectory profile. If enabled, the inhibit output is activated to inhibit (disable) the servo amplifier or stepper driver. You can enable the inhibit outputs and set their polarity as active high or active low with the <u>Configure</u> <u>Inhibit Outputs</u> function.
- All the axes, encoders and ADCs are disabled.
- All the digital I/O is re-initialized to defaults. If the user has saved defaults using the <u>Save Default Parameters</u> function, the digital IO is re-initialized to the user defaults, else it is re-initialized to the factory defaults.
- All currently executing onboard programs are stopped.
- The controller does not accept any functions, except for the following ones:
 - Get Motion Board Information
 - Read Error Message
 - Enable Auto Start
 - <u>Read Variable</u>

The shutdown functionality is disabled by default on power up. This functionality has to be enabled every time the controller is powered up. You must enable this feature only after the shutdown (E-Stop) circuit is properly configured and connected to the controller. After shutdown (E-Stop) is enabled, it can be disabled only by resetting or power cycling the controller.



Note After the controller has shut down, you have to reset it or power cycle it before it can be used again.



Note The active state of the shutdown (E-Stop) line is low. To prevent the controller from shutting down prematurely, the line

must be high (inactive) when you enable shutdown (E-Stop).

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- All read functions are available when the NI SoftMotion Controller is in a shutdown state.
- All write functions, including <u>Start Motion</u> and <u>Stop Motion</u>, are *unavailable* when the NI SoftMotion Controller is in a shutdown state.

flex_flush_rdb

Device Compatibility

Flush Return Data Buffer

Usage

status = flex_flush_rdb(u8 boardID);

Purpose

Clears the Return Data Buffer by deleting all of the buffered data.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)

Using This Function

This function clears the Return Data Buffer by repeatedly reading the RDB until the buffer is empty. All return data packets in the RDB are deleted and nothing is returned by this function.

You typically use the Flush Return Data Buffer function after an error condition when the data in the Return Data Buffer is no longer valid or relevant. This function is also useful for flushing the RDB after a programming error has caused the buffer to become skewed. Buffer skew is when the data returned by a read function using the _rtn calling convention (such as <u>Read Position</u>) does not return the expected data but rather returns data requested by a previous function.

Refer to the motion controller documentation for more information about low-level communications protocols and return data packets.

Read Board Temperature

Usage

status = flex_read_board_temperature (u8 boardID, f64 *temperature);

Purpose

Reads the temperature from the motion controller.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
temperature	f64 *	device temperature °C

Parameter Discussion

temperature is the temperature read from the temperature sensor on the motion controller in °C.

Using This Function

The Read Board Temperature function reads the temperature in °C from the temperature sensor on the motion controller. You can use the current temperature sensor reading to determine if the device requires recalibration.



Note Only 7350 controllers support the Read Board Temperature function.

flex_read_csr_rtn

Device Compatibility

Read Communication Status

Usage

status = flex_read_csr_rtn(u8 boardID, u16* csr);

Purpose

Reads the Communication Status Register (CSR).



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
csr	u16*	bitmap of communications status

Parameter Discussion

csr is the bitmap of communication status from the Communication Status Register.

For D0 Ready to Receive (RTR):

- 1 = Ready to receive a word from the host
- 0 = Not ready to receive (busy)

For D1 Ready to Send (RTS):

1 = Ready to send a word from the RDB to the host

0 = Not ready to send (RDB empty)

For D3 Command In Process (PIP):

1 = Waiting for more data from host to complete the command

0 = Idle

For D4 Packet Error (Pkt Err):

1 = Communication packet error

0 = No error

For D5 Power-Up Reset (PU Reset):

- 1 = Controller is in the Power-Up state
- 0 = Power-Up state has been cleared

For D6 Error Message (Err Msg):

1 = Modal error message pending

0 = No error

For D7 Hardware Failure (HW Fail):

- 1 = Fatal hardware error occurred
- 0 = No error

Using This Function

On power up, the controller is in a power-up reset state. Refer to <u>Clear</u> <u>Power Up Status</u> for information about clearing the reset condition.



Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- This function performs a direct read of the Communication Status Register (CSR) on the 73xx motion controllers. The CSR is a hardware register containing communication handshaking and error status bits. The NI-Motion software polls this register continuously when sending and receiving packets for handshaking and error checking purposes. Refer to the motion controller documentation for more information about low-level communication protocols and return data packets.
- You also can call this function at any time to check the communication and error status. Because the CSR is always up to date and directly accessible over the computer bus, executing this function does not affect the operation of the NI motion controller.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- If you are using the NI SoftMotion Controller, this function reads the NI SoftMotion Controller engine service status.
 - D0 Ready to Receive (RTR) indicates if the NI SoftMotion Controller is running.
 - D4 Shutdown indicates if the NI SoftMotion Controller is in a shutdown state.
- The NI SoftMotion Controller does not support the following CSR bits:
 - D1 Ready to Send (RTS)
 - D3 Command In Process (PIP)
 - D7 Hardware Failure (HW Fail)

flex_reset_defaults

Device Compatibility

Reset Default Parameters

Usage

status = flex_reset_defaults(u8 boardID)

Purpose

Resets the power-up defaults to the factory-default settings.


Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)

Using This Function

This function resets the power-up defaults to the factory-default settings for all important configuration, initialization, and trajectory parameters for use after subsequent power-up resets. When you execute this function, the default values for all parameters listed in Default Parameters are saved to nonvolatile flash memory and become the power-up defaults.



Note The effect of this function is not realized until the next time the controller is powered up from a power-down state.

You only need to use this function if you have previously modified the power-up defaults using the <u>Save Default Parameters</u> function and want to revert back to the factory defaults.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

• This function may take longer than 62 ms to process on 73*xx* controllers. This function is not guaranteed to execute in real time as it performs memory allocation.

flex_read_rdb

Device Compatibility

Read Return Data Buffer

Usage

status = flex_read_rdb(u8 boardID, u16 *number, u16 *wordCount, u8
*resource, u16 *command, u16 *commandData);

Purpose

Reads the Return Data Buffer.

Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)
number	u16*	maximum number of data packets to read
wordCount	u16*	array containing the number of 16-bit words in the command packet
resource	u16*	array containing the device number returned by the controller
command	u16*	array containing the command ID returned by the controller
commandData	u16*	array containing the data read from the return data buffer

Parameter Discussion

number as an input contains the maximum number of data packets to read. **number** as an output returns the number of data packets read.

wordCount is an array containing the number of 16-bit words in the command packet.

resource is an array containing the resource ID or IDs returned by the controller.

command is an array containing the command ID or IDs returned by the controller.

commandData is an array containing the data read from the return data buffer.

Using This Function

You can use the Read Return Data Buffer function to read the RDB.

Refer to the motion controller documentation for more information about low-level communications protocols and return data packets.

flex_save_defaults

Device Compatibility

Save Default Parameters

Usage

status = flex_save_defaults(u8 boardID)

Purpose

Saves the current operating parameters as defaults.



Parameters

Name	Туре	Description
boardID	u8	assigned by Measurement & Automation Explorer (MAX)

Using This Function

This function saves all important configuration, initialization, and trajectory parameters for use after subsequent power-up resets. When you execute this function, all parameters are saved to nonvolatile flash memory and become the power-up defaults.

If necessary, you can reinstate the factory-default parameters as the power-up defaults with the <u>Reset Default Parameters</u> function.

This function does not perform a complete state save. For proper and safe operation after power-up, certain parameters are always reset to their factory defaults to bring the controller back to a known safe state. Parameters not stored are left out by design and are typically reset to zero at power-up.



Note If you want to remember a parameter that is not included in this list, you can copy that parameter to a general-purpose variable and it is saved with this function. You can then reset the parameter to the saved value with a program designed for this purpose.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- This function may take longer than 62 ms to process on 73*xx* controllers. This function is not guaranteed to execute in real time as it performs memory allocation.
- When the controller is powered up and the onboard processors boot, the defaults are automatically applied. There is some time, however, between the controller powering up and the application of defaults.

The defaults on the step, direction, breakpoint, and analog output signals do not take effect until the power up reset state is cleared. Use the <u>Clear Power Up Status</u> function to clear the power up reset state.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

• User defaults are not applied until the power up reset state is cleared. Use the <u>Clear Power Up Status</u> function to clear the power up reset state. Refer to the documentation for your drive for the initial power on state of the drive.

Advanced Read

You can use the advanced read functions to view axis or coordinate execution data and status information, I/O status information, encoder execution data, and so on.



Note To use the advanced read functions, you must include nimotion.h and link to nimotion.lib.

Read All Axis Data Read All Axis Status Read Axis Data Read Axis Status Read Capture Compare Data Read Coordinate Data Read Coordinate Status Read Coordinate Position Read Digital I/O Data Read Encoder Data

nimcReadAllAxisData

Device Compatibility

Read All Axis Data

Usage

status = nimcReadAllAxisData(TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, TnimcAxisData* data);

Purpose

Reads the position, velocity, following error, and encoder position of the selected axis.

Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to read
data	TnimcAllAxisData*	the data record containing axis execution information

Parameter Discussion

axisHandle is the axis to read with this function. Valid values are 1 through 30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError).

data is the data record containing axis execution information in the following structure:

struct {
 u32 size;
 f64 position;
 f64 velocity;
 f64 followingError;
 f64 encoderPosition;
} TnimcAllAxisData;

Using This Function

Use this function to read the position, velocity, following error, and encoder position of the selected axis.



Caution If this function uses reflective memory, it should *not* be used for event checking. In this case, only use the information returned by this function for display purposes. Refer to Reflective Memory Functions for more information.

For open-loop stepper axes, this function returns position information in number of steps generated. For closed-loop stepper axes, it converts the primary feedback position from counts to steps and then returns the value in steps. Closed-loop stepper axes require you to correctly load values of steps per revolution and counts per revolution to function correctly.



Note For closed-loop axes, this function always returns the position of the primary feedback resource. Refer to the Read Coordinate Position function for an easy way to read the positions of all axes in a coordinate in one call.

For axis velocity, this function returns filtered velocity in counts/s (for servo axes) or steps/s (for stepper axes). The sign of velocity indicates direction of motion.

For axis following error, this function returns the instantaneous difference between the commanded trajectory position and the actual feedback position in counts for servo systems and steps for stepper systems.

Encoder position data is in quadrature count value of the encoder mapped to the selected axis.

During axis setup, you can operate the closed-loop stepper axis in openloop mode and use this function to directly measure the counts per revolution and steps per revolution for the axis. These values must be loaded in advance for subsequent closed-loop operation. Refer to the Load Counts/Steps per Revolution function for more information.

You also can use the **encoderPosition** element to return a finer reading of position in cases where the encoder resolution greatly exceeds the step resolution of the closed-loop stepper axis.



Note Because of the way the controller updates information, the values returned are not guaranteed to all be sampled in the same time slice.

Example

To call this function on axis 2, use the following syntax:

```
TnimcAllAxisData data;
data.size = sizeof(TnimcAllAxisData);
nimcReadAllAxisData(boardID, 2, &data);
```



Note You must specify the value for the **size** element of **data** before calling this function. If you do not properly set the size element, the function will return error 70023 (NIMC_parameterValueError).

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

nimcReadAllAxisStatus

Device Compatibility

Read All Axis Status

Usage

status = nimcReadAllAxisStatus(TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, TnimcAxisStatus* data);

Purpose

Reads the execution status of the selected axis.

Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to read
data	TnimcAllAxisStatus*	the data record containing axis execution information

Parameter Discussion

axisHandle is the axis to read with this function. Valid values are 1 through 30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError).

data is the data record containing axis execution information in the following structure:

struct {

u32 size; u8 axisActive; u8 moveComplete; u8 profileComplete; u8 blendComplete; u8 followingErrorExceeded; u8 velocityThresholdExceeded; u8 moving; u8 directionForward; u8 indexCaptureOccurred; u8 positionCaptureOccurred; u8 positionCompareOccurred; u8 forwardLimitActive: u8 reverseLimitActive; u8 forwardSoftwareLimitActive: u8 reverseSoftwareLimitActive; u8 homeInputActive; u8 homeFound: u8 indexFound: u8 referenceFound: } TnimcAllAxisStatus;

Using This Function

Use this function to read the execution status of the selected axis.

 \wedge

Caution If this function uses reflective memory, it should *not* be used for event checking. In this case, only use the information returned by this function for display purposes. Refer to <u>Reflective</u> <u>Memory Functions</u> for more information.

 $\overline{\mathbb{N}}$

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

The **TnimcAllAxisStatus** structure will return TnimcTrue or TnimcFalse based on the following criteria:

Element	Status
axisActive	TnimcTrue = motor is on TnimcFalse = motor is off
moveComplete	TnimcTrue = move complete TnimcFalse = move not complete
profileComplete	TnimcTrue = profile is complete TnimcFalse = profile generation in process
blendComplete	TnimcTrue = blend complete TnimcFalse = blend pending
followingErrorExceeded	 TnimcTrue = axis exceeded the programmed following error limit TnimcFalse = axis following error is below the programmed following error limit Note If you are running a contoured move or slave axis move with a stepper axis, it is possible that the contour velocity or the slave axis geared velocity exceeds the maximum step rate of the controller. In this case, the controller kills the axis and sets the following error status to true. Refer to the <i>Specifications</i> section of your controller user manual for the

	maximum step rate of your controller.
velocityThresholdExceeded	TnimcTrue = axis velocity is above the programmed velocity threshold TnimcFalse = axis velocity is below the programmed velocity threshold
moving	TnimcTrue = axis is moving TnimcFalse = axis is stopped
directionForward	TnimcTrue = axis moving forward TnimcFalse = axis moving in reverse
indexCaptureOccurred	TnimcTrue = encoder index position capture occurred TnimcFalse = encoder index position not captured
positionCaptureOccurred	TnimcTrue = position capture (trigger) occurred TnimcFalse = position capture (trigger) pending or disabled
positionCompareOccurred	TnimcTrue = position compare (breakpoint) occurred TnimcFalse = position compare (breakpoint) pending or disabled
forwardLimitActive	TnimcTrue = forward limit active TnimcFalse = forward limit inactive
reverseLimitActive	TnimcTrue = reverse limit active TnimcFalse = reverse limit inactive
forwardSoftwareLimitActive	TnimcTrue = forward software limit active TnimcFalse = forward software limit inactive
reverseSoftwareLimitActive	TnimcTrue = reverse software limit active TnimcFalse = reverse software limit inactive
homeInputActive	TnimcTrue = home input active TnimcFalse = home input inactive
homeFound	TnimcTrue = home position found

	TnimcFalse = home position not found
indexFound	TnimcTrue = index position found TnimcFalse = index position not found
referenceFound	TnimcTrue = reference position found TnimcFalse = reference position not found

Example

To call this function on axis 2, use the following syntax:

```
TnimcAllAxisStatus data;
data.size = sizeof(TnimcAllAxisStatus);
nimcReadAllAxisStatus(boardID, 2, &data);
```



nimcReadAxisData

Device Compatibility

Read Axis Data

Usage

status = nimcReadAxisData(TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, TnimcAxisData attribute, TnimcData* data);

Purpose

Reads position, velocity, or following error information on an axis.
Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to read
attribute	TnimcAxisData	attribute to read
data	TnimcData*	the value for the attribute you are reading

Parameter Discussion

axisHandle is the axis to read data from. Valid values are 1 through 30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError).

attribute is the attribute to read. The following are valid attributes:

- TnimcAxisDataPosition
- TnimcAxisDataVelocity
- TnimcAxisDataFollowingError

data is the value for the attribute you are reading in the following structure:

struct{

i32 longData; u8 boolData; f64 doubleData;

```
} TnimcData;
```

Based on the attribute, the correct member of TnimcData will be returned by the motion controller. The values of the other elements in the structure are undefined.

Read **doubleData** in the TnimcData structure to retrieve the axis position, velocity, or following error.

Using This Function

Use this function to read position, velocity, or following error information on an axis.



Caution If this function uses reflective memory, it should *not* be used for event checking. In this case, only use the information returned by this function for display purposes. Refer to <u>Reflective</u> <u>Memory Functions</u> for more information.

For open-loop stepper axes, this function returns position information in number of steps generated. For closed-loop stepper axes, it converts the primary feedback position from counts to steps and then returns the value in steps. Closed-loop stepper axes require you to correctly load values of steps per revolution and counts per revolution to function correctly.



Note For closed-loop axes, this function always returns the position of the primary feedback resource. Refer to the <u>Read</u> <u>Coordinate Position</u> function for an easy way to read the positions of all axes in a coordinate in one call.

For velocity, this function returns filtered velocity in counts/s (for servo axes) or steps/s (for stepper axes). The sign of velocity indicates direction of motion.

For following error, this function returns the instantaneous difference between the commanded trajectory position and the actual feedback position in counts for servo systems and steps for stepper systems.



Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

nimcReadAxisStatus

Device Compatibility

Read Axis Status

Usage

status = nimcReadAxisStatus(TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, TnimcAxisStatus attribute, TnimcData* data);

Purpose

Reads the specified execution status information of an axis.

Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to read
attribute	TnimcAxisStatus	attribute to read
data	TnimcData*	the value for the attribute you are reading

Parameter Discussion

axisHandle is the axis to read with this function. Valid values are 1 through 30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError)

70006 (NIMC_badResourceIDOrAxisError).

attribute is the attribute to read. The following are valid attributes:

- TnimcAxisStatusAxisActive
- TnimcAxisStatusMoveComplete
- TnimcAxisStatusProfileComplete
- TnimcAxisStatusBlendComplete
- TnimcAxisStatusFollowingErrorExceeded
- TnimcAxisStatusVelocityThresholdExceeded
- TnimcAxisStatusMoving
- TnimcAxisStatusDirectionForward

data is the value for the attribute you are reading in the following structure:

struct{

i32 longData; u8 boolData; f64 doubleData; } TnimcData;

Based on the attribute, the correct member of TnimcData will be returned by the motion controller. The values of the other elements in the structure are undefined.

Read **boolData** in the TnimcData structure to retrieve the axis execution status.

Using This Function

Use this function to read the specified execution status information of an axis.



Note This function should *not* be used for event checking. Only use the information returned by this function for display purposes. Refer to <u>Advanced Read Functions</u> for more information.

For **TnimcAxisStatusAxisActive**, TnimcTrue indicates that the motor is on and TnimcFalse indicates that the motor is off.

For TnimcAxisStatusMoveComplete,

TnimcAxisStatusProfileComplete, and

TnimcAxisStatusBlendComplete, TnimcTrue indicates that the move, profile, or blend is complete and TnimcFalse indicates that the move, profile, or blend is incomplete.

For **TnimcAxisStatusFollowingErrorExceeded**, TnimcTrue indicates that the axis exceeded the programmed following error limit and TnimcFalse indicates that the axis following error is below the programmed following error limit.

Note If you are running a contoured move or slave axis move with a stepper axis, it is possible that the contour velocity or the slave axis geared velocity exceeds the maximum step rate of the controller. In this case, the controller kills the axis and sets the following error status to true. Refer to the *Specifications* section of your controller user manual for the maximum step rate of your controller.

For **TnimcAxisStatusVelocityThresholdExceeded**, TnimcTrue indicates that the absolute value of filtered axis velocity is above the threshold and TnimcFalse indicates that the velocity is below the threshold.

For **TnimcAxisStatusMoving**, TnimcTrue indicates that the axis is moving and TnimcFalse indicates that the axis is not moving.

For **TnimcAxisStatusDirectionForward**, TnimcTrue indicates that the axis is moving forward and TnimcFalse indicates that the axis is not moving forward.



Note Refer to Function Execution Times for benchmark timing

information about your controller.

nimcReadCaptureCompareData

Device Compatibility

Read Capture Compare Data

Usage

status = nimcReadCaptureCompareData(TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, i32 index, TnimcCaptureCompareData attribute, TnimcData* data);

Purpose

Reads position compare or position capture data from the selected axis.

Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to read
index	i32	encoder to configure
attribute	TnimcCaptureCompareData	attribute to read
data	TnimcData*	the value for the attribute you are reading

Parameter Discussion

axisHandle is the axis to read with this function. Valid values are 1 through 30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError).

index is the encoder to configure. Valid value is 1 for the primary encoder.

attribute is the attribute to read. The following are valid attributes:

- TnimcCaptureCompareDataCapturedPosition
- TnimcCaptureCompareDataCaptureOccurred
- TnimcCaptureCompareDataCompareOccurred

data is the value for the attribute you are reading in the following structure:

struct{ i32 longData; u8 boolData; f64 doubleData;

```
} TnimcData;
```

Based on the attribute, the correct member of TnimcData will be returned by the motion controller. The values of the other elements in the structure are undefined.

Using This Function

Use this function to read position compare or position capture data from the selected axis.



Caution If this function uses reflective memory, it should *not* be used for event checking. In this case, only use the information returned by this function for display purposes. Refer to Reflective Memory Functions for more information.

For TnimcCaptureCompareDataCaptureOccurred and TnimcCaptureCompareDataCompareOccurred, the boolData element contains the position capture or position compare status. TnimcTrue indicates that the position capture or position compare has occurred and TnimcFalse indicates that the position capture or position compare has not occurred.

For TnimcCaptureCompareDataCapturedPosition, read doubleData in the TnimcData structure to retrieve the position capture position.

N

Note Refer to Function Execution Times for benchmark timing information about your controller.

nimcReadCoordinateData

Device Compatibility

Read Coordinate Data

Usage

status = nimcReadCoordinateData(TnimcDeviceHandle deviceHandle, TnimcCoordinateHandle coordinateHandle, TnimcCoordinateData attribute, TnimcData* data);

Purpose

Reads the velocity or following error for a specified coordinate.

Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
coordinateHandle	TnimcCoordinateHandle	coordinate to read
attribute	TnimcCoordinateData	attribute to read
data	TnimcData*	the value for the attribute you are reading

Parameter Discussion

coordinateHandle is the coordinate to read with this function. Valid values are coordinates 1 through 15. On motion controllers that support fewer than thirty axes, reading non-existent coordinates returns error 70006 (NIMC_badResourceIDOrAxisError).

attribute is the attribute to read. The following are valid attributes:

• TnimcCoordinateDataVelocity

TnimcCoordinateDataFollowingError

data is the value for the attribute you are reading in the following structure:

struct{

i32 longData; u8 boolData; f64 doubleData;

} TnimcData;

Based on the attribute, the correct member of TnimcData will be returned by the motion controller. The values of the other elements in the structure are undefined.

Read **doubleData** in the TnimcData structure to retrieve the vector space velocity and following error.

Using This Function

Use this function to read the velocity or following error for a specified coordinate.

Caution If this function uses reflective memory, it should *not* be used for event checking. In this case, only use the information returned by this function for display purposes. Refer to Reflective Memory Functions for more information.

For velocity, this function returns the filtered coordinate velocity for the coordinate move. The velocity is calculated using the root-mean-square of the filtered velocities of the individual axes that make up the coordinate.

For following error, this function returns the instantaneous difference between the commanded trajectory position and the actual feedback position in counts for servo systems and steps for stepper systems. The coordinate following error is calculated using the root-mean-square of the following errors for the individual axes that make up the coordinate.

M **Note** Refer to Function Execution Times for benchmark timing information about your controller.

nimcReadCoordinatePosition Device Compatibility

Read Coordinate Position

Usage

status = nimcReadCoordinatePosition(TnimcDeviceHandle deviceHandle, TnimcCoordinateHandle coordinateHandle, f64* position, u32 lengthPosition, u32 *fetched);

Purpose

Reads the position of all axes in a coordinate.

Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
coordinateHandle	TnimcCoordinateHandle	coordinate to read
position[]	f64	array containing position data
lengthPosition	u32	number of axes contained in the coordinate
fetched*	u32	number of axes retrieved

Parameter Discussion

coordinateHandle is the coordinate to read with this function. Valid values are coordinates 1 through 15. On motion controllers that support fewer than thirty axes, reading non-existent coordinates returns error 70006 (NIMC_badResourceIDOrAxisError).

position[] is the array containing the coordinate position information.

lengthPosition is the number of axes contained in the coordinate. If you do not know the correct size for **lengthPosition**, using a value of 0 returns error 70028 (NIMC_insufficientSizeError) but provides the correct size to use in **fetched**.

fetched contains the number of axes retrieved.

Using This Function

Use this function to read the position of all axes in a coordinate.



Caution If this function uses reflective memory, it should *not* be used for event checking. In this case, only use the information returned by this function for display purposes. Refer to <u>Reflective</u> <u>Memory Functions</u> for more information.

For servo axes, this function returns the primary feedback position in quadrature counts. For open-loop stepper axes, it returns the number of steps generated. For closed-loop stepper axes, it converts the primary feedback position from counts to steps and then returns the value in steps. Closed-loop stepper axes require correctly loaded values of steps per revolution and counts per revolution to function correctly.



Note For closed-loop axes, this function always returns the position of the primary feedback resource.

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

nimcReadCoordinateStatus

Device Compatibility

Read Coordinate Status

Usage

status = nimcReadCoordinateStatus(TnimcDeviceHandle deviceHandle, TnimcCoordinateHandle coordinateHandle, TnimcCoordinateStatus attribute, TnimcData* data);

Purpose

Reads the specified execution status information of a coordinate.

Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
coordinateHandle	TnimcCoordinateHandle	coordinate to read
attribute	TnimcCoordinateStatus	attribute to read
data	TnimcData*	the value for the attribute you are reading

Parameter Discussion

coordinateHandle is the coordinate to read with this function. Valid values are coordinates 1 through 15. On motion controllers that support fewer than thirty axes, reading non-existent coordinates returns error 70006 (NIMC_badResourceIDOrAxisError).

attribute is the attribute to read. The following are valid attributes:

- TnimcCoordinateStatusMoveComplete
- TnimcCoordinateStatusProfileComplete
- TnimcCoordinateStatusBlendComplete
- TnimcCoordinateStatusFollowingErrorExceeded

data is the value for the attribute you are reading in the following structure:

struct{

i32 longData; u8 boolData; f64 doubleData; TnimeData;

} TnimcData;

Based on the attribute, the correct member of TnimcData will be returned by the motion controller. The values of the other elements in the structure are undefined.

Read **boolData** in the TnimcData structure to retrieve the coordinate execution status.

Using This Function

Use this function to read the specified execution status information of a coordinate.



Caution If this function uses reflective memory, it should *not* be used for event checking. In this case, only use the information returned by this function for display purposes. Refer to Reflective Memory Functions for more information.

For TnimcCoordinateStatusMoveComplete, TnimcCoordinateStatusProfileComplete, and

TnimcCoordinateStatusBlendComplete, TnimcTrue indicates that the move, profile, or blend is complete and TnimcFalse indicates that the move, profile, or blend is incomplete.

For **TnimcCoordinateStatusFollowingErrorExceeded**, TnimcTrue indicates that one or more of the axes in the coordinate exceeded the programmed following error limit and TnimcFalse indicates that all of the axes in the coordinate have following error below the programmed following error limit.



Note If you are running a contoured move or slave axis move with a stepper axis, it is possible that the contour velocity or the slave axis geared velocity exceeds the maximum step rate of the controller. In this case, the controller kills the axis and sets the following error status to true. Refer to the Specifications section of your controller user manual for the maximum step rate of your controller.

nimcReadDigitalIOData

Device Compatibility

Read Digital I/O Data

Usage

status = nimcReadDigitalIOData(TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, u32 line, TnimcDigitalIOData attribute, TnimcData* data);
Purpose

Reads the attribute on a single digital I/O line.



Tip Refer to the Remarks section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to read
line	u32	digital line to read attribute from
attribute	TnimcDigitallOData	attribute to read
data	TnimcData*	the value for the attribute you are reading

Parameter Discussion

axisHandle is the axis to read with this function. Valid values are 1 through 30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error

-70006 (NIMC_badResourceIDOrAxisError).

line is the digital line of the axis to read. Valid range is 1 to 32.

attribute is the attribute to read. The following are valid attributes:

attribute	Description
TnimcDigitalIODataOutputActiveState	Returns the active state of the specified output line. TnimcTru indicates active low/active ope TnimcFalse indicates active high/active closed.
TnimcDigitalIODataInputActiveState	Returns the active state of the specified input line. TnimcTrue indicates active low/active ope TnimcFalse indicates active high/active closed.
TnimcDigitallODataOutputActive	Returns the current output sta the specified line. TnimcTrue indicates the output line is acti TnimcFalse indicates the outpu is inactive.
	Note Reading this attribu on a line configured for in returns error -70102 (NIMC_wrongIODirectionE
TnimcDigitallODataInputActive	Returns the input state of the specified line. TnimcTrue indicates the input line is active. TnimcF indicates the input line is inact Note Reading this attribu on a line configured for ou returns error -70102 (NIMC_wrongIODirectionE)

TnimcDigitallODatalOConfigu	reAsInput Returns the direction of the
	specified line. TnimcTrue indic
	input, TnimcFalse indicates ou

data is the value for the attribute you are reading in the following structure:

struct{

i32 longData; u8 boolData; f64 doubleData;

} TnimcData;

Based on the attribute, the correct member of TnimcData will be returned by the motion controller. The values of the other elements in the structure are undefined.

Using This Function

Use this function to read the attribute on a single digital I/O line. The digital I/O lines are divided among the axes.



Caution If this function uses reflective memory, it should *not* be used for event checking. In this case, only use the information returned by this function for display purposes. Refer to Reflective Memory Functions for more information.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following includes considerations you must make when you are using this function with a 73xx controller:

- The PCI-7390 does not support **TnimcDigitalIODataIOConfigureAsInput** attribute.
- The PCI-7390 has dedicated direction I/O lines. On these controllers, the pin you read (input or output) is determined by the value specified in **line** and **attribute**.

For example, to read the input active state attribute of General-Purpose Input Bit 1 on axis 2, call the Read Digital I/O Data function with the following parameters:

```
axisHandle = 2
line = 1
attribute = TnimcDigitallODataInputActiveState
```

To read the output active state of General-Purpose Output Bit 1 on axis 2, call the Read Digital I/O Data function with the following parameters:

```
axisHandle = 2
line = 1
attribute = TnimcDigitallODataOutputActiveState
```

 7330, 7340, and 7350 controllers have configurable direction I/O lines. On these controllers, the line number is translated to port and line number. For example, a call to Read Digital I/O Data on a four axis controller with eight digital I/O ports and

```
axisHandle = NIMC_AXIS2
line = 15
attribute = TnimcDigitallODataInputActiveState
```

reads the active state of port 6 bit 7.

• The following table lists the port to axis and line mapping for 7350 controllers:

Number of Axes	Axis Number	Line 0 to 7	Line 8 to 15	Line 16 to 23	Line 24 to 31
2	1	Port 1	Port 3	Port 5	Port 7
	2	Port 2	Port 4	Port 6	Port 8
4	1	Port 1	Port 5		
	2	Port 2	Port 6		
	3	Port 3	Port 7		
	4	Port 4	Port 8	—	—
6	1	Port 1			
	2	Port 2	—	—	—
	3	Port 3			
	4	Port 4	—	—	_
	5	Port 5	Port 7	—	—
	6	Port 6	Port 8		
8	1	Port 1	—	—	_
	2	Port 2	—	—	—
	3	Port 3			
	4	Port 4			
	5	Port 5			
	6	Port 6			
	7	Port 7			
	8	Port 8			

• The following table lists the port to axis and line mapping for 7330 and 7340 controllers:

Number of Axes	Axis Number	Line 0 to 7	Line 8 to 15
2	1	Port 1	Port 3
	2	Port 2	Port 4
4	1	Port 1	
	2	Port 2	

3	Port 3	
4	Port 4	

NI SoftMotion Controller Considerations

The following includes considerations you must make when you are using this function with the NI SoftMotion Controller:

The NI SoftMotion Controller does not support the **TnimcDigitalIODatalOConfigureAsInput** attribute.

nimcReadEncoderData

Device Compatibility

Read Encoder Data

Usage

status = nimcReadEncoderData(TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, i32 index TnimcEncoderData attribute, TnimcData* data);

Purpose

Reads status and data from the selected encoder.

Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to read encoder information from
index	i32	encoder index
attribute	TnimcEncoderData	attribute to read
data	TnimcData*	the value for the attribute you are reading

Parameter Discussion

axisHandle is the axis to read with this function. Valid values are 1 through 30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError).

index is the encoder index value. Valid value is 1 for the primary encoder.

attribute is the attribute to read. The following are valid attributes:

- TnimcEncoderDataPosition
- TnimcEncoderDataIndexPosition
- TnimcEncoderDataIndexCaptureOccurred

data is the value for the attribute you are reading in the following structure:

struct{

i32 longData; u8 boolData; f64 doubleData;

```
} TnimcData;
```

Based on the attribute, the correct member of TnimcData will be returned by the motion controller. The values of the other elements in the structure are undefined.

Read **doubleData** in the TnimcData structure to retrieve the encoder position or encoder index position.

Using This Function

Use this function to read status and data from the selected encoder.



For encoder position, this function returns the quadrature count value of the encoder mapped to the selected axis.

For encoder index position, this function returns the captured index position after a <u>Find Index</u> search.

Read **boolData** in the TnimcData structure to retrieve the encoder index capture status. A value of TnimcTrue indicates that the encoder index position capture occurred. A value of TnimcFalse indicates that the encoder index position was not captured.



Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

nimcReadMotionIOData

Device Compatibility

Read Motion I/O Data

Usage

status = nimcReadMotionIOData(TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, TnimcMotionIOData attribute, TnimcData* data);

Purpose

Reads status and data from the motion inputs and outputs.



Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to read
attribute	TnimcMotionIOData	attribute to read
data	TnimcData*	the value for the attribute you are reading

Parameter Discussion

axisHandle is the axis to read data from. Valid values are 1 through 30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError).

attribute is the attribute to read. The following are valid attributes:

- TnimcMotionIODataForwardLimitActive
- TnimcMotionIODataReverseLimitActive
- TnimcMotionIODataForwardSoftwareLimitActive
- TnimcMotionIODataReverseSoftwareLimitActive
- TnimcMotionIODataHomeInputActive
- TnimcMotionIODataInhibitInActive
- TnimcMotionIODataInPositionActive
- TnimcMotionIODataDriveReadyActive
- TnimcMotionIODataInhibitOutActive

data is the value for the attribute you are reading in the following structure:

struct{

i32 longData; u8 boolData; f64 doubleData; } TnimcData;

Based on the attribute, the correct member of TnimcData will be returned by the motion controller. The values of the other elements in the structure are undefined.

Read **boolData** in the TnimcData structure to poll the status of the limit, home, inhibit-in, in-position, and drive ready inputs, and the status of the inhibit-out output. TnimcTrue indicates that the signal is active and TnimcFalse indicates that the signal is inactive.

Using This Function

Use this function to read the status and data from the motion inputs and outputs on the motion controller. You can read the status of these motion inputs and outputs at any time, regardless of whether they are enabled. The limit and home input status during a Find Reference move is undefined. Refer to <u>Write Motion I/O Data</u> for more information.



Caution If this function uses reflective memory, it should *not* be used for event checking. In this case, only use the information returned by this function for display purposes. Refer to <u>Reflective</u> <u>Memory Functions</u> for more information.



Note The active state of the signals determines if an On state is active high/active closed or active low/active open. Refer to <u>Write</u> <u>Motion I/O Data</u> for more information about active state and general information about the signals.



Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx controller:

- The minimum pulse width for the inhibit-in input to be detected is approximately 1 ms.
- On the NI PCI-7390 controller, the **TnimcMotionIODataInhibitInActive** attribute does not return a valid value unless the inhibit-in signal is enabled.
- On the NI PCI-7390 controller, the **TnimcMotionIODataInPositionActive** attribute does not return a valid value unless the in-position signal is enabled.

NI SoftMotion Controller Considerations

The following includes considerations you must make when you are using this function with the NI SoftMotion Controller:

The NI SoftMotion Controller does not support the **TnimcMotionIODataInPositionActive**, **TnimcMotionIODataDriveReadyActive**, and **TnimcMotionIODataInhibitOutActive** attributes.

Advanced Write

You can use the advanced write functions for specialized I/O operations involving the digital I/O on a controller, for advanced motion I/O operations, or to configure the trajectory data for a move.



Note To use the advanced write functions, you must include nimotion.h and link to nimotion.lib.

Write Capture Compare Data

Write Digital I/O Data

Write Motion I/O Data

Write Trajectory Data

nimcWriteCaptureCompareData

Device Compatibility

Write Capture Compare Data

Usage

status = nimcWriteCaptureCompareData (TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, i32 index, TnimcCaptureCompareData attribute, TnimcData* data);

Purpose

Loads position compare or position capture data for the selected axis.

Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to configure
index	i32	encoder to configure
attribute	TnimcCaptureCompareData	attribute to read
data	TnimcData*	the value for the attribute you are reading

Parameter Discussion

axisHandle is the axis to configure with this function. Valid values are 1 through 30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError).

index is the encoder to configure. Valid value is 1 for the primary encoder.

attribute is the attribute to load. The following is the valid attribute:

TnimcCaptureCompareDataComparePulseWidth

data is the value for the attribute you are loading in the following structure:

struct{

i32 longData; u8 boolData; f64 doubleData; TraimaData;

} TnimcData;

Based on the attribute, the correct member of TnimcData must be set as follows:

• For TnimcCaptureCompareDataComparePulseWidth set **doubleData** in the TnimcData structure to the value of the desired position compare output pulse width, in seconds. For example, enter 0.00001 seconds for a 10 μ s pulse width. Valid values are the default value of 0.000002 s (200 ns) and between 0.000001 s (1 μ s) and 0.065535 s (65,535 μ s) in 1 μ s intervals.

Using This Function

Use this function to load configuration parameters for position capture and compare on the motion controller. This function allows you to set all attributes on a per axis basis.

The **TnimcCaptureCompareDataComparePulseWidth** attribute sets the pulse width of the position compare signal. Use this attribute to configure the pulse width of breakpoints that trigger devices that have specific minimum active pulse requirements.



Note A breakpoint across RTSI always generates an active high pulse of 200 ns duration.

Caution The position compare (breakpoint) frequency depends on the velocity and distance between position compare outputs and is limited by the pulse width. The position compare output frequency *must* meet the requirements specified below for correct operation.

The position compare output pulse must not overlap with the next position compare event. If the position compare output pulse overlaps with the next position compare event, position compare generation is stopped and the position compare output is disabled. To recover from this situation change the pulse width, velocity, or distance between position compare outputs. In addition, the position compare output frequency cannot be greater than the maximum value allowed for the specified position compare type. Therefore, the position compare output frequency, in hertz, at the specified pulse width, in seconds, must conform to the following two conditions:

PID rate of 62.5 µs to 250 µs

position compare frequency (hertz) < 1 pulse width (seconds)	AND	Position Compare (Breakpoint) Output Type	Maxiı Frequ
		Single	150 H
		Buffered,	2 kHz

Buffered,	1 kHz
PID rate	
greater	
than 250 μs	
Periodic	4 MHz

nimcWriteDigitalIOData

Device Compatibility

Write Digital I/O Data

Usage

status = nimcWriteDigitalIOData(TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, u32 line, TnimcDigitalIOData attribute, TnimcData* data);

Purpose

Sets the attribute on a single digital I/O line.



Tip Refer to the <u>Remarks</u> section for information about how the behavior of this function differs between controllers.

Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to configure digital lines on
line	u32	digital line to modify attribute
attribute	TnimcDigitallOData	attribute to set
data	TnimcData*	the data for the attribute you are writing

Parameter Discussion

axisHandle is the axis on which to configure digital lines with this function. Valid values are 1 through 30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error -70006 (NIMC_badResourceIDOrAxisError).

line is the digital line of the axis to apply the attribute. Valid range is 1 to 32.

attribute	Description
TnimcDigitallODataOutputActiveState	Sets the active state of the specified output line.
TnimcDigitallODataInputActiveState	Sets the active state of the specified input line.
TnimcDigitallODataOutputActive	 Sets the current output state of the specified line. Note Setting this attribute on a line configured for input returns an error.
TnimcDigitalIODatalOConfigureAsInput	Sets the direction of the specified line.

attribute is the attribute to set. The following are valid attributes:

data is the value for the attribute you are writing in the following structure:

struct{

i32 longData; u8 boolData; f64 doubleData;

} TnimcData;

Based on the attribute, the correct member of TnimcData must be set as follows:

• For **TnimcDigitallODataOutputActiveState**, set **boolData** in the TnimcData structure to TnimcTrue to set the active state to active low/active open, and set it to TnimcFalse to set the active state to active high/active closed.

- For **TnimcDigitallODataInputActiveState**, set **boolData** in the TnimcData structure to TnimcTrue to set the active state to active low/active open, and set it to TnimcFalse to set the active state to active high/active closed.
- For **TnimcDigitallODataOutputActive**, set **boolData** in the TnimcData structure to TnimcTrue to configure the output line as active, and set it to TnimcFalse to configure the output line as inactive.
- For **TnimcDigitallODatalOConfigureAsInput**, set **boolData** in the TnimcData structure to TnimcTrue to configure the line for input, and set it to TnimcFalse to configure the line for output.

Using This Function

Use this function to set the attribute on a single digital I/O line. The digital I/O lines are divided among the axes.
Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following includes considerations you must make when you are using this function with a 73xx controller:

- The PCI-7390 does not support the **TnimcDigitalIODataIOConfigureAsInput** attribute.
- The PCI-7390 has dedicated direction I/O lines. On these controllers, the pin you write to (input or output) is determined by the value specified in **line** and **attribute**.

For example, to change the input active state on line 1 on axis 2, call Write Digital I/O Data with the following parameters:

```
axisHandle = 2
line = 1
attribute = TnimcDigitallODataInputActiveState
```

The active state is applied to axis 2 General-Purpose Input Bit 1.

To change the output active state of line 1 on axis 2, call Write Digital I/O Data with the following parameters:

```
axisHandle = 2
line = 1
attribute = TnimcDigitallODataOutputActiveState
```

The active state is applied to axis 2 General-Purpose Output Bit 1.

 7330, 7340, and 7350 controllers have configurable direction I/O lines. On these controllers, the line number is translated to port and line number. For example, a call to Write Digital I/O Data on a four axis controller with eight digital I/O ports with

```
axisHandle = 2
line = 15
attribute = TnimcDigitallODataInputActiveState
```

applies the active state to port 6 bit 7.

• The following table lists the port to axis and line mapping for 7350

controllers:

Number of Axes	Axis Number	Line 0 to 7	Line 8 to 15	Line 16 to 23	Line 24 to 31
2	1	Port 1	Port 3	Port 5	Port 7
	2	Port 2	Port 4	Port 6	Port 8
4	1	Port 1	Port 5		
	2	Port 2	Port 6		
	3	Port 3	Port 7	—	—
	4	Port 4	Port 8	—	—
6	1	Port 1	—	—	—
	2	Port 2	—	—	—
	3	Port 3	—	—	—
	4	Port 4	—	—	—
	5	Port 5	Port 7	—	—
	6	Port 6	Port 8	—	—
8	1	Port 1	—	—	—
	2	Port 2	—	—	—
	3	Port 3	—	—	—
	4	Port 4	—	—	—
	5	Port 5			
	6	Port 6			
	7	Port 7			
	8	Port 8			

• The following table lists the port to axis and line mapping for 7330 and 7340 controllers:

Number of Axes	Axis Number	Line 0 to 7	Line 8 to 15
2	1	Port 1	Port 3
	2	Port 2	Port 4
4	1	Port 1	
	2	Port 2	

3	Port 3	—
4	Port 4	_

NI SoftMotion Controller Considerations

The following includes considerations you must make when you are using this function with the NI SoftMotion Controller:

The NI SoftMotion Controller does not support the **TnimcDigitalIODatalOConfigureAsInput** attribute.

nimcWriteMotionIOData

Device Compatibility

Write Motion I/O Data

Usage

status = nimcWriteMotionIOData(TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, TnimcMotionIOData attribute, TnimcData* data);

Purpose

Loads configuration parameters for the motion inputs and outputs.



Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to configure
attribute	TnimcMotionIOData	attribute to set
data	TnimcData*	the value for the attribute you are loading

Parameter Discussion

axisHandle is the axis to configure with this function. Valid values are 1 through 30. On motion controllers that support fewer than thirty axes, configuring non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError).

attribute is the attribute to load. The following are valid attributes:

- TnimcMotionIODataForwardLimitEnable
- TnimcMotionIODataReverseLimitEnable
- TnimcMotionIODataForwardSoftwareLimitEnable
- TnimcMotionIODataReverseSoftwareLimitEnable
- TnimcMotionIODataHomeInputEnable
- TnimcMotionIODataForwardLimitActiveState
- TnimcMotionIODataReverseLimitActiveState
- TnimcMotionIODataHomeInputActiveState
- TnimcMotionIODataForwardSoftwareLimitPosition
- TnimcMotionIODataReverseSoftwareLimitPosition
- TnimcMotionIODataInhibitInEnable
- TnimcMotionIODataInhibitInActiveState
- TnimcMotionIODataInPositionActiveState
- TnimcMotionIODataPulseAlarmClear
- TnimcMotionIODataAlarmClearPulseWidth
- TnimcMotionIODataDriveReadyEnable
- TnimcMotionIODataInhibitOutEnable
- TnimcMotionIODataInhibitOutTotemPole
- TnimcMotionIODataInhibitOutActiveState

data is the value for the attribute you are loading in the following structure:

struct{

i32 longData; u8 boolData; f64 doubleData; TnimeData;

} TnimcData;

Based on the attribute, the correct member of TnimcData must be set as

follows:

- For **TnimcMotionIODataForwardLimitEnable** and **TnimcMotionIODataReverseLimitEnable**, set **booIData** in the TnimcData structure to TnimcTrue to enable the hardware limit, and TnimcFalse to disable the hardware limit.
- For TnimcMotionIODataForwardSoftwareLimitEnable and TnimcMotionIODataReverseSoftwareLimitEnable, set boolData in the TnimcData structure to TnimcTrue to enable the software limit, and TnimcFalse to disable the software limit.
- For **TnimcMotionIODataHomeInputEnable**, set **boolData** in the TnimcData structure to TnimcTrue to enable the home input, and TnimcFalse to disable home input.
- For TnimcMotionIODataForwardLimitActiveState, TnimcMotionIODataReverseLimitActiveState, and TnimcMotionIODataHomeInputActiveState, set boolData in the TnimcData structure to TnimcFalse to set the active state to active high/active closed and TnimcTrue for active low/active open.
- For TnimcMotionIODataForwardSoftwareLimitPosition and TnimcMotionIODataReverseSoftwareLimitPosition, set doubleData in the TnimcData structure to the value at which you want the software limit to trigger. Software limit positions can be anywhere within the 32-bit position range, $-(2^{31})$ to $+(2^{31}-1)$.
 - Note TnimcMotionIODataForwardSoftwareLimitPosition cannot be less than TnimcMotionIODataReverseSoftwareLimitPosition.
- For **TnimcMotionIODataInhibitInEnable**, set **boolData** in the TnimcData structure to TnimcTrue to enable inhibit-in, and set it to TnimcFalse to disable inhibit-in. When the inhibit-in signal is active and enabled, the motion controller kill stops the axis. Refer to <u>Stop Motion</u> for information about kill stop. Connect the inhibit-in signal to the Servo Alarm signal from the drive.
- For **TnimcMotionIODataInhibitInActiveState**, set **boolData** in the TnimcData structure to TnimcFalse to set the active state to active high/active closed, and set it to TnimcTrue to set the active state to active low/active open.
- For **TnimcMotionIODataInPositionActiveState**, set **boolData** in the TnimcData structure to TnimcFalse to set the active state to

active high/active closed, and set it to TnimcTrue to set the active state to active low/active open.

• For **TnimcMotionIODataPulseAlarmClear**, set **boolData** in the TnimcData structure to TnimcTrue to pulse the alarm clear signal on for the time set in **TnimcMotionIODataAlarmClearPulseWidth**.



• For **TnimcMotionIODataAlarmClearPulseWidth**, set **doubleData** in the TnimcData structure to the time, in seconds, that you want the alarm clear signal to be active. Valid range is 0 – 60 s.



Note The pulse is software-timed and is accurate to approximately ±1 ms.

- For **TnimcMotionIODataDriveReadyEnable** set **boolData** in the TnimcData structure to TnimcTrue to enable the drive ready input, and set it to TnimcFalse to disable the drive ready input.
- For **TnimcMotionIODataInhibitOutEnable** set **boolData** in the TnimcData structure to TnimcTrue to enable the inhibit-out output, and set it to TnimcFalse to disable the inhibit-out output.
- For **TnimcMotionIODataInhibitOutTotemPole**, set **boolData** in the TnimcData structure to TnimcFalse to select open collector as the output configuration, and set it to TnimcTrue to select totem pole as the output configuration for the inhibit outputs.
- For **TnimcMotionIODataInhibitOutActiveState**, set **boolData** in the TnimcData structure to TnimcFalse to set the active state to active high/active closed, and set it to TnimcTrue to set the active state to active low/active open.

Using This Function

Use this function to load configuration parameters for the motion inputs and outputs on the motion controller. This function allows you to set all attributes on a per axis basis.

Hardware limit inputs, home inputs, software position limits, inhibit inputs, in-position inputs, drive ready inputs, and inhibit outputs are enhancements on the NI motion controllers and are not required for basic motion control. With the exception of the <u>Find Reference</u> function, you can operate all motion control functions without enabling or using these features. The Find Reference function requires enabled limit and home inputs for operation. Find Reference does not require enabled software limits.

Caution National Instruments recommends using limits for personal safety, as well as to protect the motion system.

The active state for each hardware limit, home, inhibit input, in-position input, and inhibit output can be configured as either active low/active open or active high/active closed.

When configured as active low, the input or output is active when there is a low signal on the pin. Conversely, active high means that the input or output is active when there is a high signal on the pin.

Configuring an active state of active open or active closed does not correspond to the level of the signal on the input or output pin. Instead, an active open state means that the input or output is active when current is not flowing through the optocoupled input. Conversely, an active closed state means that the input or output is active when current is flowing through the optocoupled input.

Forward and Reverse Limits, and Home Inputs

The hardware limit inputs are typically connected to end-of-travel limit switches or sensors. An enabled limit input causes a halt stop on the axis when the input becomes active. Active limit inputs also prohibit attempts to start motion that would cause additional travel in the direction of the limit. You also can use limit inputs as general-purpose inputs and read their status with the <u>Read Motion I/O Data</u> function.



Note For the end-of-travel limits to function correctly, the forward

limit switch or sensor must be located at the positive (count up) end of travel and the reverse limit at the negative (count down) end of travel.

An enabled home input causes a halt stop on the axis when the input becomes active. Refer to <u>Stop Motion</u> for information about halt stop. You also can use a home input as a general-purpose input and read its status with the <u>Read Motion I/O Data</u> function.

Similar to hardware limits, software limits are often used to restrict the range of travel further and avoid hitting the hardware limit switches. An enabled software limit causes the axis to smoothly decelerate to a stop when the limit position is reached or exceeded. To enable the software limits, you must first find the reference point using the Find Reference function. The forward software limit is considered active if the current position is greater than or equal to the forward software limit position. The reverse software limit is considered active if the current position is less than or equal to the reverse software limit position.

Software limits are often used to restrict the range of travel and avoid hitting the hardware end-of-travel limit switches. For example, the motor can travel at a high velocity until hitting the software limit switch, and then move more slowly until it hits the hardware limit switch.

 $\overline{\mathbb{N}}$

Note After an axis has stopped when it encounters a software limit switch, you can still move further in the same direction if you command the axis to do so. This behavior is not possible with hardware limits, but is appropriate for software limits.

Note If any axis in a coordinate move exceeds an enabled software limit position, all axes in the coordinate decelerate to a stop.

Even when disabled, you can poll the software limits with the host computer to warn of an out-of-range position. Refer to the <u>Read Motion</u> <u>I/O Data</u> function for information about reading the forward and reverse software limit status.

Inhibit Input

Use the inhibit-in input to connect a drive alarm/servo alarm signal to the motion controller so that when an alarm or other drive fault occurs, the motion is stopped using a kill stop. A kill stop asserts inhibit-out, disabling

the control loop and zeroing the DAC so that frictional forces stop the motion.

In-Position Input

Use the In-Position drive signal when the drive is closing the position loop, for example a servo drive that accepts p-command or digital signals. This signal tells the motion controller when the drive considers the motor to be at the commanded position.

When the In-Position drive signal is configured, the Move Complete status is linked to the state of the in-position input. The move is not considered complete unless the in-position input is active.

Note Refer to <u>Write Trajectory Data</u> to add or remove this signal from the move complete criteria.

Alarm Clear Output

Use the Alarm Clear Output to pulse the alarm clear signal on one axis of your system. Asserting the Alarm Clear signal clears the drive faults. The output must remain active for a minimum time, which varies depending on your drive. Refer to your drive documentation for more information. Use the **TnimcMotionIODataAlarmClearPulseWidth** attribute to set this time.

Drive Ready Input

The drive ready input can be connected to the drive ready or servo ready output of the drive. The drive ready output is active during normal operation. If the drive ready input signal is inactive, calling Start Motion returns an error. All other VIs can be executed.

Tip Check the status of this input at power-on to verify that the controller is ready to start a move.

Inhibit Output

When enabled, a per-axis inhibit output is linked to the motor off state of the corresponding axis. A killed axis (motor off) forces the corresponding inhibit output On. When the axis is active, the inhibit output is Off. Note that these On and Off states are logical states. The actual state depends on the polarity of the system.

Inhibit outputs are typically used to disable the servo amplifier or stepper drive for power saving, safety, or specific application reasons.

Note Killing a servo axis also zeros its DAC output. With torque block amplifiers, this means that the motor freewheels regardless of if the amplifier is disabled. With velocity block servo amplifiers or stepper drives, the motor does not freewheel unless the amplifier/drive is disabled with the inhibit output.

You also can use inhibit outputs as general-purpose outputs. Disabled inhibit outputs ignore the state of their corresponding axis and can be directly controlled through the Set Inhibit Output MOMO function.

For more information about Open Collector and Totem Pole drive modes, refer to the NI 7350 Hardware User Manual.



Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx controller:

- 7330, 7340, and 7350 controllers do not support the TnimcMotionIODataInhibitInActiveState and TnimcMotionIODataInPositionActiveState attributes. Use Write Digital I/O Data to set the active state for these signals.
- 7330, 7340, and 7390 controllers do not support the **TnimcMotionIODataInhibitOutTotemPole** attribute.
- 7390 controllers do not support the TnimcMotionIODataInhibitOutActiveState attribute. To set the active state on a 7390 use <u>Write Digital I/O Data</u>.
- The inhibit-in signal is software-timed latched at 1 ms. This signal must be active for approximately 1 ms to be detected by the controller.
- The following table lists the drive signals and their implementation on the controller:

Signal Name		Signal	Controller Support			
Controller	Drive	Direction	7330	7340	7350	
Inhibit-Out	Servo On	Output	Dedicated	Dedicated	Dedicated	Mε
Inhibit-In	Servo Alarm	Input	Mappable	Mappable	Mappable	De
Drive Ready	Servo Ready	Input	Mappable	Mappable	Mappable	Mε
In-Position	In- Position	Input	Mappable	Mappable	Mappable	De
Alarm Clear	Alarm Clear	Output	Mappable	Mappable	Mappable	M٤

The active state of a dedicated signal is directly configurable using

this function. The active state of a mappable signal is determined by the active state of the line being used.

For example, to set the active state of the Alarm Clear signal, a mappable signal on all controllers, set the active state of the line the signal is mapped to using the <u>Write Digital I/O Data</u> function.

To set the active state of the inhibit-in signal on a PCI-7390, which has a dedicated line for this signal, use the

TnimcMotionIODataInhibitInActiveState attribute of this function.

NI SoftMotion Controller Considerations

The following list includes considerations you must make when you are using this function with the NI SoftMotion Controller:

- NI SoftMotion Controllers do not have to connect the Alarm signal from the drive in order to use the inhibit-in signal.
- The NI SoftMotion Controller does not support the following attributes:
 - TnimcMotionIODataInPositionActiveState
 - TnimcMotionIODataPulseAlarmClear
 - TnimcMotionIODataAlarmClearPulseWidth
 - TnimcMotionIODataDriveReadyEnable
 - TnimcMotionIODataInhibitOutEnable
 - TnimcMotionIODataInhibitOutTotemPole
 - TnimcMotionIODataInhibitOutActiveState

nimcWriteTrajectoryData

Device Compatibility

Write Trajectory Data

Usage

status = nimcWriteTrajectoryData(TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, TnimcTrajectoryData attribute, TnimcData* data);

Purpose

Configures trajectory generator parameters.



Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to read
attribute	TnimcTrajectoryData	attribute to set
data	TnimcData*	the data for the attribute you are writing

Parameter Discussion

axisHandle is the axis on which to configure trajectory parameters. Valid values are 1 through 30. On motion controllers that support fewer than thirty axes, reading non-existent axes returns error 70006 (NIMC_badResourceIDOrAxisError).

attribute is the attribute to set. The following are valid attributes:

attribute	Descripti
TnimcTrajectoryDataMoveCompleteWhenDeactivated	Adds or removes motor off status as a criteria for move complete.
TnimcTrajectoryDataMoveCompleteWhenNotMoving	Adds or removes motor logical sto status as a criteria for move complete.
TnimcTrajectoryDataMoveCompleteAfterDelay	Adds or removes a user configured settling tin delay as a criteria for move complete.
TnimcTrajectoryDataMoveCompleteTimeDelay	Sets the settling tin delay. Vali values are to 0.255

	seconds.
TnimcTrajectoryDataMoveCompleteWhenInRange	Adds or removes a user configured position range as a criteria for move complete.
TnimcTrajectoryDataMoveCompleteRangeDistance	Sets the position range. Va values are to 32,787 counts, steps, or user units on the NI SoftMotio Controller
TnimcTrajectoryDataMoveCompleteMinimumActiveTime	Sets the minimum time that move complete status mu stay true. Valid value are 0 to 0.255 seconds.
TnimcTrajectoryDataMoveCompleteWhenInPositionActive	Adds or removes t in-positior input state as a criter

data is the value for the attribute you are writing in the following structure: struct{

i32 longData; u8 boolData; f64 doubleData; } TnimcData;

Based on the attribute, the correct member of TnimcData must be set as follows:

- For **TnimcTrajectoryDataMoveCompleteWhenDeactivated**, set **boolData** in the TnimcData structure to TnimcTrue to add motor off status, and set it to TnimcFalse to remove motor off status.
- For TnimcTrajectoryDataMoveCompleteWhenNotMoving, set boolData in the TnimcData structure to TnimcTrue to add motor stop status, and set it to TnimcFalse to remove motor stop status.
- For **TnimcTrajectoryDataMoveCompleteAfterDelay**, set **boolData** in the TnimcData structure to TnimcTrue to enable a user-defined delay, and set it to TnimcFalse to disable a user-defined delay.
- For **TnimcTrajectoryDataMoveCompleteTimeDelay**, set **doubleData** in the TnimcData structure to the desired time delay, in seconds, to wait for move complete.
- For **TnimcTrajectoryDataMoveCompleteWhenInRange**, set **boolData** in the TnimcData structure to TnimcTrue to enable a position range, and set it to TnimcFalse to disable a position range.
- For **TnimcTrajectoryDataMoveCompleteRangeDistance**, set **doubleData** in the TnimcData structure to the desired position range distance, in counts, steps, or user units for the NI SoftMotion Controller, for move complete.
- For **TnimcTrajectoryDataMoveCompleteMinimumActiveTime**, set **doubleData** in the TnimcData structure to the desired minimum wait time, in seconds, to wait for move complete.
- For **TnimcTrajectoryDataMoveCompleteWhenInPositionActive**, set **boolData** in the TnimcData structure to TnimcTrue to enable using the in-position input as a criteria for move complete, and set

it to TnimcFalse to disable using the in-position input.

The effect of the criteria parameters can be summarized with the following equation:

Move Complete = (Profile Complete [OR When Deactivated])

[AND (Not Moving == True)]

[AND (Delay == Done)]

[AND (| position – target position | < range distance)]

[AND (In-Position == Active)]

where [...] indicates optional criteria.

Using This Function

Use this function to configure trajectory generator parameters for reporting a move complete. When a move is complete on an axis, the corresponding bit in the Move Complete Status (MCS) register is set. Refer to <u>Read Move Complete Status</u> for information about reading the MCS register.

If **TnimcTrajectoryDataMoveCompleteWhenDeactivated** is true, any condition that causes the axis to turn its motor off (a kill or following error trip) satisfies this requirement for Move Complete.

If **TnimcTrajectoryDataMoveCompleteWhenNotMoving** is true, the axis must be logically stopped for the move to be considered complete.

If **TnimcTrajectoryDataMoveCompleteAfterDelay** is true, the axis must wait a user-defined delay after the other criteria are met before the move is considered complete. The

TnimcTrajectoryDataMoveCompleteTimeDelay attribute is typically used to wait the mechanical settling time so that a move is not considered complete until vibrations in the mechanical system have damped out. It also can be used to compensate for PID pull-in time due to the integral term. This pull-in is typically at velocities below the Run/Stop threshold.

🔊 Note You can use the

TnimcTrajectoryDataMoveCompleteAfterDelay attribute to guarantee a minimum time for the False state. The status transitions from Complete to Not Complete at the start of a move and stays in the Not Complete state for at least this delay time even in the case of a zero distance move.

If **TnimcTrajectoryDataMoveCompleteWhenInRange** is true, the axis checks its final stopping position versus its target position and only sets the Move Complete status if the absolute value of the difference is less than **TnimcTrajectoryDataMoveCompleteRangeDistance**.

Finally, if **TnimcTrajectoryDataMoveCompleteWhenInPositionActive** is true, the in-position input signal is used as a criteria for move complete, and the move is only complete if the in-position signal is active.

A non-zero value for

TnimcTrajectoryDataMoveCompleteMinimumActiveTime guarantees

the status stays in the True state for at least this minimum time, even if another move starts immediately. You can use this feature to make sure the host does not miss a Move Complete status when it polls the Move Complete Status register.

P

Tip You can adjust move complete criteria settings in MAX on the **Trajectory Settings** tab.

This function is typically called for each axis prior to using the axis for position control. After the attribute is set, it remains in effect until it is disabled. You can execute this function at any time.

When an axis starts, its corresponding bit in the Move Complete Status register is reset to zero. When the move completes, the bit is set to one. You can check the status of an axis or axes at any time by polling the MCS register. Onboard programs can use this status to automatically sequence moves with the <u>Wait on Event</u> function.

Remarks

This section includes information about how the behavior of this function differs among the controllers that support it.

NI 73xx Controller Considerations

The following list includes considerations you must make when you are using this function with a 73xx motion controller:

- The NI 7344 does not support the **TnimcTrajectoryDataMoveCompleteWhenInPositionActive** attribute.
- 7330, 7340, and 7350 controllers use <u>Configure Motion I/O Map</u> to map the in-position input to a general-purpose I/O line.

NI SoftMotion Controller Considerations

The following includes considerations you must make when you are using this function with the NI SoftMotion Controller:

The NI SoftMotion Controller does not support the **TnimcTrajectoryDataMoveCompleteWhenInPositionActive** attribute.

Advanced Methods

You can use the advanced methods for specialized I/O and move operations, and for specialized communications involving the controller and drive or host.



Note To use the advanced methods, you must include nimotion.h and link to nimotion.lib.

Axis Straight Line Move

Clear Faults

Configure Motion I/O Map

Reset Motion Controller

nimcAxisStraightLineMove

Device Compatibility

Axis Straight Line Move

Usage

status = nimcAxisStraightLineMove(TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, TnimcAxisStraightLineMoveData* data, TnimcMoveConstraints* moveConstraints);

Purpose

Performs a straight line move on an axis.

Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to read
data	TnimcAxisStraightLineMoveData*	the data record containing data for the straight line move
moveConstraints	TnimcMoveConstraints*	move constraints information

Parameter Discussion

axisHandle is the axis to read with this function. Valid values are 1 through 15. On motion controllers that support fewer than fifteen axes, reading non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError).

data is the data record containing straight line move information in the following structure:

struct{

u32 size;

TnimcAxisStraightLineMoveStartMode startMode;

TnimcAxisStraightLineMovePositionMode positionMode; f64 targetPosition;

} TnimcAxisStraightLineMoveData;

TnimcAxisStraightLineMoveStartMode is an enum with the following attributes:

attribute	Description
TnimcAxisStraightLineMoveStartModeDoNotStart	Do no start the straight line move at this time.
TnimcAxisStraightLineMoveStartModeStart	Start the straight line move with the configured parameters.

TnimcAxisStraightLineMovePositionMode is an enum with the following attributes. Refer to <u>Using This Function</u> for more information about the available position modes.

attribute	Description
TnimcAxisStraightLineMovePositionModeAbsolute	Use absolute position mode in the move.
TnimcAxisStraightLineMovePositionModeRelative	Use relative position mode in the move.

TnimcAxisStraightLineMovePositionModeVelocity	Use velocity
	mode in the
	move.

targetPosition is the target position for the straight line move in counts (servo axes) or steps (stepper axes).

moveConstraints contains move constraint information in the following structure:

struct{

u32 size;

f64 velocity; //velocity in counts/s or steps/s

f64 acceleration; //acceleration in counts/s^2 or steps/s^2

f64 deceleration; //deceleration in counts/s^2 or steps/s^2

f64 accelerationJerk; //jerk in counts/s^3 or steps/s^3

f64 decelerationJerk; //jerk in counts/s^3 or steps/s^3

} TnimcMoveConstraints;



Note accelerationJerk and decelerationJerk are not supported at this time. Use <u>Load S-Curve Time</u> or <u>Load Move Constraints</u> to configure S-curve values.

Using This Function

Use this function to perform a straight line move on an axis. The position modes are described in the following sections:

Absolute Position Mode

In absolute position mode, target positions are interpreted with respect to an origin, reference, or zero position. The origin is typically set at a home switch, end of travel limit switch, or encoder index position. An absolute position move uses the specified values of acceleration, deceleration, and velocity to complete a trajectory profile with an ending position equal to the specified absolute target position.

Caution Any single move is limited to between -2^{31} and $2^{31}-1$ counts or steps. An error is generated if you exceed this limit by specifying a target position too far from the current position.

The length of an absolute move depends upon the specified position and the current position when the move is started. If the target position is the same as the current position, no move occurs.

Relative Position Mode

In relative position mode, if a relative position move is started while motion is not in progress, specified target positions are interpreted with respect to the current position at the time the value is specified. A relative position move uses the specified values of acceleration, deceleration, and velocity to complete a trajectory profile with an ending position equal to the sum of the specified relative target position and the starting position.

If a relative move is started while motion is in progress, the new target position is calculated with respect to the target position of the move already in progress (considered to be the new starting position), as if that move had already completed successfully. Motion continues to the new relative position, independent of the actual position location when the new move is started.

Velocity Mode

In velocity mode, the axis moves at the specified velocity until you execute a <u>Stop Motion</u> function, a limit is encountered, or a new velocity is specified and you execute a <u>Start Motion</u> function. Specified target

positions have no effect in velocity mode. The direction of motion is determined by the sign of the specified velocity.

You can update velocity at any time to accomplish velocity profiling. Changes in velocity while motion is in progress use the specified acceleration and deceleration values to control the change in velocity. You can reverse direction by changing the sign of the specified velocity.
Example

To call this function to start an absolute move with a target position of 1000 counts or steps, a velocity of 100 counts/s or steps/s, and an acceleration and deceleration of 50 counts/s or steps/s on axis 2, use the following syntax:

```
TnimcAxisStraightLineMoveData data;
data.size = sizeof(TnimcAxisStraightLineMoveData);
data.targetPosition = 1000.0;
data.startMode = TnimcAxisStraightLineMoveStartModeStart;
data.positionMode = TnimcAxisStraightLineMovePositionModeAbsolute;
```

```
TnimcMoveConstraints moveConstraints;
moveConstraints.size = sizeof(TnimcMoveConstraints);
moveConstraints.velocity = 100.0;
moveConstraints.acceleration = 50.0;
moveConstraints.deceleration = 50.0;
```

nimcAxisStraightLineMove(boardID, 2, &data, &moveConstraints);

- Note You must specify the value for the size elements of data and moveConstraints before calling this function. If you do not properly set the size elements, the function will return error 70023 (NIMC_parameterValueError).
- $\overline{\mathbb{N}}$

Note Refer to <u>Function Execution Times</u> for benchmark timing information about your controller.

nimcClearFaults

Device Compatibility

Clear Faults

Usage

status = nimcClearFaults(TnimcDeviceHandle deviceHandle);

Purpose

Clears all modal errors and drive faults on all axes.

Parameters

Name	Туре	Description
deviceHandle TnimcDeviceHandle		assigned by Measurement &
		Automation Explorer (MAX)

Using This Function

Use this function to clear all modal errors and drive faults on all axes. This function pulses the alarm clear line on all axes that have mapped the alarm clear signal using <u>Configure Motion I/O Map</u>.



Note Refer to <u>Write Motion I/O Data</u> for more information about the alarm clear signal and to change the alarm clear pulse width.

The inhibit-in signal is connected to the Alarm signal of the drive. Monitor the status of the inhibit-in signal to determine if there are any drive faults. Correct the faults, then use this function to clear the alarm state of the drive.

The <u>Initialize Controller</u> function automatically calls this function if the controller is in a power-up state. If you call Initialize Controller and the controller is not in a power-up state, this function is not called.

nimcConfigureMotionIOMap

Device Compatibility

Configure Motion I/O Map

Usage

status = nimcConfigureMotionIOMap(TnimcDeviceHandle deviceHandle, TnimcAxisHandle axisHandle, TnimcMotionIOMap attribute, i32 ioAxis, TnimcMotionIOMapLineType lineType, u32 line);

Purpose

Maps a general purpose I/O line to an alternate function of an axis.

Parameters

Name	Туре	Description
deviceHandle	TnimcDeviceHandle	assigned by Measurement & Automation Explorer (MAX)
axisHandle	TnimcAxisHandle	axis to configure
attribute	TnimcMotionIOMap	the attribute to map to for the input axis
ioAxis	i32	axis that specifies the digital I/O line to map
lineType	TnimcMotionIOMapLineType	line type to map
line	u32	digital line number of the axis

Parameter Discussion

axisHandle is the axis to configure with this function. Valid values are 1 through 8. On motion controllers that support fewer than eight axes, configuring non-existent axes returns error

70006 (NIMC_badResourceIDOrAxisError). Refer to <u>Axes</u> for axis resource IDs.

attribute is the action to map to a digital I/O line. The following are valid attributes:

attribute	Description
TnimcMotionIOMapDefaultInput	Maps a general purpose I/O line as the default input line.
TnimcMotionIOMapDefaultOutput	Maps a general purpose I/O line as the default output line.
TnimcMotionIOMapShutdown	Maps a general purpose input line as the shutdown (E-Stop) line. Refer to <u>Enable Shutdown</u> for more information about the shutdown (E- Stop) functionality of the controller.
TnimcMotionIOMapInhibitOut	Maps a general purpose output line as an inhibit-out line. Refer to <u>Write</u> <u>Motion I/O Data</u> for more information about the inhibit outputs.
TnimcMotionIOMapAlarmClear	Maps a general purpose output line as an alarm clear line. Refer to <u>Write Motion I/O Data</u> for more information about alarm clear.
TnimcMotionIOMapInhibitIn	Maps a general purpose input line as an inhibit-in line. Refer to <u>Write</u> <u>Motion I/O Data</u> for more information about the inhibit inputs.
TnimcMotionIOMapDriveReady	Maps a general purpose input line as a drive ready line. Refer to <u>Write</u> <u>Motion I/O Data</u> for more information about the drive ready

	signal.
TnimcMotionIOMapInPosition	Maps a general purpose input line as an in-position line. Refer to <u>Write Motion I/O Data</u> for more information about the in-position inputs.

ioAxis is the axis of the digital I/O line to map.

lineType line type to apply the selected **attribute** to. The following is the valid attribute:

attribute	Description
TnimcMotionIOMapLineTypeDigitalIO	Maps a digital I/O line.

line is line of the **ioAxis** to map.

Using This Function

This function maps a general-purpose I/O line to an alternate function of an axis such as inhibit output. For example, to map axis 3 generalpurpose output bit 1 to the inhibit output of axis 3, the function call appears as

status = nimcConfigureMotionIOMap(*boardID*, 3,

TnimcMotionIOMapInhibitOut, 3, TnimcMotionIOMapLineTypeDigitalIO, 1);



Notes

- For all attributes except for **TnimcMotionIOMapShutdown**, the **axisHandle** input must be the same as the **ioAxis** input.
- The action **TnimcMotionIOMapShutdown** applies to the device level. The axisHandle input is ignored. Furthermore, only one general-purpose input can be mapped to this attribute.
- 7330, 7340, and 7350 controllers do not support the TnimcMotionIOMapInhibitOut and **TnimcMotionIOMapShutdown** attributes because these controllers have dedicated pins for these signals.
- 7390 controllers do not support the TnimcMotionIOMapInhibitIn and **TnimcMotionIOMapInPosition** attributes because this controller has dedicated pins for these signals.

The following table lists the drive signals and their implementation on the controller:

Signal Name		Signal	Controller Support			
Controller	Drive	Direction	7330	7340	7350	7390
Inhibit-Out	Servo On	Output	Dedicated	Dedicated	Dedicated	Mappable
Inhibit-In	Servo Alarm	Input	Mappable	Mappable	Mappable	Dedicate
Drive Ready	Servo Ready	Input	Mappable	Mappable	Mappable	Mappable
In-Position	In-	Input	Mappable	Mappable	Mappable	Dedicate

	Position					
Alarm Clear	Alarm Clear	Output	Mappable	Mappable	Mappable	Mappabl

The active state of a dedicated signal is directly configurable using this function. The active state of a mappable signal is determined by the active state of the line being used.

For example, to set the active state of the Alarm Clear signal, a mappable signal on all controllers, set the active state of the line the signal is mapped to using the <u>Write Digital I/O Data</u> function.

To set the active state of the inhibit-in signal on a PCI-7390, which has a dedicated line for this signal, use the

TnimcMotionIODataInhibitInActiveState attribute of the <u>Write Motion</u> <u>I/O Data</u> function.

You cannot use this function to map signals on NI 7330/7340/7350 controllers on lines greater than line 7. For example, on an NI 7356 controller, you can use this function to configure axis 5, lines 0 through 7 (port 5), but you cannot configure lines 8 through 15, which are on the second port (port 7) of axis five. Refer to the *Remarks* section of <u>Write</u> Digital I/O Data for the port to axis and line mapping for these controllers.

nimcResetController

Device Compatibility

Reset Motion Controller

Usage

status = nimcResetController(TnimcDeviceHandle deviceHandle);

Purpose

Resets the motion controller to the power-up state.

Parameters

Name	Туре	Description
deviceHandle TnimcDeviceHandle		assigned by Measurement &
		Automation Explorer (MAX)

Using This Function

Use this function to reset the motion controller to the power-up state. After resetting the controller, you must call <u>Initialize Controller</u> or <u>Clear</u> <u>Power Up Status</u> to use the controller.

Important Information

Warranty Copyright Trademarks Patents Warning Regarding Use of NI Products

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 document 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.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

Trademarks

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on ni.com/legal for more information about <u>National Instruments trademarks</u>.

FireWire® is the registered trademark of Apple Computer, Inc.

Handle Graphics®, MATLAB®, Real-Time Workshop®, Simulink®, Stateflow®, and xPC TargetBox® are registered trademarks, and TargetBox™ and Target Language Compiler™ are trademarks of The MathWorks, Inc.

Tektronix® and Tek are registered trademarks of Tektronix, Inc.

Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help**»**Patents** in your software, the patents.txt file on your CD, or <u>ni.com/patents</u>.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS. SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE. INJURY. OR DEATH. THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES. INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR

APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- <u>Support</u>—Online technical support resources at ni.com/support include the following:
 - Self-Help Resources—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable <u>KnowledgeBase</u>, product <u>manuals</u>, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
 - Free Technical Support—All registered users receive free Basic Service, which includes access to hundreds of Applications Engineers worldwide in the <u>NI Discussion</u> <u>Forums</u> at ni.com/forums. National Instruments Applications Engineers make sure every question receives an answer.

For information about other <u>technical support options</u> in your area, visit ni.com/services or <u>contact</u> your local office at ni.com/contact.

- <u>Training and Certification</u>—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- <u>System Integration</u>—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

NI 7330 and 7340 Timing Information

The following table lists the execution time for selected functions, as well as for the different attributes of those functions, if applicable. Also included is timing information for axes or vector spaces, if both are supported with a given function.

Note The times listed are benchmark times and are not a guarantee of function execution times on your system. Times do not take into account other operations, onboard programs, or other factors that slow performance.

Function	Function Return Time (ms) using 1.47 GHz Processor ¹	Function Return Time (ms) using 700 MHz Processor [‡]
Load Target Position	1.1	1.2
Read Position	1.7	1.9
Read Motion I/O Data		
Forward Limit Active		
Reverse Limit Active	1.3	1 /
Forward Software Limit Active	1.3	
Reverse Software Limit Active	1.4	1.4
Home Input Active	1.4	1.5
Inhibit-In Active	1.6	1.5
In-Position Active	1.6	1.7
Drive Ready Active	1.6	1.7
Inhibit-Out Active	1.6	1.7
	1.6	1.7
		1.7
Read I/O Port	1.3	1.4
Set I/O Port MOMO	1.1	1.2
Load Vector Space Position	1.1	1.2
Read Velocity		

axis		
vector		
	1.8	1.8
	1.9	1.9
Read Velocity in RPM		
axis		
vector	2.0	2.1
	2.1	2.2
Load Velocity		
axis		
vector	1 2	1.2
	1.2	1.0
	1.2	1.5
Load Velocity in RPM		
axis		
vector	2.7	2.8
	2.8	2.9
Read Move Complete Status	0.1	0.2
Configure Vector Space	1.3	1.4
Check Move Complete Status	1.6	1.7
Check Blend Complete Status	1.4	1.5
Read Communication Status	0.1	0.2
Initialize Controller	607.1	607.1
Read Digital I/O Data		
Output Active State		
Input Active State	1 4	15
Output Active	1 4	1 5
Input Active	1.7	1.5

Configure As Input	1.4	1.5
	1.4	1.5
	1.4	1.5
Read Following Error	1.8	1.9

[†] An AMD Athlon 1.47 GHz PC with 256 MB of RAM was used for timing tests. Tests run in "above normal priority" execution to achieve the most accurate results.

[‡] An Intel Pentium 3 700 MHz PC with 386 MB of RAM was used for timing tests. Tests run in "above normal priority" execution to achieve the most accurate results.

NI 7334, 7342, and 7344 Timing Information

The following table lists the execution time for selected functions, as well as for the different attributes of those functions, if applicable. Also included is timing information for axes or vector spaces, if both are supported with a given function.

Note The times listed are benchmark times and are not a guarantee of function execution times on your system. Times do not take into account other operations, onboard programs, or other factors that slow performance.

Function	Function Return Time (ms) using 1.47 GHz	Function Return Time (ms) using 700 MHz
Lood Torget Desition		
	1.1	1.2
Read Position	1.9	1.9
Read I/O Port	1.6	1.7
Set I/O Port MOMO	1.1	1.2
Load Vector Space Position	1.3	1.4
Read Velocity axis vector	1.9 2.0	2.0 2.1
Read Velocity in RPM		
axis		
vector	2.1 2.1	2.2 2.2
Load Velocity		
axis		
vector	1.5	1.6

	1.2	1.3
Load Velocity in RPM		
axis		
vector	3.4	3.5
	1.9	1.9
Read Move Complete Status	0.1	0.1
Configure Vector Space	1.7	1.8
Check Move Complete Status	1.5	1.6
Check Blend Complete Status	1.6	1.7
Read Communication Status	0.1	0.1
Initialize Controller	620.4	620.4
Read Following Error	1.8	1.8

[†] An AMD Athlon 1.47 GHz PC with 256 MB of RAM was used for timing tests. Tests run in "above normal priority" execution to achieve the most accurate results.

[‡] An Intel Pentium 3 700 MHz PC with 386 MB of RAM was used for timing tests. Tests run in "above normal priority" execution to achieve the most accurate results.

NI 7350 Timing Information

The following table lists the execution time for selected functions, as well as for the different attributes of those functions, if applicable. Also included is timing information for axes or vector spaces, if both are supported with a given function.

Note The times listed are benchmark times and are not a guarantee of function execution times on your system. Times do not take into account other operations, onboard programs, or other factors that slow performance.

Function	Function Return Time (ms) using 1.47 GHz Processor ¹	Function Return Time (ms) using 700 MHz Processor [‡]
Load Target Position	1.1	1.2
Read Position	0.1	0.1
Read Axis Data Position Velocity Following Error	0.1 0.1 0.1	0.2 0.2 0.2
Read Axis Status Axis Active Move Complete Profile Complete Blend Complete Following Error Velocity Threshold Reached Moving Direction Forward	0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1	0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2

	0.1	0.2
Read All Axis Data	0.2	0.3
Read All Axis Status	0.1	0.2
Read Coordinate Data		
Velocity		
Following Error	0.2	0.3
	0.2	0.3
Pead Coordinate Position	0.3	0.3
Read Encoder Data	0.5	0.5
Position		
Index Position		
Index Capture	0.1	0.2
	1.6	1.7
	0.1	0.2
Read Motion I/O Data		
Forward Limit Active		
Reverse Limit Active	0 1	0.2
Forward Software Limit Active	0.1	0.2
Reverse Software Limit Active	0.1	0.2
Home Input Active	0.1	0.2
Inhibit-In Active	0.1	0.2
In-Position Active	1.5	1.6
Drive Ready Active	1.5	1.6
Inhibit-Out Active	1.5	1.6
	1.5	1.6
Read Capture Compare Data		
Captured Position		
Capture Status		

Compare Status	1.6	1.7
	0.1	0.2
	0.1	0.2
Read I/O Port	0.2	0.3
Set I/O Port MOMO	1.1	1.2
Load Vector Space Position	1.2	1.3
Read Velocity		
axis		
vector	0.1	0.0
	0.1	0.2
	0.2	0.3
Read Velocity in RPM		
axis		
vector	0.1	0.2
	0.1	0.2
	0.2	0.3
Load Velocity		
axis		
vector	1 2	1 /
	1 1	1.4
	<u><u></u> </u>	1.2
Load Velocity in RPM		
axis		
vector	3 1	3.2
	1.8	1.8
	1.0	1.0
Read Move Complete Status	0.1	0.1
Configure Vector Space	1.8	1.8
Check Move Complete Status	1.5	1.6
Check Blend Complete Status	1.5	1.6
Read Communication Status	0.1	0.1
---	-------------------	-------------------
Initialize Controller	1218.0	1218.0
Read Coordinate Status		
Move Complete		
Profile Complete Blend Complete Following Error Exceeded	0.2 0.2 0.2	0.3 0.3 0.3
	0.2	0.3
Read Digital I/O Data Output Active State		
Input Active State Output Active Input Active Configure As Input	1.3 1.3 0.2	1.3 1.3 0.2
	0.2	0.2 1.3
Read Following Error	2.1	2.1

[†] An AMD Athlon 1.47 GHz PC with 256 MB of RAM was used for timing tests. Tests run in "above normal priority" execution to achieve the most accurate results.

[‡] An Intel Pentium 3 700 MHz PC with 386 MB of RAM was used for timing tests. Tests run in "above normal priority" execution to achieve the most accurate results.

NI 7390 Timing Information

The following table lists the execution time for selected functions, as well as for the different attributes of those functions, if applicable. Also included is timing information for axes or vector spaces, if both are supported with a given function.

Note The times listed are benchmark times and are not a guarantee of function execution times on your system. Times do not take into account other operations, onboard programs, or other factors that slow performance.

Function	Function Return Time (ms) using 1.47 GHz Processor ¹	Function Return Time (ms) using 700 MHz Processor [‡]
Load Target Position	1.1	1.2
Read Position	0.1	0.2
Read Axis Data Position		
Velocity Following Error	0.1 0.1 0.1	0.2 0.2 0.2
Read Axis Status Axis Active Move Complete Profile Complete Blend Complete Following Error Velocity Threshold Reached Moving Direction Forward	0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1	0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2

	0.1	0.2
Read All Axis Data	0.2	0.3
Read All Axis Status	0.1	0.2
Read Coordinate Data		
Velocity		
Following Error	0.2	0.3
	0.3	0.4
Road Coordinate Desition	0.2	0.2
Read Cooluliale Position	0.2	0.3
Reau Encouer Data		
Fusition		
Index Position	0.1	0.2
index Capture	1.5	1.6
	0.1	0.2
Read Motion I/O Data		
Forward Limit Active		
Reverse Limit Active	0.1	0.2
Forward Software Limit Active	0.1	0.2
Reverse Software Limit Active	0.1	0.2
Home Input Active Inhibit-In Active	0.1	0.2
	0.1	0.2
In-Position Active	1.2	1 /
Drive Ready Active	1.0	1.4
Inhibit-Out Active	1.3	1.4
	1.3	1.4
	1.5	1.4
Read Capture Compare Data		
Captured Position		
Capture Status		

Compare Status	1.5	1.6
	0.1	0.2
	0.1	0.2
Read I/O Port	0.2	0.3
Set I/O Port MOMO	1.0	1.1
Load Vector Space Position	1.1	1.2
Read Velocity		
axis		
vector	0.1	0.2
	0.1	0.2
	0.2	0.3
Read Velocity in RPM		
axis		
vector	0.1	0.2
	0.1	0.2
	0.2	0.3
Load Velocity		
axis		
vector	1.0	1.2
	1 1	1.0
	1.1 	1.2
Load Velocity in RPM		
axis		
vector	27	27
	1 5	1.6
	1.0	1.0
Read Move Complete Status	0.1	0.1
Configure Vector Space	1.4	1.5
Check Move Complete Status	1.5	1.6
Check Blend Complete Status	1.4	1.4

Read Communication Status	0.1	0.1
Initialize Controller	565.5	608.5
Read Coordinate Status		
Move Complete		
Profile Complete	0.2	0.2
Blend Complete	0.2	0.3
Following Error Exceeded	0.2	0.3
	0.2	0.3
	0.2	0.3
Read Digital I/O Data		
Output Active State		
Input Active State	13	1 4
Output Active	1 3	1 /
Input Active	1.5	1.4
•	0.2	0.3
	0.2	0.3
Read Following Error	2.1	2.1

[†] An AMD Athlon 1.47 GHz PC with 256 MB of RAM was used for timing tests. Tests run in "above normal priority" execution to achieve the most accurate results.

[‡] An Intel Pentium 3 700 MHz PC with 386 MB of RAM was used for timing tests. Tests run in "above normal priority" execution to achieve the most accurate results.

Device Compatibility

Enable 1394 Watchdog

Note This function is no longer supported.

Legacy Information

Branch Offices

Office	Telephone Number
Australia	1800 300 800
Austria	43 662 457990-0
Belgium	32 (0) 2 757 0020
Brazil	55 11 3262 3599
Canada	800 433 3488
China	86 21 6555 7838
Czech Republic	420 224 235 774
Denmark	45 45 76 26 00
Finland	385 (0) 9 725 72511
France	33 (0) 1 48 14 24 24
Germany	49 89 7413130
India	91 80 41190000
Israel	972 0 3 6393737
Italy	39 02 413091
Japan	81 3 5472 2970
Korea	82 02 3451 3400
Lebanon	961 (0) 1 33 28 28
Malaysia	1800 887710
Mexico	01 800 010 0793
Netherlands	31 (0) 348 433 466
New Zealand	0800 553 322
Norway	47 (0) 66 90 76 60
Poland	48 22 3390150
Portugal	351 210 311 210
Russia	7 495 783 6851
Singapore	1800 226 5886
Slovenia	386 3 425 42 00

South Africa	27 0 11 805 8197
Spain	34 91 640 0085
Sweden	46 (0) 8 587 895 00
Switzerland	41 56 2005151
Taiwan	886 02 2377 2222
Thailand	662 278 6777
Turkey	90 212 279 3031
United Kingdom	44 (0) 1635 523545
United States (Corporate)	512 683 0100