VSI Motion Control API
User Guide

Document Revision 1.3

(Updated Oct. 21, 2015)

Phoenix, AZ USA

For more information please visit the product web page:

www.vitalsystem.com/motion/sdk

Contents

License Agreement

Before using this software, please take a moment to go thru this License agreement. Any use of this software indicate your acceptance to this agreement.

It is the nature of all machine tools that they are dangerous devices. In order to be permitted to use this software on any machine you must agree to the following license:

I agree that no-one other than the owner of this machine, will, under any circumstances be responsible, for the operation, safety, and use of this machine. I agree there is no situation under which I would consider Vital Systems, or any of its distributors to be responsible for any losses, damages, or other misfortunes suffered through the use of this software. I understand that this software is very complex, and though the engineers make every effort to achieve a bug free environment, that I will hold no-one other than myself responsible for mistakes, errors, material loss, personal damages, secondary damages, faults or errors of any kind, caused by any circumstance, any bugs, or any undesired response by the board and its software while running my machine or device.

I fully accept all responsibility for the operation of this machine while under the control of this software, and for its operation by others who may use the machine. It is my responsibility to warn any others who may operate any device under the control of DSPMC board of the limitations so imposed.

I fully accept the above statements, and I will comply at all times with standard operating procedures and safety requirements pertinent to my area or country, and will endeavor to ensure the safety of all operators, as well as anyone near or in the area of my machine.

WARNING: Machines in motion can be extremely dangerous! It is the responsibility of the user to design effective error handling and safety protection as part of the system. VITAL Systems shall not be liable or responsible for any incidental or consequential damages. By Using the DSPMCv2 motion controller, you agree to the license agreement.

Introduction

The VSI Motion Control API is a set of native C/C++ libraries designed as a simple, yet comprehensive, motion control interface for Vital System Inc. Ethernet Motion Controllers that can be imported into custom applications.

For windows platforms, the libraries come in the standard .dll file format for Windows.

Available VSI Motion Control API libraries:

- HiCONMotionControlAPI.dll for HiCON OEM (pn7752) and HiCON Integra (pn7766)
- DSPMCMotionControlAPI.dll for DSPMC (pn7762)

Features:

- **Compact Native Libraries**
  The VSI Motion Control API libraries are made in native C/C++ which allow them to be integrated into any kind of application platform (C/C++ or .NET).

- **Cross-platform support**
  The VSI Motion Control API libraries are designed for minimal porting to provide compatibility across multiple platforms.

- **Simplified program flow for ease of integration**
  The hardware abstraction layer handles most “lower-level” operations, such as network communication and motion control, leaving the user with invoking only the necessary control functions.

- **Verbose Debugging Console**
  The VSI Motion Control API has built-in functionality to open a console window which prints all messages and notifications in “layman” terms.

- **High-Speed UDP network communication**
  Communication with VSI Motion Controllers is handled via Ethernet through the use of UDP transmission. This ensures high-speed, low-latency, and lossless data integrity.

- **Customizable Data Exchange Rate**
  Command Data and Status data can be exchanged at a minimum rate of 5ms.

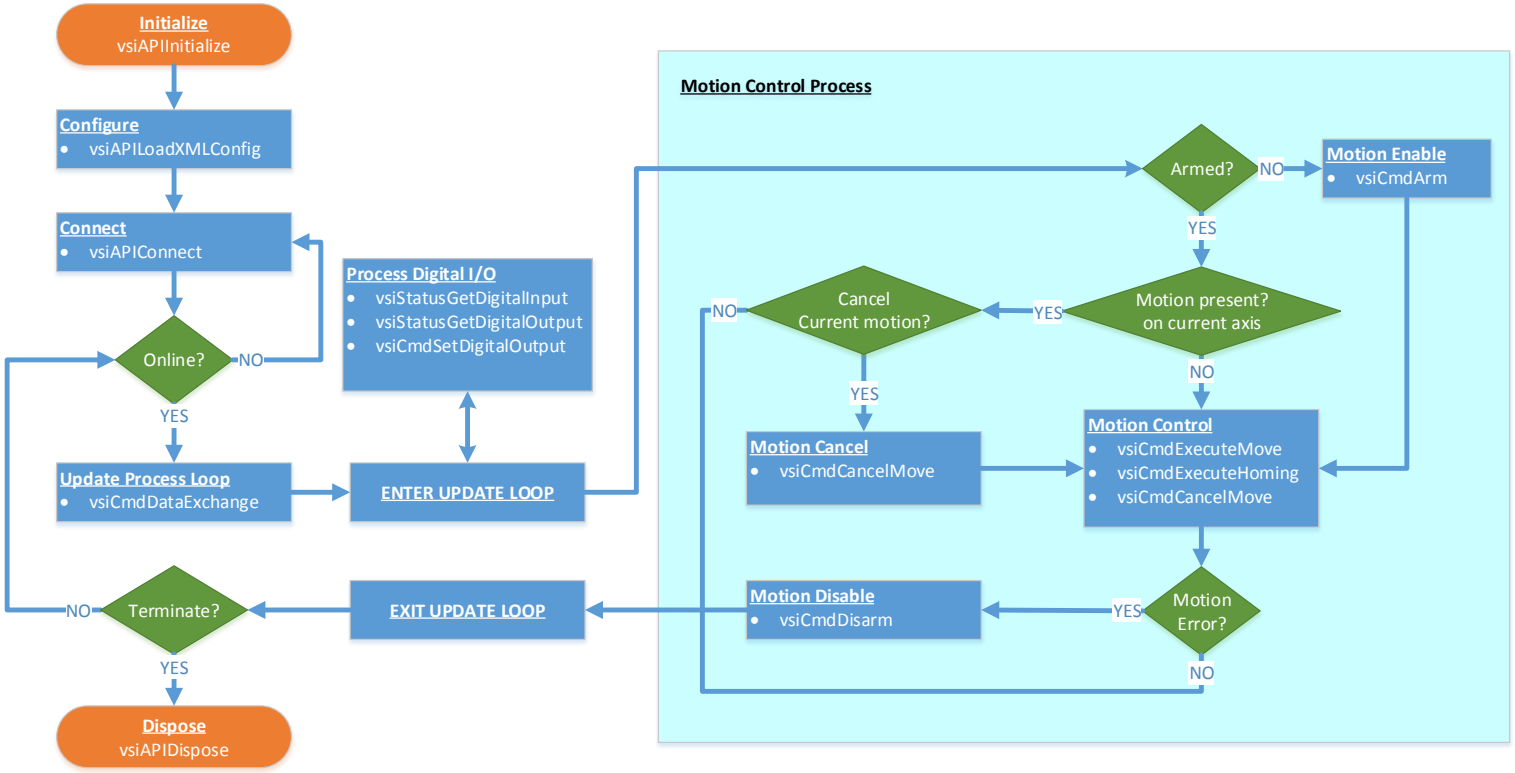
- **Single-Axis or Coordinated Linear Multi-Axis Motion Control**
  Single-Axis Control allows the user to manipulate each axis one by one. Each axis is controlled independently from the others allowing the user to manually synchronize motion. Coordinated

Linear Multi-Axis control allows the user to move multiple axes at once in a coordinated trajectory with a specified trajectory velocity and acceleration. This is similar to G0 or G1 in GCode.

- **Multiple motion modes: Relative, Absolute, Velocity**
  Relative motion moves an axis to a specified target position in relation to the current position. Absolute motion moves an axis to an absolute target position. Velocity motion moves an axis indefinitely in the given direction.

Program Flowchart

The VSI Motion Control API requires a sequence of functions to be called in the appropriate order to operate properly. Below is a flowchart with a graphical description of the ideal VSI Motion Control API Behavior.



Definition of Terms

The VSIMotionAPI is built in native C/C++ and as such, can be imported into several programming languages. Because of the varied data-type specifications across multiple languages (in terms of size and functionality), it is necessary to consult the specification for the desired programming language when importing the API in order to avoid runtime errors such as displaced/misaligned data.

***NOTE**: It is highly recommended to view the provided samples and their source code available on the VSI Motion API webpage.*

Provided below are the data-types that are used in the API functions:

Type	**Description**
int32	32-bit signed integer.
uint32	32-bit unsigned integer.
float	32-bit single precision floating point value
double	64-bit double precision floating point value
char*	Array of single-byte characters (ANSI). Also known as “string” in other languages, but not to be confused with “std::string” in the C++ standard libraries.
bool	Single-byte Boolean value assigned with “false” (0), or “true” (non-zero value).

***NOTE**: Data types followed by an asterisk (*) are reference types (also known as pointer types in C++). This usage must be specified accordingly when interfacing with the API functions (e.g. “ref” in C#, or “ByRef “in BASIC).*

VSI API Functions

These functions are directly tied to control of the whole API.

int32 vsiAPIOpenConsole()

Open a debug Console. Console is tied to application. All notifications are printed on the Console

int32 vsiAPIInitialize()

Initialize the Motion API. ALWAYS CALL THIS FIRST

int32 vsiAPIDispose()

Dispose the Motion API. ALWAYS CALL THIS LAST

int32 vsiAPIConnect(char serial, int32 statusPollIntervalMS, int32 timeoutMS)*

Find and connect to any Controller on the network.

Parameters:

- serial – Serial number of a particular controller to connect to. This value is passed as a string (or a standard character array).
- statusPollIntervalMS – Interval in milliseconds at which data is exchanged with the controller
- timeoutMS – If no message is received from the controller for this time period in milliseconds, then the controller is flagged as offline

int32 vsiAPIDisconnect()

Disconnect from the current controller. Does nothing if controller is offline.

int32 vsiAPIGetVersion(char version)*

Returns the current version of the Motion Control API.

Parameters:

- version – Return string. **Allocate at least 10 characters to for this character array.**

int32 vsiAPIGetLastNotification(char error, int32* length)*

Returns the Last Notification made by the Motion Control API. This is usually called right after the HiCON is disarmed to determine the disarming error.

Parameters:

- error – Return string. **Allocate a minimum of 200 characters for this character array**.
- Length – length of the return string.

int32 vsiAPILoadXMLConfig(char filePath)*

Load the controller config from an XML File. This needs to be called only when the current config needs to be updated with new data. Config is always stored after loading.

NOTES:

- This function will not affect the motion controller until "vsiAPIDownloadConfig" is called.

Parameters:

- filePath – Full name (directory, filename, and extension) of the file to be loaded.

int32 vsiAPIDownloadConfig()

Download the current config to the controller. If the XML config has been changed during disarmed state, this function must be called to apply those changes to the controller.

NOTES:

- This function is also called when the API attempts to connect to a controller (failure to download terminates the connection).
- This function cannot be called while the controller is offline, armed, or if there is no XML File loaded.

VSI Command Functions

These functions are used to direct the behavior of the current motion controller. All of these functions will do nothing if there is no controller connected (or flagged as offline).

System commands

int32 vsiCmdArm(unsigned int32 axisSelection)

Commands the controller to arm and enable motion output.

Parameters:

- axisSelection – Bit mask used to determine which axes need to be armed. bit0=X, bit1=Y, bit2=Z and so on. Example: To arm X, Y, Z and B axes, a value of 00010111 (23 in decimal) is required.

int32 vsiCmdDisarm()

Disarms the controller to disarm and disable all motion.

int32 vsiCmdDataExchange()

Sends all outgoing data to the motion controller and receives all input data. This function must be called in the application update loop.

int32 vsiCmdClearAxisPosition(int32 axis)

Sets the current axis position to zero. Cannot be called while controller is armed.

Parameters:

- axis – The axis to clear position.

int32 vsiCmdSetAxisPosition(int32 axis, float units)

Set the current axis position to the specified value.

Parameters:

- axis – The target axis.
- units – The position (in units) to assign to the current axis.

int32 vsiCmdClearEncoderCounts(int32 channel)

Sets the current encoder counts to zero.

Parameters:

- channel – The encoder channel to clear counts.

int32 vsiCmdSetEncoderCounts(int32 encoder, int32 counts)

Sets the current encoder counts to the specified value.

Parameters:

- channel – The target encoder channel.
- counts – The number of counts to assign to the encoder channel.

int32 vsiCmdSetDACOutput(int32 channel, double volts)

Set a specified amount of volts to be outputted on a DAC channel. These volts operate outside the PID loop, so the channel in question must not be currently mapped to an enabled axis. This function is chiefly used for spindle motion.

Parameters:

- channel – The DAC channel to output volts.
- volts – Amount of volts to be outputted on the current channel. Valid range is -10V through +10V.

int32 vsiCmdSetDigitalOutput(int32 port, int32 pinNumber, bool value)

Set the current active state of a Digital Output. DSPMC uses raw pin notation.

Parameters:

- port – Output port of the controller. HiCON = P11 – P14; DSPMC = 0 (raw pin notation)
- pinNumber – Output pin number. HiCON = 0 – 7; DSPMC = 0 – 31 (raw pin notation)
- value – Active state to set the digital output.

Motion Commands

NOTE: *Before motion commands can be performed, the controller must first be properly configured via the XML config file. See "vsiAPILoadXMLConfig", and "vsiAPIDownloadconfig" above.*

The controller (and all axes that will receive motion commands) ***must be armed****, as a motion command issued on a disabled axis will throw a motion error.*

Motion functions are **NON-BLOCKING** which means that a function call for any of the motion functions will immediately return and not wait for the actual motion to finish. As such, the program must wait for motion to complete (in some cases) so that the axis may transition to the "Idle" state before issuing new motion commands.

The presence of active motion can be verified by calling "vsiStatusIsMoving". Active motion can also be stopped by calling "vsiCmdCancelMove".

Linear Motion Commands

- VSI_MOVE_RELATIVE – Target position is made in relation to current position. (value: 2)
- VSI_MOVE_ABSOLUTE – Target position is the absolute position. (value: 4)
- VSI_MOVE_VELOCITY – Motion moves infinitely in one direction. Target position's sign (+/-) is used to determine direction. (value: 8)

int32 vsiCmdExecuteMove(int32 axis, double position, double speed, double accel, int32 moveFlags, unsigned int32 seqID)

Execute a point-to-point move on a specified axis.

Parameters:

- axis – Axis to perform the move.
- position – Target/Destination position of the move.
- speed – Maximum Speed to move at.
- accel – Acceleration to the max speed.
- moveFlags – Flags used to determine the type of motion being performed .
- seqID – User-defined sequence ID used to track which move is currently being performed by the motion controller.

If active motion from a prior vsiCmdExecuteMove call is already present on the axis, calling this function will "continue on" from the current motion.

For example, if the axis is currently moving at a speed of 100 units/min, and a new motion command was issued with a speed of 200units/min, then the axis will ramp up from 100 to 200 units/min using the new acceleration and will stop at the new final position.

However, a motion error is thrown if the target direction of the new motion is in the opposite direction of the current motion. In this case, it is necessary to wait for the current motion to finish before issuing the new motion command.

int32 vsiCmdExecuteLinearMove(int32 axisMap, double position[], double speed, double accel, int32 moveFlags)

Execute a point-to-point move on a specified axis.

Parameters:

- axisMap – bitmask used to determine which axes will execute the motion (e.g. bit0=X, bit1=Y, bit2=Z, etc).
  Example: To move X, Y, Z, and B, the axisMap value would be "23" (binary: 00010111).
- positions – an array of doubles (double[8] for DSPMC, double[6] for HiCONs) which contains the target positions for the designated axes.
- speed – Maximum trajectory speed to move at.
- accel – Trajectory acceleration to the max speed.
- moveFlags – Flags used to determine the type of motion being performed .

Arc/Circular Motion Commands

Arc Motion Errors

Arc Motion commands will return the following error codes if the motion fails to execute. A return code of "0" denotes that the motion has successfully started.

Error	**Description**
10001	Axis not idle error. This error is returned if one of the axes involved with the Arc Motion was not idle/still when the Arc Motion command was started.
10500	Inconsistent radius error. This error can be returned if the "*start-to-center*" distance is not equal to the "target-to-center" distance.
10501	360° angle impossible error. This error is returned by the "DoArcMoveRadius", and "DoArcMoveAngle" functions since they cannot mathematically interpolate the center point of a 360° *arc.*

int32 vsiCmdExecuteArcMoveCenter (int32 axisOfRotation, double axisPositions[3], double offset1, double offset2, double speed, bool clockwise)

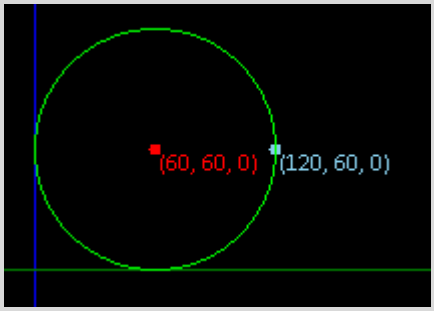
Execute an Arc/Circular trajectory move around an axis of rotation by specifying a center point relative to the initial position.

Parameters:

- *axisOfRotation* – The axis where the arc/circle trajectory will rotate on.
  - A value of **0** means that the arc will be plotted on the **YZ** plane.
  - A value of **1** means that the arc will be plotted on the **XZ** plane.
  - A value of **2** means that the arc will be plotted on the **XY** plane.
- *axisPositions* – an array of 3 doubles (64-bit floating point precision) which contains the target/end positions for the arc trajectory.
- *offset1* – Offset from the current axis position to use for the center point. Usage is determined by the "axisOfRotation" parameter.
  - If "axisOfRotation" is **0**, then this value is used as the **Y offset**.
  - If "axisOfRotation" is **1**, then this value is used as the **X offset**.
  - If "axisOfRotation" is **2**, then this value is used as the **X offset**.
- *offset2* – Offset from the current axis position to use for the center point. Usage is determined by the "axisOfRotation" parameter.
  - If "axisOfRotation" is **0**, then this value is used as the **Z offset**.
  - If "axisOfRotation" is **1**, then this value is used as the **Z offset**.
  - If "axisOfRotation" is **2**, then this value is used as the **Y offset**.
- *speed* – max arc trajectory speed/feedrate.
- *clockwise* – specifies the direction of the arc motion. (1 = clockwise, 0 = counter-clockwise).

Example: This code snippet will produce the following trajectory. *Start Position = {120, 60, 0}.*

```
float targetPositions[6];
targetPositions[0] = 120;
targetPositions[1] = 60;
targetPositions[2] = 0;
DoArcMoveCenter(2, targetPositions, -60, 0, 120, 1);
```



int32 vsiCmdExecuteArcMoveRadius (int32 axisOfRotation, double axisPositions[3], double radius, double speed, bool clockwise)

Execute an Arc/Circular trajectory move around an axis of rotation by specifying the radius of the arc/circle trajectory in order to interpolate the arc center.

NOTE: *This function has the following mathematical limitations:*

- *The start point cannot be equal to the end point (360° arc center point cannot be interpolated).*
- *Cannot generate arcs with angles greater than 180°.*

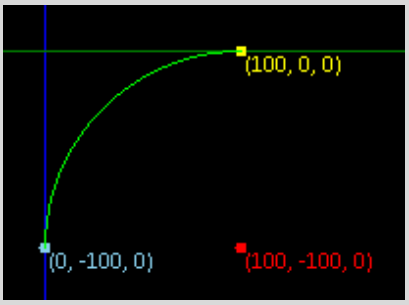
For arcs larger than 180°, use "DoArcMoveCenter", or call this function twice.

Parameters:

- *axisOfRotation* – The axis where the arc/circle trajectory will rotate on.
  - A value of **0** means that the arc will be plotted on the **YZ** plane.
  - A value of **1** means that the arc will be plotted on the **XZ** plane.
  - A value of **2** means that the arc will be plotted on the **XY** plane.
- *axisPositions* – an array of 3 doubles (64-bit floating point precision) which contains the target/end positions for the arc trajectory.
- *radius* – The radius of the specified arc/circle trajectory.
- *speed* – max arc trajectory speed/feedrate.
- *clockwise* – specifies the direction of the arc motion. (1 = clockwise, 0 = counter-clockwise).

Example: This code snippet will produce the following trajectory. *Start Position = {100, 0, 0}.*

```
float targetPositions[6];
targetPositions[0] = 0;
targetPositions[1] = -100;
targetPositions[2] = 0;
DoArcMoveRadius(2, targetPositions, 100, 120, 0);
```



int32 vsiCmdExecuteArcMoveAngle (int32 axisOfRotation, double axisPositions[3], double arcAngle, double speed, bool clockwise)

Execute an Arc/Circular trajectory move around an axis of rotation by specifying the angle of the arc/circle trajectory in order to interpolate the arc center.

NOTE: *This function has the following mathematical limitations:*

- *The start point cannot be equal to the end point (360° arc center point cannot be interpolated).*

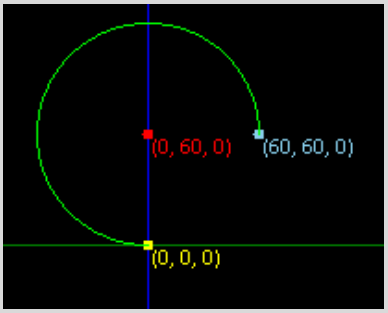
For larger arcs such as a full 360° circle, use "DoArcMoveCenter", or call this function twice with both instances generating a 180° arc.

Parameters:

- *axisOfRotation* – The axis where the arc/circle trajectory will rotate on.
  - A value of **0** means that the arc will be plotted on the **YZ** plane.
  - A value of **1** means that the arc will be plotted on the **XZ** plane.
  - A value of **2** means that the arc will be plotted on the **XY** plane.
- *axisPositions* – an array of 3 doubles (64-bit floating point precision) which contains the target/end positions for the arc trajectory.
- *arcAngle* – The angle of the arc trajectory to generate.
- *speed* – max arc trajectory speed/feedrate.
- *clockwise* – specifies the direction of the arc motion. (1 = clockwise, 0 = counter-clockwise).

Example: This code snippet will produce the following trajectory. *Start Position = {0, 0, 0}.*

```
float targetPositions[6];
targetPositions[0] = 60;
targetPositions[1] = 60;
targetPositions[2] = 0;
DoArcMoveRadius(2, targetPositions, 270, 120, 1);
```



Other Motion Commands

int32 vsiCmdExecuteHoming(int32 axis, double homeOffset, double speed, double accel)

Execute a homing sequence on the specified axis.

Parameters:

- axis – Axis to perform the move.
- homeOffset – The position assigned to the axis at the end of the homing sequence.
- speed – Maximum Speed to move at.
- accel – Acceleration to the max speed.

int32 vsiCmdCancelMove(int32 axis, bool instantStop)

Cancels the current move on the specified axis.

Parameters:

- axis – Axis in motion. A value of **-1** can also be specified which cancels motion on all axes (and motion sequences that work on multiple axes such as arc/linear motion).
- instantStop – Set to true to have the axis abruptly stop. Set to false to have the axis decelerate to a stop.

int32 vsiCmdLoadBufferedMotion(double bufferedMotion[][VSI_MAX_AXIS], int32 motionPointCount)

Loads a set of buffered motion vectors (1 vector per millisec resolution) to the API. The buffered motion is downloaded to the controller on the next call of *vsiCmdDataExchange().*

NOTES:

- Only the allowed maximum points from the motion buffer is downloaded when *vsiCmdDataExchange()* is called. Calling the data exchange gradually consumes the motion buffer as necessary until it is fully consumed. The maximum motion buffer size can be specified in the config XML *(see <MaxMotionBufferPoints>)*.

Parameters:

- bufferedMotion – A 2D array containing several motion vectors. A motion vector is an array of axis points. 1st dimension is the vector index, 2nd dimension is the axis.
- motionPointCount – The number of motion vectors in the *bufferedMotion* array.

VSI Status Functions

These functions are used to read information about the current controller.

int32 vsiStatusIsOnline(bool value)*

Reads the current online state of the controller.

Parameters:

- value – Returns the current online state.

int32 vsiStatusIsArmed(bool value)*

Reads the current armed state of the controller.

Parameters:

- value – Returns the current armed state.

int32 vsiStatusIsMoving(int32 axis, bool value)*

Reads motion on a specified axis.
Parameters:

- axis – Axis to check for motion. If a value of **-1** is specified, the function returns "**false**" if no axes are moving, or "**true**" if at least one axis is in motion.
- value – Returns the true if the axis is currently in motion.

int32 vsiStatusIsMotionDone(int32 axis, unsigned int32 seqID, bool value)*

Checks if the motion flagged with the specified sequence ID is done.

Parameters:

- axis – Axis to check for motion.
- seqID – Sequence ID to check for.
- value – Returns the true if the motion flagged with the seqID is done.

int32 vsiStatusIsHomeFound(int32 axis, bool value)*

Checks if the homing sequence has finished for the specified axis.

Parameters:

- axis – Axis to check for motion.

- value – Returns the true if the axis homing sequence is finished.

int32 vsiStatusGetSerial(char serial)*

Reads the serial number of the current online motion controller

Parameters:

- serial – Return string containing the serial number. Allocate at least 10 characters.

int32 vsiStatusGetFollowError(int32 axis, double value)*

Reads the current following error of an axis. The following error is the difference of the the commanded position and the actual. This value is mostly useful for debugging how much of a following error was incurred during a disarm.

Parameters:

- axis – Axis to check for following error.
- value – Return value containing the following error value.

int32 vsiStatusGetFollowErrorBits(unsigned int32 errorBits)*

Reads which axes have experienced a following error limit. This is useful for determining which axes were responsible for a "following error" disarm.

Parameters:

- errorBits – Return value containing a bitmask which indicates the axes that incurred a following error.

int32 vsiStatusGetMotionSequenceID(int32 axis, unsigned int32 seqID)*

Reads the current motion seqID on a specified axis that is being executed by the controller.

Parameters:

- axis – Axis to check for motion.
- seqID – Return value containing the current seqID.

int32 vsiStatusGetAxisPosition(int32 axis, double position)*

Reads the current position of an axis. The axis position depends on the type of feedback selected. If the axis uses encoder feedback, the position from encoder counts is shown. If stepper output with no feedback is selected, position from the outputted steps is shown.

Parameters:

- axis – Axis to check for position.
- position – Return value containing current actual position of an axis.

int32 vsiStatusGetAxisPositions(double axisPositions)*

Reads the current position of all axes and returns an array of doubles

int32 vsiStatusGetEncoderCounts(int32 channel, int32 value)*

Reads the current encoder counts from a specified channel.

Parameters:

- channel – Encoder channel to check.
- value – Return value the current encoder counts.

int32 vsiStatusGetDigitalInput(int32 port, int32 pinNumber, bool value)*

Read the active state of a specified digital input.

Parameters:

- port – Input port of the controller. HiCON = P11 – P14; DSPMC = 0 (raw pin notation)
- pinNumber – Input pin number. HiCON = 0 – 15; DSPMC = 0 – 63 (raw pin notation)
- value – Return value containing the current active state of the digital input.

int32 vsiStatusGetDigitalOutput(int32 port, int32 pinNumber, bool value)*

Read the active state of a specified digital output.

Parameters:

- port – Output port of the controller. HiCON = P11 – P14; DSPMC = 0 (raw pin notation)
- pinNumber – Output pin number. HiCON = 0 – 7; DSPMC = 0 – 31 (raw pin notation)
- value – Return value containing the current active state of the digital output.