



VSI C-Macro User Guide

Document Revision 1.5

(Updated Sep. 26, 2023)

© 2020 Vital Systems Inc

Buford, GA USA

www.vitalsystem.com



Extremely Important Reminder

When operating machines, take extreme precautions. The machines can have enormous power even with a small motor. Never come inside a machine path while powered. Operating machines without necessary precautions can result in severe injury or even death.



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 any Vital System Inc. motion controller and accompanying software, you agree to the license agreement.

Contents

License Agreement	3
I. Macro Basics	4
1. What is a VSI C-Macro?	4
2. VSI Macro Loader	4
How to run a VSI C-Macro:	5
3. Running your VSI C-Macro from within Mach3 and Mach4	5
4. Mach DROs and LEDs	5
5. Program Format	6
6. Additional Information and Supported Libraries	6
II. VSI C-Macro Functions	7
1. Misc Functions	7
Print(const char *format, ...)	7
Sleep(long milliseconds)	7
2. Input/Output Functions	8
long GetPin(long port, long pin)	8
SetPin (int port, int pin, int state)	8
long GetLED(long index)	8
SetLED(long index, long state)	9
float GetDRO(long index)	9
SetDRO(long index, float value)	9
long GetControl (int index)	9
float GetControlFloat (int index)	11
SetControl (int index, int value)	11
3. Motion Functions	13
long DoMotionAxis(int axis, float finalPosition, float speed, float accel, long mode)	14
long DoMotionXYZ(float posX, float posY, float posZ, float speed, float accel, long mode)	15
long DoLinearMove (long axisMap, float* axisPositions, float speed, float accel, long mode)	15
long DoHoming(long axis, float homePosition, float speed, float accel, long homingParams)	16
CancelMove(long axis, long instantStop)	17
4. Arc/Circle Motion Functions	18
Arc Motion Errors	18
long DoArcMoveCenter (long axisOfRotation, float* axisPositions, float offset1, float offset2, float speed, bool clockwise)	19
long DoArcMoveRadius (long axisOfRotation, float* axisPositions, float radius, float speed, bool clockwise)	20
long DoArcMoveAngle (long axisOfRotation, float* axisPositions, float arcAngle, float speed, bool clockwise)	21
III. Debug Registers	22

License Agreement

Before using any Vital System Inc. (VSI) hardware and/or software, please take a moment to go thru this License agreement. Any use of this hardware and 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 VSI hardware and/or 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 Inc., or any of its distributors to be responsible for any losses, damages, or other misfortunes suffered through the use of VSI hardware and/or software. I understand that the VSI hardware and/or 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 VSI hardware and/or software while running my machine or device.

I fully accept all responsibility for the operation of this machine while under the control of VSI hardware and/or 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 VSI hardware and/or software 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.

I. Macro Basics

1. What is a VSI C-Macro?

VSI C-Macros are programs written in C programming language (.c file) which can be downloaded and executed directly inside any Vital System Inc. Motion Controller. The program allows the user to manipulate high speed I/O and launch motion commands, as well as, communicate with PC programs, such as, Mach3 and Mach4 over Ethernet.

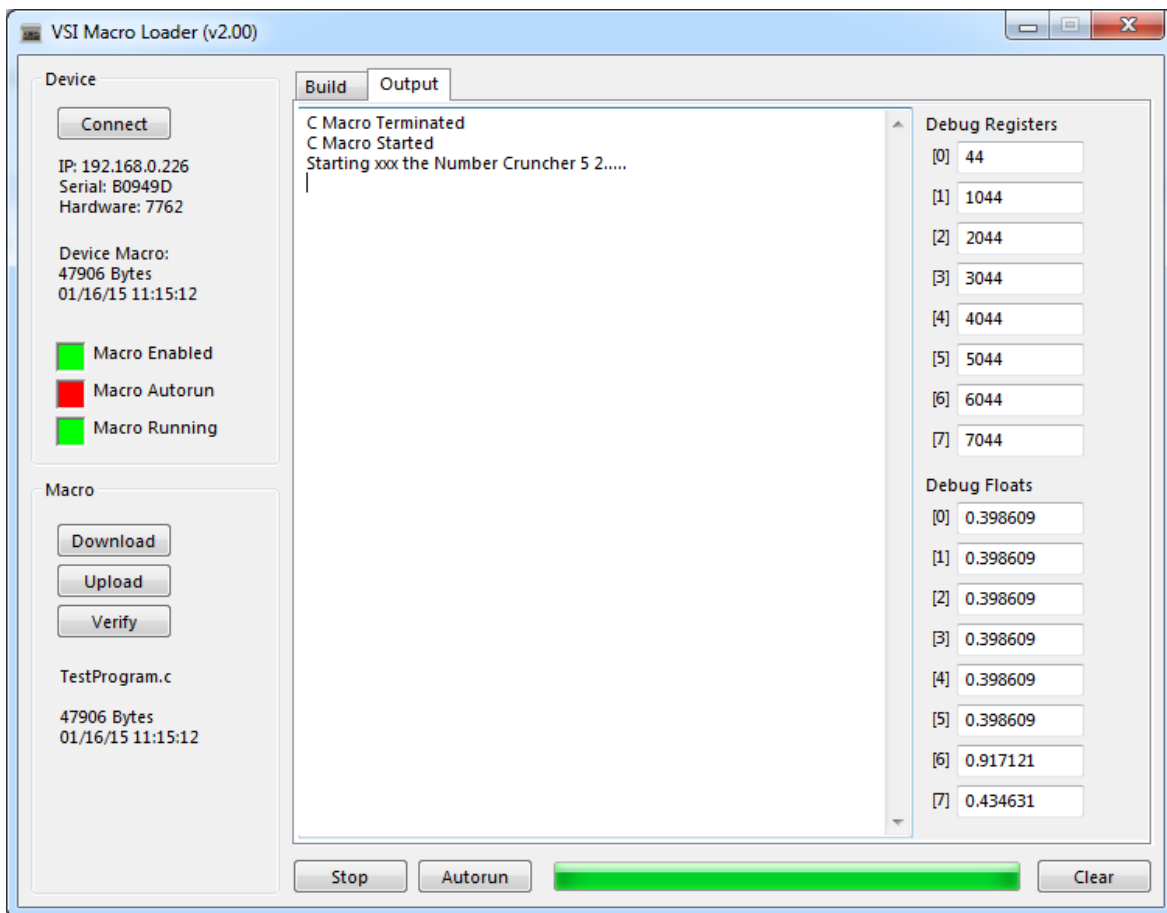
Advantages:

- Real-time processing
- No Network Latency
- Instant I/O state changes
- Standalone operation capable (no PC intervention)
- Allows custom features that are not supported by CNC Software such as Mach.

2. VSI Macro Loader

VSI Macro Loader is an application that is used to install and debug Macro programs (written in BASIC or C language) for VSI Motion Controllers.

Download link: <http://www.vitalsystem.com/portal/motion/VSIMacroLoader.zip>



How to run a VSI C-Macro:

NOTE: *If controller is already configured for standalone operation, Step1 can be skipped.*

1. Open Mach3 or Mach4 and follow the setup as specified in the Mach3 or Mach4 Software Integration manual (if not yet done). This is where all configuration parameters are taken from.
2. Run the VSI Macro Loader application.
3. Click the “Connect” button to open the connection dialog window.
4. If your VSI Motion Controller is on the same network as your PC, it will be listed as an entry in the dialog window. Click on it, then click OK.
5. If a connection was successfully established, click the “Download” button and select which Macro (.bas or .c file) you would like to download. This will also compile the program.
6. If your C-Macro has any compilation errors, they will be listed in the “Build” window.
7. If the download was successful, click the “Run/ Stop” button to run the Macro program.
8. Use the output window to get feedback and check for errors.

NOTE: *Macro programs may also be allowed to execute automatically on power-up by clicking the “Autorun” button, however this is only recommended when the Macro Program is fully debugged and error-free.*

3. Running your VSI C-Macro from within Mach3 and Mach4

The downloaded VSI C-Macro can be started from within Mach3 or Mach4.

- Mach3 – Turn on LED 2035.
- Mach4 – Set the “RunDeviceMacro” Register to any non-zero value (such as “1”).

4. Mach DROs and LEDs

There are certain ranges of Mach DROs (32-bit Floating Point values) and LEDs (Binary 0 or 1) that are shared between Mach3/Mach4 and the Macro Program.

NOTE: *In Mach4, LEDs and DROs are accessed as registers “VLED_xxxx” and “VDRO_xxxx”.*

Type	Range	Description	C-Macro Function
LED	2000...2031	LEDs written by Mach. Read-only inside Macro Program	GetLED
LED	2050...2081	LEDs written by Macro. Read-only inside Mach	GetLED, SetLED
DRO	2000...2049	DROs written by Mach. Read-only inside Macro	GetDRO
DRO	2050...2099	DROs written by Macro. Read-only inside Mach	GetDRO, SetDRO

5. Program Format

The C-Macro program must ALWAYS follow the format below. It also serves as the basic template.

```
#include "cmacro.h" // (REQUIRED) This is the C-Macro library
#include <math.h> // (optional)
/*include other libraries here*/

void CMacro() // (REQUIRED) this is the C-Macro Main Function and the entry point of your program
{
    // Start of C Macro
    while(1) // main loop
    {
        SetPin(11, 1, 0); // turn off output on Port 11 Pin 1
        Sleep(250); // delay ¼ second

        SetPin(11, 1, 1); // turn on output on Port 11 Pin 1
        Sleep(250); // delay ¼ second

        SetDRO(2050, sin(pi / 2)); //set DRO 2050 to sin 90 degrees
    }
    // End of C Macro. The end of this function signals the termination of the macro
}
```

6. Additional Information and Supported Libraries

The VSI C-Macro feature supports the following standard C libraries:

- <math.h>
- <string.h>
- <stdlib.h>

NOTE: For information on the standard C libraries, as well as general tutorials about writing C programs, please refer to these useful links:

<http://www.cprogramming.com/tutorial/c-tutorial.html>

<http://www.cplusplus.com/reference/library/>

II. VSI C-Macro Functions

NOTE: When writing VSI C-Macros, it is recommended to use an IDE (such as [Notepad++](#)) as opposed to a plain text editor for ease of use and colored syntax.

1. Misc Functions

Print(const char *format, ...)

Prints a formatted string. Print output is shown on the VSI Macro loader Output Window.

NOTE: This function uses the same format and parameters as the standard "Printf()" function in C. As such, it can use the same string formats that "Printf()" supports.

Example: The following code snippet prints some standard formatted messages.

```
Print("This is my debug message\n");           // displays "This is my debug message"
long number = 7762;
Print("My number is %d\n", number);           // displays "My number is 7762"
float value = (2 / 3);
Print("My value is %g\n", value);             // displays "My value is 0.66666667..."
```

Sleep(long milliseconds)

Pauses the Macro program execution for the specified amount of milliseconds.

Parameters:

- *milliseconds* – amount of time to sleep in milliseconds.

2. Input/Output Functions

long GetPin(long port, long pin)

Returns current state of a digital input.

Parameters:

- port – the logical port where the digital input pin is located (*valid range [11 – 14]*).
- pin – the digital input pin (*valid range [0 – 15]*).

Port	DSPMC (7763)	Integra (7866)	Integra (7766)
11	J1 Digital Inputs [0 – 15]	J13 and J14 Digital Inputs	J13 and J14 Digital Inputs
12	J2 Digital Inputs [16 – 31]	J7 Digital Inputs	J7 Digital Inputs
13	J3 Digital Inputs [32 – 47]	J8 Digital Inputs	J8 Digital Inputs
14	J4 Digital Inputs [48 – 63]	J10 Digital Inputs	J10 Digital Inputs

SetPin (int port, int pin, int state)

Sets the active state of a digital output.

Parameters:

- port – the logical port where the digital input pin is located (*valid range [11 – 14]*).
- pin – the digital input pin (*valid range [0 – 8]*).
- state – the new active state of the digital output (*0 or 1*).

Port	DSPMC (7763)	Integra (7866)	Integra (7766)
11	J1 Digital Outputs [0 – 7]	J15 Digital Outputs	J15 Digital Outputs
12	J2 Digital Outputs [8 – 15]	J7 Digital Outputs	J7 Digital Outputs
13	J3 Digital Outputs [16 – 23]	J8 Digital Outputs	J8 Digital Outputs
14	J4 Digital Outputs [24 – 31]	-	-

long GetLED(long index)

Read the specified LED state.

Parameters:

- index – the LED index. Index range [2000 – 2031] are LEDs written by Mach. Index range [2050 – 2081] are written by the motion controller via *SetLED*.

SetLED(long index, long state)

Write a bit value (0 or 1) to the specified LED.

Parameters:

- *index* – the LED index (*valid range [2050 – 2081]*).
- *state* – the new active state of the LED (*0 or 1*).

float GetDRO(long index)

Read the specified DRO value.

Parameters:

- *index* – the DRO index. Index range [2000 – 2049] are DROs written by Mach. Index range [2050 – 2099] are written by the motion controller via *SetDRO*.
- *state* – the new active state of the LED (*0 or 1*).

SetDRO(long index, float value)

Write a floating point value to the specified DRO.

Parameters:

- *index* – the DRO index (*valid range [2050 – 2099]*).
- *state* – the new active state of the LED (*0 or 1*).

long GetControl (int index)

Read internal data of the VSI Motion Controller specified by the “index” parameter. C index definitions are also available for ease of use.

Parameters:

- *index* – indicator for specific control data field (*see table below*).

Example: The following code snippet reads the feedback position of each axis by using a loop:

```
int axis;  
int axisPositions[8];  
for( axis = 0; axis < 8; axis ++)  
    axisPositions[axis] = GetControl( GET_CTRL_AXIS_FEEDBACK_POS + axis );
```

HiCON Control Parameters

Index	C #define	Description
0 – 31	GET_CTRL_DIGI_INPUTS	Digital Input States (0 – 31) located on P11 and P12
35	GET_CTRL_DRIVE_ENABLE	PID Active (or Drive Enabled) State
36	GET_CTRL_FIFO_LEVEL	Command Position FIFO Level.
40 – 71	GET_CTRL_DIGI_OUTPUTS	Digital Output States (0 – 31)
80 – 85	GET_CTRL_AXIS_FEEDBACK_POS	Axis Feedback Position
100 – 101	GET_CTRL_ANALOG_INPUT	Analog Input Voltage in millivolts
120	GET_CTRL_MOTION_ACTIVE	Motion active (all axis). Returns “1” if motion is present, or “0” if not.
126	GET_CTRL_ERROR_STATE	Error LED state
130 – 137	GET_CTRL_ENCODER_COUNTS	Current encoder counts
140 – 171	GET_CTRL_DIGI_INPUTS_EXT	Digital Input States (32 – 63) located on P13 and P14
190 – 195	GET_CTRL_SEQUENCE_IN_PROGRESS	Returns “1” if a motion sequence is in progress, or “0” if not.
200 – 205	GET_CTRL_AXIS_HOMED	Returns “1” if the axis is homed, or “0” if not.
210 – 215	GET_CTRL_AXIS_MOTION_ACTIVE	Motion active (single axis). Returns “1” if motion is present, or “0” if not.

float GetControlFloat (int index)

Read **float-specific** internal data of the VSI Motion Controller specified by the “index” parameter. C index definitions are also available for ease of use.

Parameters:

- *index* – indicator for specific control data field (see table below).

Example: The following code snippet reads the velocity of each axis by using a loop:

```
int axis;
float axisVelocity[6];
for( axis = 0; axis < 6; axis ++ )
    axisVelocity [axis] = GetControl( GET_CTRL_AVG_VELOCITY + axis );
```

HiCON Control Parameters

Index	C #define	Description
90 – 95	GET_CTRL_AXIS_COMMAND_POS	Axis Command Position
110	GET_CTRL_THREADING_RPM	Current Threading RPM
180 – 185	GET_CTRL_AVG_VELOCITY	Average Velocity on Axis in units/min
400 – 405	GET_CTRL_AXIS_INPUT_GAIN	Axis Input Gain
410 – 415	GET_CTRL_COUNTS_PER_UNIT	Axis Counts Per Unit
500	GET_CTRL_DAC_COUNTS	Current Voltage on specified analog output in millivolts.

SetControl (int index, int value)

Sets internal data of the VSI Motion Controller specified by the “index” parameter. C index definitions are also available for ease of use.

Parameters:

- *index* – indicator for specific control data field (see table below).
- *value* – the new value for the control data field.

Example: The following code snippet turns off all digital outputs.

```
int index = 0;
for( index = 0; index < 32; index ++ )
    SetControl( SET_CTRL_DIGI_OUTPUTS + index, 0 );
```

HiCON Control Parameters

Index	C #define	Description
40 – 71	SET_CTRL_DIGI_OUTPUTS	Digital Output States (0 – 31)
80 – 85	SET_CTRL_AXIS_POSITION	Sets Axis 0 – 5 Position to the specified value
90	SET_CTRL_POSITION_SYNC	Inform Mach to Sync to the current axis positions. The position sync will occur only when all motor positions are still.
91	SET_CTRL_DRIVE_ENABLE	Arm/Disarm the motion controller. “Value” is a bitmask of which axes to arm (e.g. 7 => 00000111, arms axes: x, y, and z). A value of 0 will disarm the motion controller.
92	SET_CTRL_DISARM_STOP_MACRO	Set Macro to terminate when the controller disarms.
100	SET_CTRL_DAC_OUTPUT	Write milliVolts where “value” ranges from 0 to 10,000 to the Spindle DAC Channel
130 – 138	SET_CTRL_ENCODER_COUNTS	Write any value to Encoder Counter Channel 0 – 7
140 – 145	SET_CTRL_AXIS_INPUT_ID	Change axis control input index. This can be used to set an axis to mirror the command position of another axis (e.g. when using slave axes). The “value” can be set to 0 – 5 to specify which axis to mirror the command position from.
150 – 155	SET_CTRL_MOTION_OVERRIDE	Toggle Macro Axis Motion Override. A value of 1 allows ONLY the macro to control motion. A value of 0 removes the override.

3. Motion Functions

NOTE: Before motion commands can be performed, the controller must first be configured from within Mach3 or Mach4 (this only needs to be done once if running in standalone operation). Instructions for this can be found in the Mach3 or Mach4 integration manual.

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 on all axes can be verified via **GetControl(120)** or **GetControl(210 – 217)** for individual axes. Active motion can also be stopped by calling **CancelMove(axis)**, which cancels the motion process on the specified axis.

Motion Modes

Motion Functions utilize a “mode” parameter to indicate the type of motion to execute.

Mode Type	Value	C #define	Description
Incremental	2	MOVE_TYPE_RELATIVE	Position parameter is used as the distance from the current position.
Absolute	4	MOVE_TYPE_ABSOLUTE	Position parameter is used as the destination in absolute coordinates (as referenced to machine zero position).
Velocity	8	MOVE_TYPE_VELOCITY	Only the sign (+ or -) of the position parameter is used to determine the direction of the move, since a velocity move is an infinite move in a given direction.

Motion Error Codes

Motion functions return a non-zero error code. A return value of zero indicates that the motion was started successfully with no errors.

C# define	Value	Description
MOTION_ERROR_NONE	0	Motion executed successfully.
MOTION_ERROR_EXEC	301	Motion could not be executed.
MOTION_ERROR_CANCEL	302	Motion could not be cancelled.
MOTION_ERROR_DIRECTION_CHANGE	303	This error is returned when an axis is currently executing motion, and a new motion command is called with a destination in the opposite direction. This can be avoided by cancelling the current motion or waiting for it to stop.

MOTION_ERROR_INVALID_PARAM	304	One of the parameters passed with the motion function was bad (e.g. accel or velocity was zero).
MOTION_ERROR_AXIS_DISABLED	305	The axis is currently disabled. This can be avoided by enabling the axis in the “SetControl(91, axisMap)” call.
MOTION_ERROR_AXIS_OUT_OF_RANGE	502	The specified axis does not exist.

long DoMotionAxis(int axis, float finalPosition, float speed, float accel, long mode)

Execute point to point motion for selected axis from current position.

Parameters:

- *axis* – the axis to perform motion with (*X=0, Y=1, Z=2, etc*).
- *finalPosition* – the destination position (actual usage depends on the *mode* parameter).
- *speed* – max speed of the motion (*units/min*).
- *accel* – acceleration of the motion (*units/sec²*).
- *mode* – ([see this section](#)).

Example: The following examples show the different modes of calling DoMotionAxis where “accel”=2.45 and “speed”=10.2

```
DoMotionAxis( 1, 500.25, 10.2, 2.45, 2 ) // Moves axis 1 to 500.25 units from the current position.
```

```
DoMotionAxis( 1, 500.25, 10.2, 2.45, 4 ) // Moves axis 1 to 500.25 units from the zero reference.
```

```
DoMotionAxis( 1, -1, 10.2, 2.45, 8 ) // Moves axis 1 infinitely in the negative direction.
```

If active motion from a prior DoMotionAxis() 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 commanding the new motion.

Example: How to properly change the axis motion direction.

```
int axis = 0; // This example assumes that the axis is at position zero
DoMotionAxis( axis, 100, 100, 10, 4 ); // move to the positive direction

while ( GetControl( 210 + axis ) != 0 ) // wait for the current motion to finish
    Sleep (10);

DoMotionAxis( axis, -100, 100, 10, 4 ); // move the axis to the negative direction
```

[long DoMotionXYZ\(float posX, float posY, float posZ, float speed, float accel, long mode\)](#)

Execute a coordinated point to point motion for the x, y, and z axes.

Parameters:

- posX – the destination position of the X axis (actual usage depends on the *mode* parameter).
- posY – the destination position of the Y axis (actual usage depends on the *mode* parameter).
- posZ – the destination position of the Z axis (actual usage depends on the *mode* parameter).
- speed – max speed of the coordinated linear motion (*units/min*).
- accel – acceleration of the coordinated linear motion (*units/sec²*).
- mode – ([see this section](#)).

Example: Move X to 500.25, Y to 600.25, and Z to 700.25 inches, with an acceleration of 2.45 inches/sec² and maximum speed of 10.2 inches/minute (assuming units are set to inches in Mach).

```
DoMotionXYZ( 500.25, 600.25, 700.25, 10.2, 2.45, MOVE_TYPE_ABSOLUTE );
```

[long DoLinearMove \(long axisMap, float* axisPositions, float speed, float accel, long mode\)](#)

Execute a coordinated point to point move on the specified axes.

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).
- axisPositions – an array of floats *float[6]* for *HiCONS* which contains the target positions for the designated axes.
- speed – max speed of the coordinated linear motion (*units/min*).
- accel – acceleration of the coordinated linear motion (*units/sec²*).
- mode – ([see this section](#)).

Example: Move (to absolute coordinates) X to 50.25, Y to 60.9, and A to 70 inches, with an acceleration of 2.45 inches/sec² and maximum speed of 10.2 inches/minute (assuming units are set to inches in Mach).

```
float axisPositions[8];
axisPositions[0] = 50.25; // X axis position
axisPositions[1] = 60.9; // Y axis position
axisPositions[3] = 70;   // A axis position

int axisMap = 11; // binary (00001011)

DoLinearMove( axisMap, axisPositions, 10.2, 2.45, MOVE_TYPE_ABSOLUTE);
```


long DoHoming(long axis, float homePosition, float speed, float accel, long homingParams)

Execute a homing sequence on the specified axis to find its point of reference.

Parameters:

- *axis* – the axis to home ($X=0, Y=1, Z=2$, etc).
- *homePosition* – the axis position will be set to this value when the homing process is successful.
- *speed* – max speed of the coordinated linear motion (*units/min*).
- *accel* – acceleration of the coordinated linear motion (*units/sec²*).
- *homingParams* – bitmask used for specifying the following options:
 - *Bit0* – Reverse homing direction.
 - *Bit1* – Use Home Sensor.
 - *Bit2* – Use Index Pulse.

NOTE: This function will return an error if both “Use Index Pulse” (bit1) and “Use Home Sensor” (bit2) are set to zero.

It is important to monitor the result of the homing routine by checking `GetHiCON(190 – 195)` to check if the homing process is still ongoing for the axis. `GetHiCON(200 – 205)` is used to check if the home position has been found for the specified axis upon completion of the homing process.

Homing Sequence for Home Sensor (no index pulse):

1. Move in one direction attempting to search for the home sensor.
2. Once the sensor is found, it reverses direction (using 20% of the specified velocity) to unblock the sensor
3. The moment the sensor is unblocked, the homing sequence is completed and the “homePosition” is used as the new home position.

Homing Sequence for Index Pulse Only:

1. If the index pulse is already active, the axis moves a short distance to move away from the index pulse.
2. The axis moves in one direction attempting to find the Index Pulse.
3. The moment the index pulse is found, the homing sequence is completed and the “homePosition” is used as the new home position.

Homing Sequence for Home Sensor with Index Pulse:

1. Move in one direction attempting to search for the home sensor.
2. Once the sensor is found, it reverses direction (using 20% of the specified velocity) to unblock the sensor
3. When the home sensor is unblocked, the axis keeps moving until the index pulse is found.
4. The moment the index pulse is found, the homing sequence is completed and the “homePosition” is used as the new home position.

Example: Home the X axis by using the home sensor and index pulse.

```
int homingParams = 0;
homingParams |= 2;    // use home sensor
homingParams |= 4;    // use index pulse

DoHoming(0, 0, 50, 10, homingParams);
```

CancelMove(*long* axis, *long* instantStop)

Cancels the active motion on the specified axis.

Parameters:

- *axis* – the axis on which to cancel motion (*X=0, Y=1, Z=2, etc.*).
- *instantStop* – indicates how the axis should stop (0 = decelerate to a stop, 1 = stop instantly).

Example: Stop motion on the X and Z axis.

```
CancelMove(0, 0);  
CancelMove(2, 0);
```

4. Arc/Circle Motion Functions

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.

long DoArcMoveCenter (long axisOfRotation, float* axisPositions, float offset1, float offset2, float speed, bool clockwise)

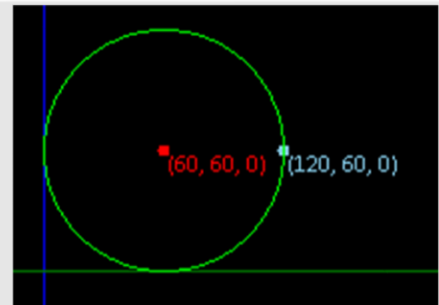
Execute an Arc/Circular trajectory move around an axis of rotation by specifying a center point relative to the initial position.

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 floats *float[6]* for *HICONs* 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);
```



long DoArcMoveRadius (long axisOfRotation, float* axisPositions, float radius, float 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°.

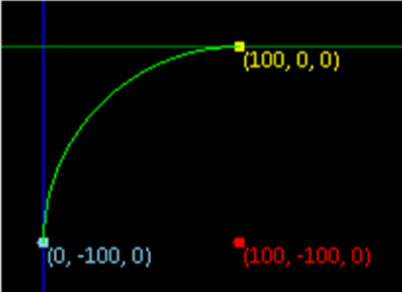
For arcs larger than 180°, use “DoArcMoveCenter”, or call this function twice.

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 floats `float[6]` for HiCONs 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);
```



long DoArcMoveAngle (long axisOfRotation, float* axisPositions, float arcAngle, float 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 floats *float[6]* for HICONs 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);
```

III. Debug Registers

When running a VSI C-Macro program, debug registers are allocated by default. These registers are automatically updated in the VSI Macro Loader Output View to reflect their current values in the VSI C-Macro program, and as such, are a helpful tool when watching values is necessary.

The number of available debug registers are as follows:

- `DebugRegisters[8]` (32-bit integer registers)
- `DebugFloats[8]` (32-bit floating point registers)

Example: The following code snippet writes the current axis feedback positions (in counts) to the debug registers for watching.

```
int index = 0;
for(index = 0; index < 8; index++)
    DebugRegisters[index] = GetControl(GET_CTRL_AXIS_FEEDBACK_POS + index);
```

Example: The following code snippet writes the current feedback positions (in units) to the debug registers for watching.

```
int index = 0;
for(index = 0; index < 8; index++)
{
    float feedbackCounts = GetControl(GET_CTRL_AXIS_FEEDBACK_POS + index);
    float countsPerUnit = GetControl(GET_CTRL_COUNTS_PER_UNIT + index);

    DebugFloats[index] = (feedbackCounts / countsPerUnit);
}
```

Debug Registers	
[0]	44
[1]	1044
[2]	2044
[3]	3044
[4]	4044
[5]	5044
[6]	6044
[7]	7044
Debug Floats	
[0]	0.398609
[1]	0.398609
[2]	0.398609
[3]	0.398609
[4]	0.398609
[5]	0.398609
[6]	0.917121
[7]	0.434631