



**Just the FAQ's on
Integrated Motion Control
with Think & Do Software
THINK & DO - APPLICATION NOTE #00023
(updated Feb 2000 for new commands)**

IS MOTION CONTROL AS EASY TO LEARN AND FUN TO USE AS EVERYTHING ELSE IN THINK & DO?

The answer to this question is a definitive YES! Think & Do realized there are lots of *discrete control* applications that also involve some form of *motion control*. Think & Do has added a motion control block to its flowchart programming environment and motion control drivers to its IO interface system. Motion blocks are used just like other flowchart blocks and allow a user to develop control logic that is independent of the specific motion card that is eventually used in the system. In addition, configuration of motion cards is graphical and intuitive, utilizing the same integrated tagname database as flowcharts and HMI.

Think & Do supports motion controllers from Galil, MEI, Douloi and AcroLoop. Now with Think & Do you can use one cohesive set of tools to solve the motion control, discrete control, PID or HMI portion of your application. Oh yeah, and it's ½ the price of anything else on the market! There are also serial, DeviceNet, ProfiBus and Ethernet (in the works) motion control solutions available.

WHAT IS INTEGRATED MOTION CONTROL?

Integration of discrete control with motion commands

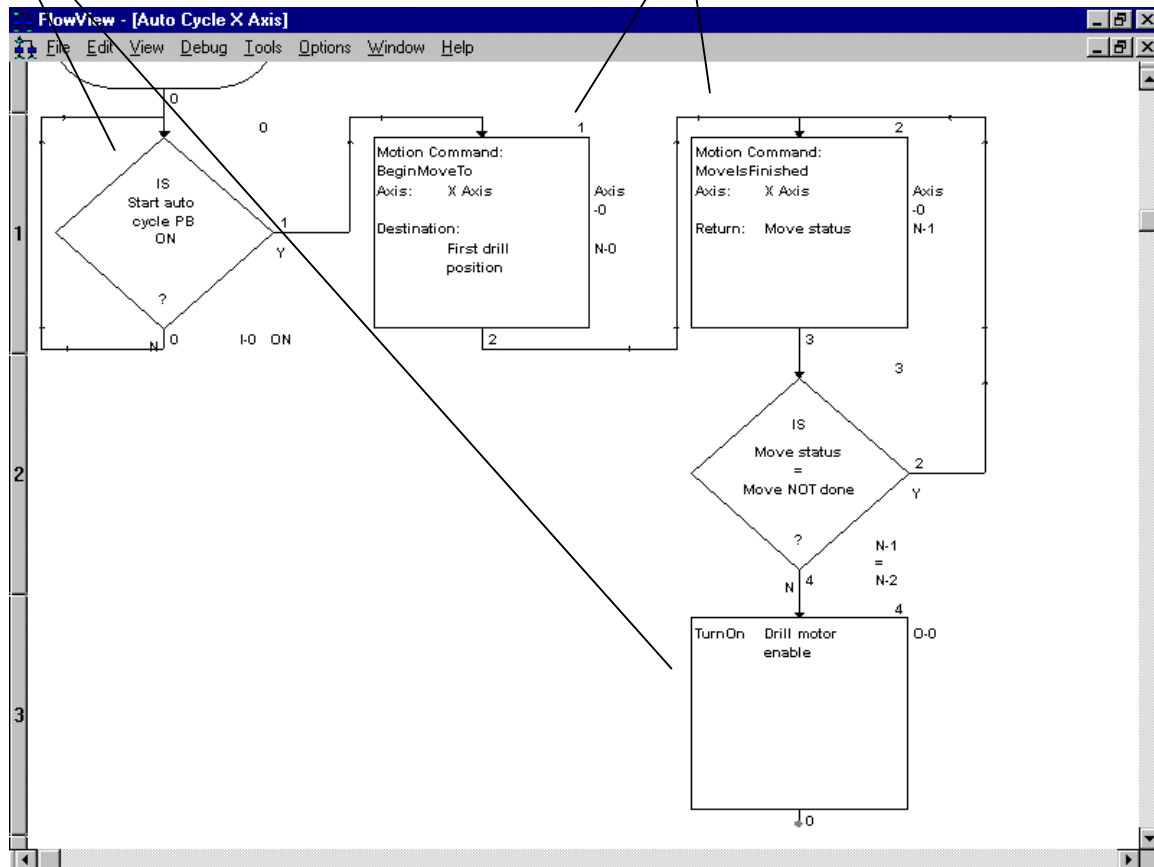
Discrete control commands are typically used for doing things like - turning ON a digital output to energize a solenoid, starting a timer to see if an action has timed-out, incrementing a counter when a part passes a proximity switch, moving data from one place to another. Motion control commands are for making motion - commanding a motor to position an axis at a specific location, jogging a mechanical slide into position during machine setup, moving a part located on an X-Y-Z table into position so it can be drilled, homing an axis, etc.

Integration means you have the luxury of tightly linking together the discrete control logic (which may be doing things like your material handling functions or controlling the tooling) with the motion commands. Integration also gives the control programmer much greater flexibility and power in designing useful *diagnostics* for the process. *Diagnostics* that can tell the operator what is wrong so he can keep the process running more efficiently.

Integration of these two classes of operations means allowing one cohesive language and programming environment for creating modular logic. Following is a figure showing what some flowchart logic would look like with motion and discrete control commands:

DISCRETE CONTROL BLOCKS

MOTION BLOCKS



This logic sequence goes like this:

1. This flowchart starts out in a decision block waiting for the input signal “Start Auto Cycle PB” to come ON. This is block 0, a discrete control block.
2. Once it gets this input signal it will execute a motion block and issue the command “BeginMoveTo” using “X Axis” as the axis and a parameter “First Drill Position” as the location. Since “First Drill Position” is an internal variable it’s value could come from some user input field in the HMI, a data file with part program positions or it may have been calculated by some other logic based on other parameters. (block 1)
3. After issuing the “BeginMoveTo” command to the motion controller (this will take about 1 msec) it branches to yet another motion block issuing another command to the motion controller. This next command is really a query “MoveIsFinished” which asks the controller if the last move command has reached it’s target yet. MoveIsFinished returns a number representing the status of the move. (block 2)
4. Block 3 is a decision block that looks at the returned value from MoveIsFinished and if it’s 0 block 3 will loop back to block 2 and issue the query again. So while the motion controller is controlling the axis to finish the move this particular flowchart is in a loop executing blocks 2 and 3 to determine when the move is finished. This loop

that contains blocks 2 & 3 is a good place to insert some diagnostics - to check for timeout conditions or maybe if some other error exists in the system.

5. After block 3 (we are now sure the axis has reached it's target position successfully) we execute block 4 which is another discrete control block that turns on the drill device to drill the part. After this the flowchart may go on to execute some other motion blocks, data manipulation or whatever the programmer deems necessary.

WHAT ARE THE BENEFITS OF MOTION WITH THINK & DO?

Easy to learn and Fun to use

Easy to Learn means being able to graphically program motion actions and discrete control in a user-friendly programming environment - drag and drop programming at it's finest. Motion commands are in an easy to understand language; e.g. "BeginMoveTo Axis-X 4000" will make an absolute move of Axis-X to a position of 4000. Diagnostics can be easily incorporated with flowchart-based logic around the motion commands as well.

It's easier to learn one method of programming than three

Our philosophy about programming languages is that engineers today have too many other technologies to master, they don't need to learn yet another programming language. That's why we have such a strong belief in flowcharts. One language can do it all and it doesn't require knowledge of an arcane syntax. One method for programming all of the elements of your control problem just makes more sense. However, if your motion application is very specialized, and needs to be more tightly linked to the low-level commands available on the motion control hardware, then we don't prevent you from using those facilities available on the card itself (i.e. you can still use the motion control vendors proprietary language).

Spend more time solving your application problem and less learning the idiosyncrasies of three programming languages

Today's automation requirements are putting a greater burden on engineers to design more user friendly user interfaces for the operator, thorough diagnostics and improved process control. Engineers would rather spend time on devising creative and effective methods to solve their application problems - less time on mastering multiple programming languages. This is the answer to that predicament.

Tight integration with discrete control commands

Being able to mix motion commands with discrete control commands has many benefits. Diagnostics - while in a decision block waiting for an axis to reach it's target position you can add other blocks to check for some mechanical sub-assembly that may not be in position. Coordination between control blocks, motion and data.

Motion hardware configuration is graphical

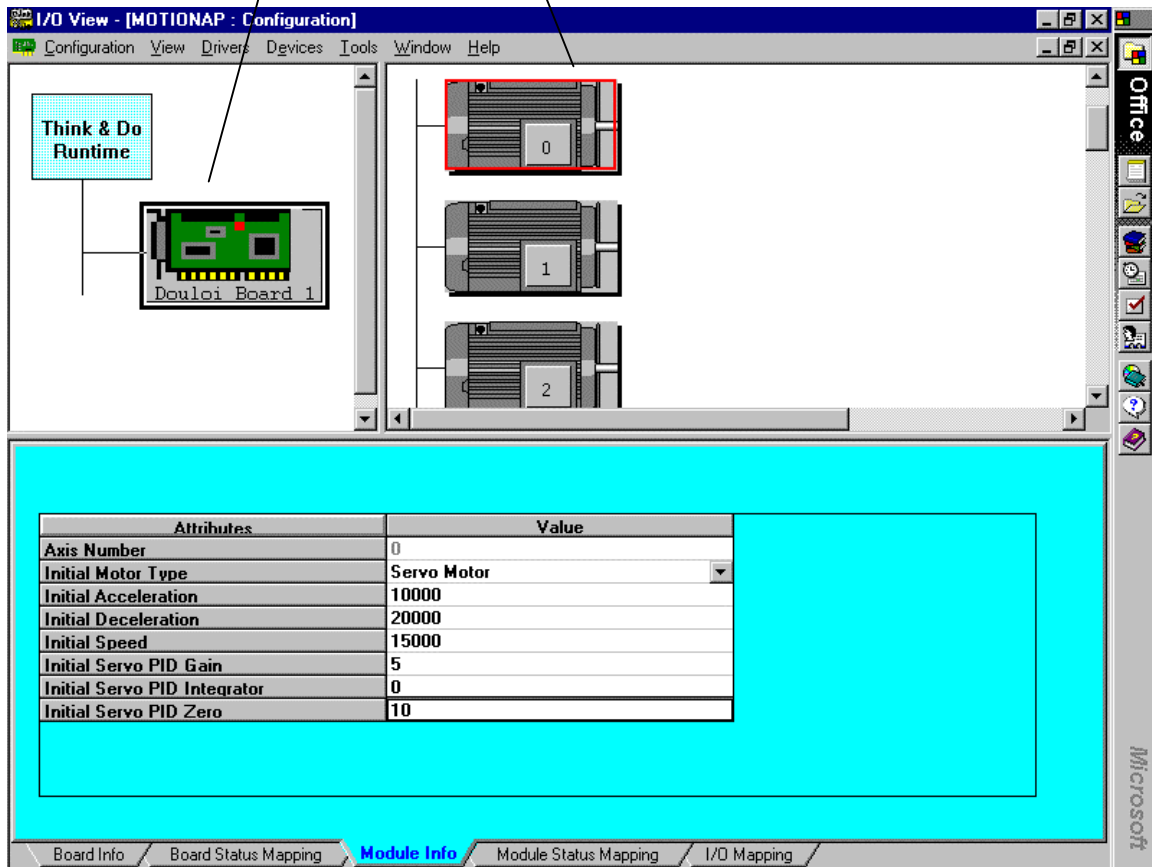
The motion controller interface card looks like an IO driver to Think & Do's IO configurator (IOVIEW). The steps required to configure the IO interface for the motion hardware are:

1. Select the motion controller driver (Douloi, AcroLoop, etc). This will create a graphical image of the interface card and physical motors (axes).
2. Map physical axes to logical ones in any order you like (point and click just like you map inputs and outputs for digital IO).
3. Configure default parameters for individual axes. Parameters such as motor type (servo or stepper), acceleration, deceleration, speed, PID, etc. This is a fill in the table operation within IOVIEW.

You could probably say that was as easy as 1-2-3... but wait! It gets even easier! With the addition of two motion blocks in a flowchart, one to enable the axis and one to issue a jog command, you can have an axis operational already.

An example of IOVIEW configuration follows:

Motion controller interface card AND motors (axes) -



This tab shows an individual motors' initial setup parameters (speed, accel, etc). The other tabs are used for mapping physical axes to logical axes names and motion system status information.

HOW DO MOTION BLOCKS WORK WITHIN A FLOWCHART?

So you want to know how the motion control blocks, within the flowchart, interact with the motion controller PC card. What goes on *under the covers* so to speak. Here's the scoop:

Motion controller hardware/firmware closes the loop

The *motion loop* is not closed by the software. It is closed and controlled by the motion controller hardware. Typically the motion controllers use a DSP-based architecture, some rely more on software but the software executes on a co-processor (i.e. the main CPU of the PC is not used). Most motion controllers can run standalone, they do not need the host

PC except as an HMI and source of commands/parameters. So, if the host PC should go down for some reason the motion controller will still control the axes in a stable fashion. Most motion controllers have some sort of watchdog circuitry such that if the *following error* gets too large, or some external ESTOP circuit tells them to stop the motors, they will stop the motion safely.

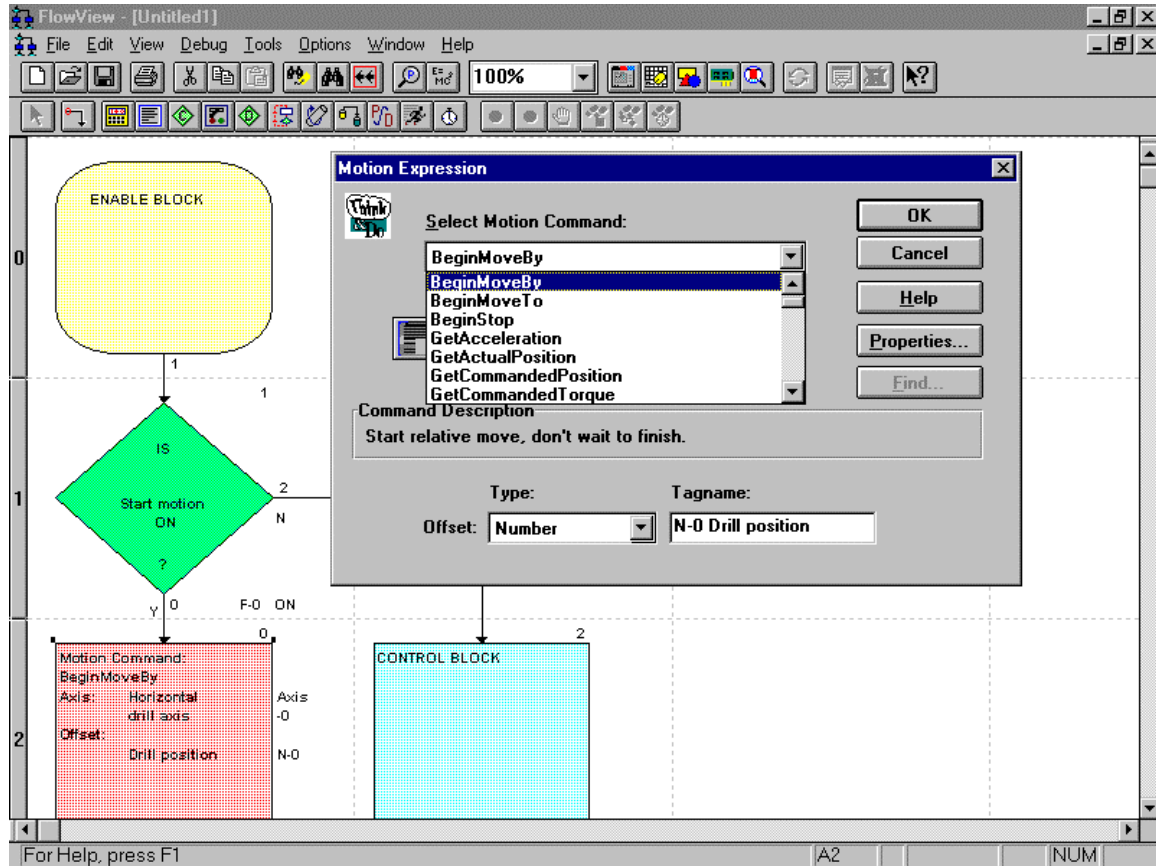
Motion commands and parameters get sent to motion controller

Think & Do is a host program to the motion controller system - we are feeding commands and parameters to the controller. The commands are things like *abort move*, *set acceleration*, *move to absolute position*. These commands typically have parameters and when a block is executed the command with it's parameters gets sent to the controller. It takes the controller some finite amount of time to digest each of these commands (1 msec typically but it varies depending on the motion controller vendor). A motion controller may be executing one command while another is being issued to it (example - executing a *MoveToAbsolutePosition* while a new command is issued to *SetSpeed* to a new value).

WHAT CAN A MOTION BLOCK DO?

Motion blocks get executed within flowchart logic just like discrete control blocks. Remember that a project consists of many flowcharts executing at once. At any instant of time there may be multiple flowcharts issuing many motion commands to the controller simultaneously. Some motion blocks are *commands* and some are *queries*, returning attributes of the controller or last motion command.

Here's a sample of what the motion block dialog looks like in the flowchart editor:



Generic commands versus vendor-specific

Think & Do has implemented the motion block so that a generic set of motion commands can be used on any vendor's motion controller. So, what this means is you can write a flowchart program using the generic motion commands with *vendor A* controller and then use the same flowchart program with *vendor B's* controller (logic can be reusable from one project to another regardless of the controller vendor - what a COOL idea!) Of course this is dependent upon use the generic commands. There are currently 38 generic motion commands (see below). Vendor-specific commands are for implementing motion commands which are unique to a particular vendor's controller. Examples include BeginUserTask, SetUserFloat, ConfigureIOBitAsOutput... features which a motion vendor has implemented in a unique way.

Motion commands

This is a list of the motion commands and their meaning. They are grouped according to whether they are generic or vendor-specific. Parameters are usually an axis name, some position (integer as a data item, array elements, or fixed), possibly a floating point parameter in some special cases (where noted).

GENERIC MOTION COMMANDS

COMMAND	DESCRIPTION	PARAMETERS
Abort	Abruptly stop the motion. Does not do a controlled decel. Generally used for emergency use only.	axis
BeginMoveBy	Start RELATIVE move, don't wait to finish	axis, offset
BeginMoveTo	Start absolute move, don't wait to finish	axis, position
BeginStop	Decelerate and stop motion gracefully	axis
GetAcceleration	Return current acceleration	axis, return value
GetActualPosition	Return current position	axis, return value
GetCommandedPosition	Return motor position that was commanded	axis, return value
GetCommandedTorque	Return current torque setting. With most motion controllers you can set a limit on the torque output.	axis, return value
GetDeceleration	Return current decel setting	axis, return value
GetDestinationPosition	Return position where move will finish	axis, return value
GetErrorLimit	Return current error limit (following error of servo) setting	axis, return value
GetErrorPosition	Return difference between commanded position and actual (actual following error)	axis, return value
GetNegativeLimit	Return current negative limit setting (soft limit of axis travel)	axis, return value
GetPIDGain	Return current PID gain value (Proportional term)	axis, return value
GetPIDIntegrator	Return current PID integrator value (Integrator term)	axis, return value
GetPIDZero	Return the PID damping factor (derivative)	axis, return value
GetPositiveLimit	Return current positive (soft) limit	axis, return value
GetProfileVelocity	Return current commanded speed	axis, return value
GetSampleRate	Return current sample rate frequency. Some controllers let you configure the	axis, return value

	motion planning system sample rate (ex - 250 -1000 usec)	
GetSpeed	Return current speed setting	axis, return value
Jog	Command axis at specified speed indefinitely	speed as Number/Array/Fixed Integer
MovelsFinished	Returns non zero if move is finished. Issued after a <i>move absolute</i> or <i>move relative</i> to determine if target location is reached.	axis, return value
SetAccel	Set accel rate in counts per second squared	axis, integer
SetActualPosition	Set what physical position should be. i.e. redefine current position. This would typically be used during a homing sequence.	axis, position
SetCoordinateInversion	Change the position regarded as positive.	Axis, Inversion state
SetDecel	Set decel rate used for move ends and stops	axis, decel rate
SetErrorLimit	Set the lag limit for command vs. actual position (following error)	axis, limit
SetLoopInversion	Set sign change for feedback loop. This is handy when you don't want to have to physically rewire the encoder feedback.	Axis, inversion state
SetMotorEnable	Turn motor on or off	axis, motor state
SetMotorType	Set motor to be servo or stepper	axis, motor type
SetNegativeLimit	Set negative direction boundary (soft limit)	axis, neg limit
SetPIDGain	Set the PID gain value (proportional term)	axis, gain
SetPIDIntegrator	Set the PID integrator value	axis, integrator
SetPIDZero	Set PID damping factor	axis, damping value
SetPositiveLimit	Set positive direction boundary (soft limit)	axis, positive limit
SetSampleRate	Set controller sample rate	axis, sample rate
SetSpeed	Set the slew speed for an axis	axis, speed

NOTE1: All parameters are AXIS or INTEGER (32 bit 'Number' within Think & Do data table) or FLOAT (for vendor specific). Most integer parameters can be specified as 1) fixed integers, 2) variables ('Number' data item within Think & Do) or 3) as an element in an array. Arrays are very powerful because this means you could do something like store all your X-Y table positions in an array and create some very compact logic that

would iterate through all the X-Y locations stored in the array and move your part to all those X-Y positions.

NOTE2: Integer parameters are 32 bit integers.

VENDOR SPECIFIC MOTION COMMANDS

Vendor-specific commands fall into a few categories such as:

- Task control/monitoring of tasks on the motion controller hardware.
- Configuring and read/write of local I/O on the motion controller hardware.
- Reading/writing of global data that is local to the card, such data is usually being used by tasks which reside on the motion controller itself.
- Watchdog hardware functionality on the motion controller.
- Issuing of vendor specific (native) commands (as if you were writing code to run on the card itself)

So far these commands have been implemented for vendor specific features.

Douloi

COMMAND	DESCRIPTION	PARAMETERS
Douloi->BeginUserTask	Begin an independent task that will run on the motion controller. These tasks run at a much higher frequency (250 usec) than flowchart tasks within NT. These types of tasks must be written in a vendor-specific language (ex - Pascal in this case). They are only useful for very high speed types of operations. Typically not required.	axis, task name
Douloi->ResetWatchDog	Reset the motion system watchdog	axis
Douloi->UserHasDisabled	Returns non-zero if user has disabled the system	axis, return value
Douloi->WatchDogHasTripped	Returns non-zero if watchdog has shut down motion activity	axis, return value
Douloi->AbortUserTask	Abort independent user task running on the motion hardware	axis, task name
Douloi->UserTaskPresent	Returns non-zero if task is present on Douloi card	axis, return value
Douloi->SetUserBoolean	Write a boolean value to a global variable defined on the card. This is typically a value that is being used by <i>user tasks</i> on the Douloi hardware. These types of commands are used to read/write user-defined	axis, variable index , value

	data structures on the card.	
Douloi->SetUserLongInteger	Write a long int (32 bit) value to global data structure on card.	axis, variable index ,value
Douloi->SetUserFloat	Write a floating point (32 bit) value to global data structure on card.	axis, variable index ,value
Douloi->GetUserBoolean	Returns value of a user defined boolean value from the card	axis, return value , variable index
Douloi->GetUserLongInteger	Returns value of a user defined long integer value from the card	axis, return value , variable index
Douloi->GetUserFloat	Returns value of a user floating point value from the card	axis, return value , variable index
Douloi->ConfigureIOBitAsOutput	The Douloi card has it's own local IO (48 bits user-configurable as out or inputs) .For basic motion applications you will not need to use it. If you need to configure it you can do so with this command.	Axis, bit number , value
Douloi->SetOutputEnable	Enable the I/O bits configured as outputs	axis, value
Douloi->SetOutputBit	Set the value of an output bit	axis, bit number, value
Douloi->GetInputBit	Return the value of an I/O bit	axis, return value, bit
Douloi->ArmIndexCapture	Arm an axis to capture it's position on it's index pulse (used for homing)	axis
Douloi->ArmInputCapture	Arm an axis to capture it's position when it receives an input pulse (used for an alternative method of homing)	axis
Douloi->CaptureHasTripped	Returns non-zero when position is captured on index or input pulse	axis, return value
Douloi->GetCapturePosition	Once a capture has occurred this command will return the position of the axis at the instant that the capture event occurred	axis, return
Douloi->SetCaptureTrip	Used to establish what signal transition constitutes a capture event (low-to-high or high-to low)	axis, return
Douloi->SetCommandedTorque	Used only when the motor is being run in "open loop", this procedure sets the value for the digital to analog convertor for the physical axis related to this t1Axis	axis, value
Douloi->SetEnable	If SetEnable(on) then the analog output for the receiving axis or axis	

	group is turned on , and the amp enable is asserted. If SetEnable(off) the analog voltage is set to 0 and the amp enable line is not asserted.	
--	--	--

MEI

COMMAND	DESCRIPTION	PARAMETERS
MEI->SetStop	Generate a Stop Event on an axis	axis
MEI->GetNegativeLevel	Get the active state of the negative limit input	axis, level
MEI->GetPositiveLevel	Get the active state of the positive limit input	axis, level
MEI->ControllerRun	Clear an Abort Event on an axis	axis
MEI->ClearStatus	Clear a Stop Event or an E-Stop Event on an axis	axis
MEI->ControllerIdle	Generate an Abort Event on an axis	axis
MEI->GetInPosition	Get the width of the in-position window	axis, window
MEI->SetInPosition	Set the width or the in-position window	axis, window
MEI->GetErrorLimit	Get the maximum position error limit	axis, window, action
MEI->SetErrorLimit	Set the maximum position error limit	axis, window, action
MEI->GetNegativeSwLimit	Get the negative software position limit	axis, pos, action
MEI->SetNegativeSWLimit	Set the negative software position limit	axis, pos, action
??MEI->GetPosSWLimitAction		
??MEI->SetPosSWLimitAction		
MEI->StartTMove	Begin a non-symmetrical trapezoidal profile move	axis, final, vel, accel, decel
MEI->GetPositiveSWLimit	Get the positive software position limit	axis, pos, action
MEI->SetPositiveSWLimit	Set the positive software position limit	axis, pos, action
MEI->GetAmpFault	Get the amp fault input action	axis, action
MEI->SetAmpFault	Set the amp fault input action	axis, action
MEI->GetHome	Get the home/index logic action	axis, action
MEI->SetHome	Set the home/index logic action	axis, action
MEI->GetNegativeLimit	Get the negative limit input action	axis, action
MEI->SetNegativeLimit	Set the negative limit input action	axis, action

MEI->GetPositiveLimit	Get the positive limit input action	axis, action
MEI->SetPositiveLimit	Set the positive limit input action	axis, action
MEI->GetAmpEnableLevel	Get the Run state or the amp enable output	axis, level
MEI->GetAmpFaultLevel	Get active state of amp fault input	axis, level
MEI->GetHomeLevel	Get the active state of the home/index logic	axis, level
MEI->SetEStop	Generate an E-Stop Event on an axis	axis, rate

Galil

COMMAND	DESCRIPTION	PARAMETERS
Galil->Abort	Stops a motion instantly without a controlled deceleration.	axis
Galil->GetPhysicalAxis	Returns the physical axis number.	axis
Galil->GetScaleFactor	Returns the scale factor value.	axis
Galil->ReadAnalogChannel	Reads the analog input value from the controller.	axis, channel #
Galil->UploadFile	Uploads the specified file from the controller.	axis, file name
Galil->DownloadFile	Downloads file to the controller.	axis, file name
Galil->SendCommand	Sends ASCII value that corresponds to the command you want to send to the controller. * Send ANY Galil native command * BGX, SP 1000, etc	axis, ASCII string
Galil->SetSpeed	Sets the slew speed of any or all axes for independent moves.	axis, speed
Galil->SetSampleRate	Sets the control sample rate frequency.	axis, rate
Galil->SetPositiveLimit	Sets the positive direction boundary.	axis, limit
Galil->SetPIDDerivative	Sets the PID derivative value.	axis, derivative
Galil->SetPIDIntegrator	Sets the PID integrator value.	axis, integrator
Galil->SetPIDGain	Sets the PID gain.	axis, gain
Galil->SetNegativeLimit	Sets negative direction boundary.	axis, limit
Galil->SetMotorType	Set specified axis to servo or stepper.	axis, type
Galil->SetMotorEnable	Enable servo control using the current position.	axis
Galil->SetErrorLimit	Set the lag limit for command vs. actual position.	axis, limit
Galil->SetDecel	Sets the linear deceleration rate of the motors for independent moves.	axis, decel. rate
Galil->SetActualPosition	Sets the axis' physical position.	axis, position
Galil->SetAccel	Sets the linear acceleration rate of	axis, accel. rate

	the motors for independent moves.	
Galil->MoveIsFinished	Returns a non-zero value if a move is finished.	axis
Galil->GetSpeed	Returns current speed setting.	axis
Galil->Jog	Commands specified axis at specified speed indefinitely.	axis, speed
Galil->GetSampleRate	Returns current sample rate frequency.	axis
Galil->GetProfileVelocity	Returns current commanded speed.	axis
Galil->GetPositiveLimit	Returns positive direction boundary.	axis
Galil->GetPIDDerivative	Returns current PID derivative value.	axis
Galil->GetPIDIntegrator	Returns Current PID integrator value.	axis
Galil->GetPIDGain	Returns current PID gain value.	axis
Galil->GetNegativeLimit	Returns negative direction boundary.	axis
Galil->GetErrorPosition	Returns difference command vs actual position.	axis
Galil->GetErrorLimit	Returns current error limit setting.	axis
Galil->GetDestinationPosition	Returns where move will finish.	axis
Galil->GetDeceleration	Returns current deceleration.	axis
Galil->GetCommandedTorque	Returns current torque setting	axis
Galil->GetCommandedPosition	Returns the commanded position.	axis
Galil->GetActualPosition	Returns the current physical condition.	axis
Galil->GetAcceleration	Returns current acceleration.	axis
Galil->BeginStop	Decelerates and stop motion.	axis
Galil->BeginMoveTo	Starts absolute move, doesn't wait to finish.	axis, destination
Galil->BeginMoveBy	Starts relative move, don't wait to finish.	axis, offset

Acroloop

COMMAND	DESCRIPTION	PARAMETERS
Acroloop->SetMoveCounter	Sets axis move counter and increments.	axis, counter, increment
Acroloop->GetMoveCounter	Returns the axis move counter value.	axis
Acroloop->SetJerk	Sets the jerk for an axis. If jerk is zero, the acceleration profile is rectangular otherwise it is	axis, jerk

	trapezoidal.	
Acroloop->SetStopRamp	Sets the deceleration ramp to be used at the end of the next move. If value is zero, then the move will end without ramping, otherwise the move will ramp down at the specified rate to the velocity set by the FVEL command.	axis, decel.
Acroloop->SetDeceleration	Sets the deceleration for an axis.	axis, decel.
Acroloop->SetAcceleration	Sets the acceleration for an axis.	axis, accel.
Acroloop->SetSpeed	Sets the slew speed for axis motion.	axis, speed
Acroloop->BeginMoveBy	Starts relative move, and doesn't wait to finish.	axis, offset
Acroloop->BeginMoveTo	Starts the absolute move and doesn't wait to finish.	axis, destination
Acroloop->BeginMoveByNoRamp	Starts relative move, and doesn't wait to finish with no stop ramp.	axis, offset
Acroloop->BeginMoveToNoRamp	Starts the absolute move and doesn't wait to finish with no stop ramp.	axis, destination
Acroloop->BeginMoveByNoRamp(F)	Starts relative move, and doesn't wait to finish with no stop ramp (using Float as offset).	axis, offset
Acroloop->BeginMoveToNoRamp(F)	Starts the absolute move and doesn't wait to finish with no stop ramp (using Float as offset)	axis, destination
Acroloop->FastStatus	Performs a fast status request.	axis
Acroloop->Jog	Jogs reverse, forward, or stop depending on the value of jog state.	axis, jog state
Acroloop->SetJogParams	Sets the acceleration, speed and deceleration for jogging.	accel., speed, decel.
Acroloop->SetFloat	Sets a float parameter.	axis, parameter index, value
Acroloop->SetLong	Sets a long parameter.	axis, parameter index , value
Acroloop->SetFlag	Sets a flag parameter.	axis, flag #, value
Acroloop->GetFloat	Gets a float parameter.	axis, parameter index
Acroloop->GetLong	Gets a long parameter.	axis, parameter index
Acroloop->GetFlag	Gets a flag parameter.	axis, flag #
Acroloop->GetUserDouble	Gets a user double variable.	axis, program #, array #, parameter index
Acroloop->GetUserSingle	Gets a user single variable.	axis, program #,

		array #, parameter index
Acroloop->GetUserLong	Gets a user long variable	axis, program #, array #, parameter index
Acroloop->SetUserDouble	Sets a user long variable	axis, program #, array #, parameter index, value
Acroloop->SetUserSingle	Sets a user single variable	axis, program #, array #, parameter index, value
Acroloop->SetUserLong	Sets a user long variable	axis, program #, array #, parameter index, value
Acroloop->SendString	Send a 'string' command to Acroloop controller. These are the native Acrobasic commands. Basically what this means is that any command which the Acroloop card understands can be used.	axis, string

Configuration

Some initial configuration is done through IOView when you set-up the driver interface (things such as initial accel/decel/speed and PID values). All configuration settings are also available through the motion block (accel/decel/speed, following error, soft limits, etc). These settings can be made on-the-fly (depending on whether motion controller supports changing such parameter). As an example you could change the speed while an axis was approaching it's target position.

ARE SPECIAL DATA ITEM TYPES REQUIRED FOR MOTION?

Within Think & Do we have added one special data item type and that is the AXIS data item. An AXIS data item gets mapped to a physical axis on a specific piece of motion controller hardware. All the other parameters that are required for motion commands are mostly 32 bit integers ('Number' data item within Think & Do) or floating point numbers.

CAN I USE THINK & DO MOTION CONTROL WITH STEPPER'S OR SERVO'S?

Yes, as long as the motion controller hardware you choose supports steppers (all of them do). If you want to user encoder feedback for verification of stepper position that is possible too.

IS THIS A GOOD PACKAGE FOR COORDINATED MOTION APPLICATIONS?

It depends on the motion controller card you choose. All the cards we support can deal with multi-axis coordinated moves. Typically this is done with the motion controller's native language. With Galil and Acroloop this can be accomplished directly from the Think & Do flowchart motion blocks. With Douloi it works a little differently.

We have had customers build complete CNC controllers using the Galil controller and Think & Do using some VB code for the G-code interpreter. We don't handle G-code/M-code part program execution, coordinate transformations, etc., directly from within Think & Do, however some of this functionality could be built-in using flowcharts. Some of the motion controllers we interface to are being used in CNC applications with custom written software front-ends. There are some limited CNC machine tool applications that may be more economically addressed using Think & Do's motion approach. Call one of our application engineers and ask.

WHAT DOES IT COST?

Think & Do costs \$1295.00, this includes all the software drivers, HMI, etc (there are no hidden costs). Motion controller interface cards costs anywhere from \$250 - \$700 per axis depending on the number of axes, performance and features of the card, and the vendor.