

## Chapter 4. Application Design

### Chapter Objectives

The information in this chapter will enable you to:

- Recognize and understand important considerations that must be addressed before you implement your application
- Understand the capabilities of the system
- Customize the system to meet your requirements
- Use examples to help you develop your application

### Motion Control Concepts

This section discusses basic motion control concepts that you should be familiar with as you develop your application.

#### Move Profiles

In any motion control application, the most important requirement is precise position, whether it be with respect to time or velocity. A motion profile represents the velocity of the motor during a period of time in which the motor changes position. The type of motion profile that you need depends upon the motion control requirement that you specify. The basic types of motion profiles are described below. The JSI can perform all of the profiles discussed in this chapter.

#### Triangular and Trapezoidal Profiles

For constant acceleration control systems, velocity, acceleration, and distance parameters are defined before the system can execute a preset move. The value of these parameters determines the type of motion profile as either triangular or trapezoidal.

A triangular profile results when the velocity and acceleration are set so that the motor does not attain the defined velocity before the motor travels half of the distance that you specify. This results from either a relatively low acceleration, a relatively high velocity, or both. The motion profile for this move is shown in Figure 4-1.

A trapezoidal profile (see Figure 4-1) results when the motor reaches the defined velocity before the motor moves half of the specified distance. A trapezoidal profile may occur if you specify a low velocity with a high acceleration or a long distance.

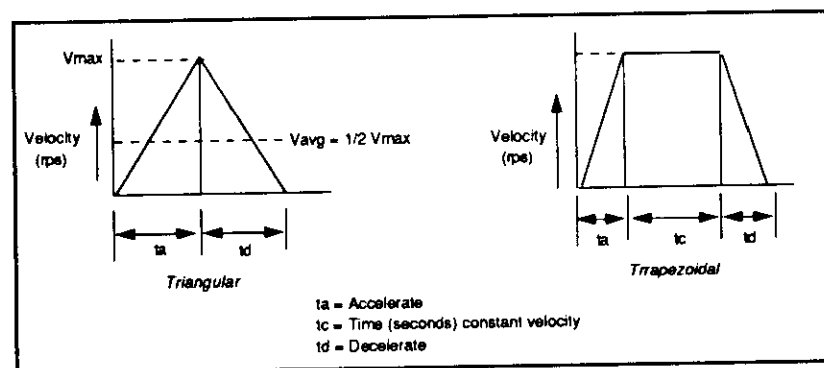


Figure 4-1. Triangular and Trapezoidal Profiles

#### Move Times: Calculated vs Actual

You can calculate the time it takes to complete a move by using the acceleration, velocity, and distance values that you define. However, you should not assume that this value is the actual move time. There is calculation delay and motor settling time that makes your move longer. You should also expect some time for the motor to settle into position. You can decrease the motor's settling time by tuning the JSI controller and the drive. Tuning is discussed later in this chapter.

**Incremental vs. Absolute Positioning**

A preset move is a move distance that you specify (in encoder steps). You can select preset moves by putting the JSI into normal mode using the Mode Normal (**MN**) command. Preset moves allow you to position the motor in relation to the motor's previous stopped position (incremental moves) or in relation to a defined zero reference position (absolute moves). You can select incremental moves by using the Mode Position Incremental (**MPI**) command. You can select absolute moves using the Mode Position Absolute (**MPA**) command

**Continuous Mode Moves**

The Mode Continuous (**MC**) command accelerates the motor to the velocity that you last specified with the Velocity (**V**) command. The motor continues to move at the specified velocity until you issue the Stop (**S**) command or specify a velocity change. To change velocity while the motor is moving, use the **V** command followed by the **G** command. The Continuous mode is useful for applications that require constant movement of the load, when the motor must stop after a period of time has elapsed (rather than after a fixed distance), or when the motor must be synchronized to external events such as trigger input signals.

Command	Description
> PS	Waits to Continue (C) command to execute next command
> IN1A	Sets up <b>IN 1</b> (Input 1) as trigger bit 1
> IN2A	Sets up <b>IN 2</b> (Input 2) as trigger bit 2
> LD3	Disables the CW and CCW limits ( <i>this command is not necessary if you have installed the limits</i> )
> MC	Sets unit to the Continuous mode
> A2	Sets acceleration to 2 rps <sup>2</sup>
> V.1	Sets velocity to 0.1 rps
> G	Executes the move (Go)
> T1	Waits 1 second after the motor reaches constant velocity
> V.5	Sets velocity to 0.5 rps
> G	Changes velocity (Go)
> TR1Ø	Waits for trigger bit 1 to go on and bit 2 to go off
> VØ	Sets velocity to 0 rps
> G	Changes velocity to 0 rps
> MN	Changes mode to preset mode
> C	Instructs the unit to continue executing commands

This sequence sets the JSI to the Continuous mode. The motor reaches 0.1 rps, waits one second, changes velocity to 0.5 rps, waits for you to turn **IN 1** (Input 1) on and turn **IN 2** (Input 2) off, and then stops. Note that the commands **VØ** and **G** stop the motor (the **S** command is not a buffered command and cannot be used in this situation, unless you wish to halt the operation in the middle of the program).

**Incremental Mode Preset Moves**

When you are in the Incremental mode (**MPI**), a preset move moves the motor the specified distance from its starting position. You can specify the direction of the move in two ways. You can specify the direction by using the optional sign (D+8,000 or D-8,000), or you can define it separately with the Set Direction (**H**) command (H+ or H-).

Command	Description
> LD3	Disables the CW and CCW Limits ( <i>this command is not necessary if you have installed the limits</i> )
> MPI	Sets unit to Incremental Position Mode
> A2	Sets acceleration to 2 rps <sup>2</sup>
> V5	Sets velocity to 5 rps
> D8ØØØ	Sets distance to 8,000 steps (two revolutions)
> G	Executes the move (Go)
> G	Repeats the move (Go)
> H	Reverses direction of next move
> G	Executes the move (Go)

The motor moves two revolutions and stops. It then moves another two revolutions in the same direction and stops. The motor changes direction and moves two revolutions.

### Absolute Mode Preset Moves

A preset move in the absolute mode (**MPA**) moves the motor the distance that you specify (in encoder steps) from the absolute zero position. You can set the absolute position to zero with the Position Zero (**PZ**) command or by cycling the power to the controller. The absolute zero position is initially the power-up position.

The direction of an absolute preset move depends upon the motor position at the beginning of the move and the position you command it to move to. For example, if the motor is at absolute position +12,800, and you instruct the motor to move to position +5,000, the motor will move in the negative direction a distance of 7,800 steps to reach the absolute position of +5,000.

The JSI powers up in Incremental mode. When you issue the Mode Position Absolute (**MPA**) command, it sets the mode to absolute. When you issue the Mode Position Incremental (**MPI**) command the unit switches to Incremental mode. The JSI retains the absolute position, even while the unit is in the Incremental mode. You can use the Position Report (**PR**) command to read the absolute position.

In the following example, the motor makes its moves with respect to the absolute zero position. The motor makes a final move to return to the absolute zero position.

If you are using an absolute encoder, you must ensure that the encoder is operating within the valid range. If you move the encoder beyond the roll-over point, the JSI will lose track of its position. Refer to the manual that accompanied your absolute encoder for a more detailed discussion of roll-over points.

If you use absolute encoder feedback (**CFB2**) upon powering up the JSI controller, the JSI will read the actual position of the absolute encoder. You can use the **IDPA** command to determine the absolute encoder's position. In Absolute mode, all the moves you make will be with respect to this absolute position. If you use the Position Zero (**PZ**) command, the absolute position (**DPA**) does not change. The distance that you define for motor movement, however, will be measured from the point specified as zero.

<u>Command</u>	<u>Description</u>
> <b>MPA</b>	Sets unit to Absolute Position mode
> <b>A2</b>	Sets acceleration to 2 rps <sup>2</sup>
> <b>V10</b>	Sets velocity to 10 rps
> <b>PZ</b>	Sets the current position as zero position
> <b>D10000</b>	Sets distance to 10,000 steps
> <b>G</b>	Moves to absolute position 10,000 (Go)
> <b>D20000</b>	Sets distance to 20,000 steps
> <b>G</b>	Moves the motor to absolute position 20,000 (Go)
> <b>D0</b>	Sets the move distance to absolute position 0
> <b>G</b>	Moves to absolute position zero
> <b>MPI</b>	Sets controller to Incremental Position mode

The absolute positioning function described in this section refers to the controller's ability to retain the position of the motor while the JSI is powered up. However, once power to the JSI is lost, the motor position information is also lost.

## Application Considerations

### *Positional Accuracy vs. Repeatability*

This section contains information that you should consider and evaluate when designing and developing your system.

Some applications require high absolute accuracy. Others require repeatability. You should clearly define and distinguish these two concepts when you address the issue of system performance.

If the positioning system is taken to a fixed place and the coordinates of that point are recorded. The only concern is how well the system repeats when you command it to go back to the same point. For many systems, what is meant by accuracy is really repeatability. Repeatability measures how accurately you can repeat moves to the same position.

Accuracy on the other hand, is the error in finding a random position. For example, suppose the job is to measure the size of an object. The size of the object is determined by moving the positioning system to a point on the object and using the move distance required to get there as the measurement value. In this situation, basic system accuracy is important. The system accuracy must be better than the tolerance on the measurement that is desired.

Consult the technical data section of The Compumotor Catalog for more information on accuracy and repeatability.

### *Achieving Maximum Motor/Drive Performance with the JSI*

When you use the JSI controller with a DC motor or any other analog devices, you can expect to achieve the maximum velocity that is specified in the DC drive/motor user guide. Certain factors may prevent the motor from achieving its specified maximum velocity. The drive you are using is the main factor that may prevent your motor from achieving its maximum velocity. You must tune your drive with the load attached to the motor to get the best response from the drive. After achieving the peak performance from the drive, you can tune the JSI (PID tuning) to obtain the optimum performance from your entire system.

When you tune your drive and the JSI controller, consider the type of load that you are moving. The load will affect the system's response and settling time. For detailed information on tuning, refer to *Tuning the JSI* at the end of this chapter.

---

## Closed-Loop Operation

The JSI uses position feedback to position the load accurately. The drive, or the servo valve you are controlling must control the velocity feedback. To close the position loop of the JSI controller, you can use either an incremental encoder or a Compumotor absolute encoder. You can configure the JSI to accept different feedback devices with the Configure Feedback Device (**CFB**) command.

- **CFB1**: Indicates feedback device is incremental encoder
- **CFB2**: Indicates feedback device is AR23 absolute encoder with -4 decoder box
- **CFB3**: Indicates feedback device is AR-C, AL-C, or Photo-Trak absolute encoder
- **CFB4**: Indicates feedback device is AR23 absolute encoder with -1 decoder box

Once you define the feedback device and the resolution of the feedback device, the JSI controller will scan either the POSITION FEEDBACK input for absolute encoder feedback (**CFB2**, **CFB3**, or **CFB4**) or the ENCODER input for incremental encoder feedback (**CFB1**). You must turn off the JSI with the **OFF** command before changing the encoder type or resolution.

In incremental encoder mode (**CFB1**), the JSI can accept TTL Square Wave quadrature pulses in either a differential or single-ended fashion. Compumotor's incremental encoders are differential quadrature encoders (refer to Figure 3-4 in Chapter 3 for a wiring diagram of this encoder). You can also define encoder resolution with the **CFB1** command. The default setting is 4,000 steps per revolution. The maximum frequency of pulses that the JSI can read is 60 KHz before quadrature. Therefore, after quadrature, the maximum frequency the JSI can read is 240 KHz.

In the absolute encoder mode (**CFB2**, **CFB3**, or **CFB4**), the JSI reads the parallel data that the absolute encoder provides. Because it takes longer to process parallel positional information, the servo sample period for the PID loop will be slowed to 1,024 microseconds from 512 microseconds. However, the I/O functions are still performed every 512 microseconds. Refer to Chapter 3, *Installation*, for connection diagrams for the absolute encoders.

#### Incremental Encoder Example

This example sets up the JSI to receive feedback from the incremental encoder and make a closed loop move.

Command	Description
> LD3	Disables CW and CCW limits
> OFF	Shutdown JSI Output
> CFB1, 4000	Defines the feedback device as incremental encoder with resolution of 4,000 steps/rev after quadrature
> ON	Turn on JSI output
> A10	Sets acceleration at 10 rps <sup>2</sup>
> V5	Sets velocity at 5 rps
> D4000	Sets move distance to 1 revolution
> G	Executes the move (Go)

#### Absolute Encoder Example

Command	Description
> LD3	Disables CW and CCW limits
> OFF	Disables <b>COMMAND OUT</b>
> CFB2, 16384	Defines the feedback device as AR23 -4 absolute encoder with a resolution of 16,384 steps/rev after quadrature
> ON	Enables <b>COMMAND OUT</b>
> A10	Sets acceleration at 10 rps <sup>2</sup>
> V5	Sets velocity at 5 rps
> D16384	Sets move distance to 1 revolution
> G	Executes the move (Go)

#### Homing Function

You can use this function to establish a home reference position. You can instruct the JSI to move the motor to a home position with the Go Home (**GH**) command. This home position is typically established by mounting a load-activated switch to the home limit input so that the switch turns on when the load is in the desired position.

When you use an incremental encoder, the Z Channel or Index Channel, if any, may be used in conjunction with the home limit switch to establish the home position. The home position is located where the edge selected with the OS command of the Home Limit input occurs. (i.e., the controller recognizes the home position as the position where the home limit signal makes a transition from on to off, or from off to on, depending on the selected edge and the initial direction of the go home move.) Once it recognizes the selected edge, the motor decelerates to a stop. After coming to a stop, the JSI positions the motor away from the selected edge of home limit signal in the opposite direction of the initial direction of the go home move. After coming to a stop a second time, the JSI creeps the motor towards the selected edge at Go Home Final Velocity (**GHF**) until the home limit input becomes active again or the home limit and the encoder Z Channel pulse are active. The homing function is designed to accommodate encoder index signals that are typically of short duration. The function works best with this kind of home signal.

You must ensure that the final approach starts from the opposite side of the signal from the selected edge. If the final approach direction is positive, the final move must start from the negative side of the selected edge. If there is significant backlash and friction in the system, and the indexer is instructed to go home in the clockwise direction, the motor may end up on the wrong side of the signal and execute its final approach in the wrong direction.

This problem can also occur if the motor's go home speed is high, and the Home limit signal is delayed, (by a relay or Programmable Controller for example). In such a situation, you should initiate homing operations from the opposite side of the selected edge of home.

When you conclude the homing operation, the indexer resets its internal position counter. You can determine the true indexer position referenced to an encoder with the Report Absolute Encoder Position (**DPA**) command.

You can use the home limit input in conjunction with the encoder's Channel Z input to select a final home position. To activate homing Z bit, you must activate the **OSD1** command. In this situation, a load-activated switch connected to the Home Limit input locates the general home position area, and the indexer channel signal from the encoder is used for final Home positioning. The Channel Z and Home Enable inputs must both be active to mark the home position.

Under interface control, the Go Home (**GH**) command has the form **GH**<direction><velocity>. This indicates which direction to move, and at what velocity. For example, the command GH-2 sends the motor in the negative direction at 2 rps in search of the home signal. The acceleration parameter for this move is the home value for acceleration (**GHA** command). If an end-of-travel limit is activated before home is found, the controller reverses direction and attempts to find the home position again. If the other limit is activated before the controller finds home, the controller will stop trying to go Home. The controller can indicate whether or not the homing process was successful by responding to the Request Indexer Status (**R**) and Go Home Status (**RG**) commands.

<u>Command</u>	<u>Description</u>
> LD3	Disables Positive and Negative limit inputs
> OSB1	Back up to Home limit
> OSD1	Enables Go Home to Z bit
> GHA1	Sets go home acceleration to 1 rps <sup>2</sup>
> GE.5	Sets Go Home in the positive direction at 0.5 rps

The motor starts to move toward the home position. Upon encountering the home switch, the motor comes to a stop. The motor changes direction and searches for the home switch again. When it sees the switch, the motor looks for the Z bit (if **OSB1** and **OSD1** are enabled) to turn on after the home limit switch is closed. In encoder mode, home limit switch and Z bit must be on at the same time for the motor to Go Home successfully.

## Programmable Inputs

You can use the Trigger Pause (**TR**) command to cause a sequence of buffered commands to wait for one or more inputs to come to a preferred state. Inputs **IN 1 - IN 13** are set at the factory (default setting) to function as programmable inputs.

You may also configure programmable inputs to perform other functions. Refer to the Configure Input Mode (**IN**) command for a description of the other functions that are available.

Command	Description
> <b>IN1C</b>	Sets up <b>IN 1</b> as Kill (K) input
> <b>IN2G</b>	Sets up <b>IN 2</b> as Go (G) input.
> <b>IN3A</b>	Sets up <b>IN 3</b> as trigger input 1
> <b>IN4A</b>	Sets up <b>IN 4</b> as trigger input 2
> <b>MN</b>	Sets the unit to Preset mode
> <b>A2</b>	Sets acceleration to 2 rps <sup>2</sup>
> <b>V5</b>	Sets velocity to 5 rps
> <b>D25000</b>	Sets distance to 25,000 steps
> <b>TR01</b>	Waits for trigger input 1 ( <b>IN 3</b> ) to be off and trigger input 1 ( <b>IN 4</b> ) to be off
> <b>G</b>	Executes 25,000-step move (Go)

This example program configures **IN 1** as a Kill input and **IN 2** as a Start input. **IN 3** and **IN 4** are set up as programmable trigger inputs. If you activate the Start input (**IN 2**), the motor will move 25,000 steps. The rest of the program waits for the trigger inputs to match before executing the move. At any time during the operation of the JSI, if you activate **IN 1**, the controller immediately stops the operation and clears the command buffer.

## Delays

You can use the Time (**T**) command to halt the operation of the indexer function for a preset time. If you are in the Continuous mode, you may use the Time (**T**) command to run the motor at continuous velocity for a set time, then change to a different velocity.

In the Preset mode, the motor finishes the move before the indexer executes the time delay.

Command	Description
> <b>PS</b>	Waits for the controller to receive a Continue ( <b>C</b> ) command before executing the next command
> <b>G</b>	Moves motor 25,000 steps
> <b>T5</b>	Waits 5 seconds after the move ends
> <b>H</b>	Changes motor direction
> <b>G</b>	Moves motor 25,000 steps in the opposite direction
> <b>C</b>	Continues execution

## Branching

You can perform conditional branching with the Compare Error Flag (**IFER**), Compare User Flag (**IFFL**), and Compare Input Status (**IFIN**) commands. All three of these commands are very similar to IF\_THEN statement in BASIC programming. If the condition matches, the JSI will perform the commands that immediately follow the **IF** command. If the condition does not match, the JSI will skip all of the commands that follow the **IF** command until it reaches the End of IF Statement (**NIF**) command.

**IFER Command**

This command checks to see if any error condition exists to execute a conditional command. This command is useful if you wish to trap different error conditions (Drive Disabled, User Fault Input Activated, Excessive Position Error, etc). For a detailed description of this command, refer to Chapter 5, *Software Reference*.

<u>Command</u>	<u>Description</u>
> XD1Ø	Defines Sequence 10. When a fault occurs, this sequence 10 will execute. The sequence is defined with the Set Fault or Kill Sequence (XFK1Ø) command.
> IFER1	If hardware CCW LIMIT switch is reached, perform the following commands:
> A1Ø	Sets acceleration to 10 rps <sup>2</sup>
> V5	Sets velocity to 5 rps
> D1ØØØØ	Sets distance to 10,000 steps
> G	Executes the move (Go)
> NIF	Ends IF statement
> IFERX1	If hardware CW LIMIT switch is reached, perform the following commands:
> A1Ø	Sets acceleration to 10 rps <sup>2</sup>
> V5	Sets velocity to 5 rps
> D-1ØØØØ	Sets distance to 10,000 steps in the opposite direction
> G	Executes the move (Go)
> NIF	Ends IF statement
> IFERXXXXXX1	If Drive Enable (ENBL+) input is opened, perform the following commands:
> 1" CONNECT_	Writes to the RS-232C port to inform the user that drive enable is disconnected.
> 1"DRIVE_	
> 1"ENABLE_	
> 1"INPUT_	
> NIF	Ends IF statement
> XT1Ø	Ends Sequence loop
> XFK	Sets Sequence 10 as the Fault sequence.

**IFFL Command** This command uses the pattern set by the User Flag (SFL) command to execute the conditional commands. This command is useful if you wish to make a decision based on previous sequence executions that will set or clear the user flag bits. For example, in an application with several sequences, at the end of each sequence, you can assign different bit patterns with the SFL command. If you select these sequences from the host computer, you may wish to make different moves depending on the sequence you ran. For a detailed description of this command, refer to Chapter 5, *Software Reference*.

<u>Command</u>	<u>Description</u>
> PS	Waits for the controller to receive a Continue (C) command before executing the next command
> SFL1Ø1Ø	Sets user flag bits 7 and 5 and clears bits 6 and 4, the remaining bits are not altered
> IFFL1Ø1Ø	If user flag bits 5 and 7 are set, and bits 6 and 4 are clear, perform the following commands
> A1Ø	Sets acceleration to 10 rps <sup>2</sup>
> V5	Sets velocity to 5 rps
> D25ØØØ	Sets distance to 25,000 steps
> G	Executes the move (Go)
> NIF	Ends IF statement
> C	Continues execution

The **IFFL** pattern matches the **SFL** setting. The motor moves 25,000 steps.

**IFIN Command** This command compares the input pattern (**CW**, **CCW**, and **HOME LIMIT** outputs and **IN 1 - IN 13** inputs) to execute the conditional commands. This command is useful for branching and performing conditional moves using the programmable inputs. For a detailed description of this command, refer to Chapter 5, *Software Reference*.

<u>Command</u>	<u>Description</u>
> <b>IFINXXX10</b>	If <b>IN 1</b> is active and <b>IN 2</b> is not active, issue the following commands:
> <b>A10</b>	Sets acceleration to 10 rps <sup>2</sup>
> <b>V5</b>	Sets velocity to 5 rps
> <b>D25000</b>	Sets distance to 25,000 steps
> <b>G</b>	Executes the move (Go)
> <b>NIF</b>	Ends IF statement
> <b>IFINXXX01</b>	If <b>IN 1</b> is not active (open) and <b>IN 2</b> is active (closed), issue the following commands:
> <b>A10</b>	Sets acceleration to 10 rps <sup>2</sup>
> <b>V5</b>	Sets velocity to 5 rps
> <b>D5000</b>	Sets distance to 5,000 steps in the opposite direction
> <b>G</b>	Executes the move (Go)
> <b>NIF</b>	Ends IF statement
> <b>IFINXXX1</b>	If <b>IN 1</b> is active, do the following command.
> <b>1"DONE</b>	Ends message saying done
> <b>NIF</b>	Ends IF statement

### Subroutines

When you use the Goto Sequence (**XG**) and the Execute a Sequence (**XR**) command sequences, you can execute different sequences from within a sequence. These commands are similar to GOTO and GOSUB commands in BASIC programming. If you use an **XG** command, the program will move to the sequence that you specified in the **XG** command. After executing the specified sequence, the system will not return to the original sequence. It will remain in the current sequence, unless it receives another execution command (**XG** or **XR**). However, if you use the **XR** command, the program will return control to the original sequence that contained the **XR** command. Program control will return to the original sequence when a Terminate Sequence (**XT**) command is reached in a sequence. This prompts the program to return to the sequence that initiated the move to another sequence.

You can nest as many as 16 different levels of sequences within one program. For example, when you exit Sequence 1 to execute Sequence 2 with the **XR2** command, you can execute to Sequence 3 from Sequence 2. This procedure of nesting **XR** commands can be repeated 16 times. The **XG** command has no limit since the program will not return control to the original sequence.

<u>Command</u>	<u>Description</u>
> <b>XE2</b>	Erases Sequence 2
> <b>XD2</b>	Defines Sequence 2
<b>A10</b>	Sets acceleration to 10 rps <sup>2</sup>
<b>V5</b>	Sets velocity to 10 rps
<b>D2000</b>	Sets distance to 2,000 steps
<b>G</b>	Executes the move (Go)
> <b>XT</b>	Ends Sequence 2 definition
> <b>XE3</b>	Erases Sequence 3
> <b>XD3</b>	Defines Sequence 3
<b>A10</b>	Sets acceleration to 10 rps <sup>2</sup>
<b>V5</b>	Sets velocity to 10 rps
<b>D-2000</b>	Sets distance to 2,000 steps
<b>G</b>	Executes the move (Go)
> <b>XT</b>	Ends Sequence 3 definition
> <b>XE1</b>	Erases Sequence 1
> <b>XD1</b>	Defines Sequence 1
<b>XR2</b>	Executes Sequence 2
<b>XR3</b>	Executes Sequence 3
> <b>XT</b>	Ends Sequence 1 definition
> <b>XR1</b>	Executes Sequence 1

In the previous example, when you execute Sequence 1, the program moves to Sequence 2. After executing Sequence 2, the program returns to Sequence 1. The program then moves to execute Sequence 3.

Command	Description
> XE1	Erases sequence 1
> XD1	Defines sequence
IFINXXX1	If IN 1 is active (Closed)
XG2	Executes Sequence 2
NIF	Ends If statement
A1	Sets acceleration to 1 rps <sup>2</sup>
V5	Sets velocity to 5 rps
D25000	Sets distance to 25,000 steps
G	Executes move.
> XT	Ends Sequence 1 definition
> XR1	Executes Sequence 1

In the following example, when you execute Sequence 1, the program checks the input pattern. If IN 1 is on, the program moves to execute Sequence 2. After executing sequence 2, program does not return to Sequence 1. If IN 1 is off, the program ignores the XG2 command and makes the 25,000-step move.

**Loops**

You may use the Loop (L) command to repeat certain programs. You can nest Loop commands up to 16 levels deep.

Command	Description
> PS	Pauses command execution until the controller receives a Continue (C) command
> MPI	Sets unit to Incremental mode
> A5	Sets acceleration to 5 rps <sup>2</sup>
> V5	Sets velocity to 5 rps
> L5	Loops 5 times
D2000	Sets distance to 2,000 steps
G	Executes the move (Go)
T2	Delays 2 seconds after the move
> N	Ends loop
> C	Initiates command execution to resume

The motor moves a total of 10,000 steps

The example below shows how you can nest a small loop inside a major loop. In this example, the motor makes 2 moves and returns a line feed. The unit repeats these procedures and will continue to repeated until you instruct the unit to stop.

Command	Description
> PS	Pauses command execution
> L	Loops indefinitely
> 1LF	Sends a line feed
> L2	Loops twice
> G	Executes 2,000-sstep move
> T.5	Waits 0.5 seconds
> N	Ends loop
> N	Ends loop
> C	Continues command execution

This command execution continues until you issue the Stop (S) or Kill (K) commands.

### **POBs (Programmable Output Bits)**

You can turn the programmable outputs (**OUT 1 - OUT 8**) on and off with the Output (**O**) and Immediate Output (**IO**) commands. Outputs **OUT 1** through **OUT 8** are factory set as programmable outputs. However, you can configure all of the outputs to perform different functions (Moving/Not Moving, Amp Off, Strobe, etc.) with the Configure Output(**OUT**) command. Refer to the **OUT** command in the Chapter 5, Software Reference for descriptions of the available functions. You can use these outputs to turn on and off other devices (i.e., lights, switches, etc.).

<u>Command</u>	<u>Description</u>
> PS	Pauses command execution until JSI receives Continue ( <b>C</b> ) command
> A10	Sets acceleration to 10 rps <sup>2</sup>
> V5	Sets velocity to 5 rps
> D25000	Sets distance to 25,000 steps
> OUT1A	Sets <b>OUT 1</b> as a programmable output
> OUT2A	Sets <b>OUT 2</b> as a programmable output
> OUT3B	Sets <b>OUT 3</b> as a Moving/Not Moving output
> O10	Turns <b>OUT 1</b> on and <b>OUT 2</b> off
> C	Executes the move
> O01	Turn <b>OUT 1</b> off and <b>OUT 1</b> on
> C	Initiates command execution to resume

This example above defines **OUT1** and **OUT2** as programmable outputs and **OUT3** as a Moving/Not Moving output. Before the motor moves 25,000 steps, **OUT1** is turned on and **OUT2** is turned off. These outputs will remain in this state until the move is completed, then **OUT1** will turn off and **OUT2** will be turned on. While the motor is moving, **OUT3** remains on.

### **Move Completion Signal**

When you complete a move, you may use the JSI's programming capability to signal the end of the current move. In a preset move, you may use one of the following commands:

- **LF** Line feed
- **CR** Carriage return
- **O** Output command
- **"** Quote command

<u>Example# 1</u>	<u>Command</u>	<u>Description</u>
	> A2	Sets acceleration to 2 rps <sup>2</sup>
	> V.5	Sets velocity to 0.5 rps
	> D12500	Sets distance to 12,500 steps
	> G	Executes the move (Go)
	> 1LF	Sends a line feed over the RS-232C interface

The motor moves 12,500 steps. When you complete the move, the unit issues a line feed from the JSI to the host over the RS-232C interface.

<u>Example #2</u>	<u>Command</u>	<u>Description</u>
	> A2	Sets acceleration to 2 rps <sup>2</sup>
	> V.5	Sets velocity to 0.5 rps
	> D12500	Sets distance to 12,500 steps
	> G	Executes the move (Go)
	> 1CR	Sends a carriage return

The motor moves 12,500 steps. When the JSI completes the move, the unit issues a carriage return from the JSI to the host over the RS-232C interface.

Example #3	Command	Description
	> OUT1A	Sets Output 1 (OUT 1) as a programmable output
	> A2	Sets acceleration to 2 rps <sup>2</sup>
	> V.5	Sets velocity to 0.5 rps
	> D12500	Sets distance to 12,500 steps
	> G	Executes the move (Go)
	> O1	Turns on Output 1

The motor moves 12,500 steps. When the move is completed, Output 1 is turned on.

Example #4	Command	Description
	> A2	Sets acceleration to 2 rps <sup>2</sup>
	> V.5	Sets velocity to 0.5 rps
	> D12500	Sets distance to 12,500 steps
	> G	Executes the move (Go)
	> 1"DONE	Sets the Quote DONE message

The motor moves 12,500 steps. When you complete the move, the unit issues the DONE message from the JSI to the host over the RS-232C interface.

### Sequence Commands

Use the following commands to define, erase, and run sequences. Refer to Chapter 5, *Software Reference*, for detailed descriptions and syntax of the following commands.

Command	Description
> XBS	Reports the number of bytes available for sequence programming
> XD	Starts sequence definition
> XE	Deletes sequence from EEPROM
> XQ	Sets/resets interrupted Run mode
> XRP	Runs a sequence with a pause
> XT	Ends sequence definition
> XU	Uploads sequence
> XR	Runs a sequence
> SSJ1	Runs a sequence defined by BCD sequence inputs
> XG	Exits current sequence and moves to execute another sequence

A sequence is a series of commands. These commands are executed in order whenever the sequence is run. Immediate commands cannot be stored in a sequence, just as they cannot be stored in the command buffer. Only buffered commands may be used in a sequence.

The JSI has 8,000 bytes of non-volatile memory to store 100 sequences. You can use the **XBS** command to determine how many bytes are available in the sequence buffer. The sequence buffers may have variable lengths, so you may have one long sequence or several short ones, as long as the total length does not exceed the 8,000 bytes of allocated space.

*The commands that you enter to define a sequence are presented vertically in the examples below. This was done to help you read and understand the commands. When you are actually typing these commands into your terminal, they will be displayed horizontally.*

To begin the definition of a sequence, enter the Define Sequence (**XD**) command immediately followed by sequence identifier number (1 to 100) and a delimiter. The Terminate Sequence (**XT**) command ends the sequence definition. All commands that you enter after the **XD** command and before the **XT** command will be executed when the sequence is run. An example is provided below.

<u>Command</u>	<u>Description</u>
> <b>XE1</b>	Erases Sequence 1
> <b>XD1</b>	Begins definition of Sequence 1
<b>A2</b>	Sets acceleration to 2 rps <sup>2</sup>
<b>V10</b>	Sets velocity to 10 rps
<b>D5000</b>	Sets distance to 5000 steps
<b>G</b>	Executes the move (Go)
<b>H</b>	Reverses direction
<b>G</b>	Executes the move (Go)
<b>XT</b>	Ends definition of sequence
> <b>XR1</b>	Runs Sequence 1

You can run a sequence by entering the **XR** command immediately followed by a sequence identifier number (1 to 100) and a delimiter. If you have enabled the Continuous Sequence Scan mode (**SSJ1**), you can also execute a sequence by switching the Sequence Select inputs, after you define the inputs as sequence-select inputs with the Configure Inputs (**IN**) command.

Once you define a sequence, it cannot be redefined until you delete it. You can delete a sequence by entering the **XE** command immediately followed by a sequence identifier (1 to 100) and a delimiter. You may then redefine that sequence.

Sequence 100 (if you have defined it) is always run when you power up the system or when you reset the indexer with the Reset (**Z**) command. For convenience, you may find it advantageous to place all of your set-up commands in Sequence 100.

Sequences that you define are automatically saved into the JSI's non-volatile memory. The only way to erase these sequences is by using the Erase Sequence (**XE**) command.

### Selecting Sequences

After you define the sequences from the RS-232C interface, you can execute the sequences by using one of the following modes of operation:

- Stand-alone: Use thumbwheel switches to select and run the sequence.
- Computer Interface: Use the Run Sequence (**XR**) command to run the sequences.
- PLC (Programmable Logic Controller): Use the sequence select inputs to run a sequence.

### Stand-alone Operation

This section explains and provides examples of how to store programs and run them with remote switches, and run them automatically when you power up the system. First, you will need to enter the programs into the JSI. You will need a terminal or a computer with RS-232C communication capabilities for programming the JSI controller.

**Power-Up Sequence Execution**

You can program the JSI to execute a sequence of commands on power up (sequences can be used as subroutines).

Sequence 100 always runs on power up. To run another sequence on power up, put an **XR<num>** (or **XG<num>**) at the end of sequence 100. If sequence 100 is empty, nothing happens on power up. Refer to Chapter 5, Software Reference, for Detailed descriptions and syntax of the following commands.

<u>Command</u>	<u>Description</u>
> <b>XE100</b>	Erases Sequence 100
> <b>XD100</b>	Begins definition of sequence 100
<b>LD3</b>	Disables limits if they are not connected
<b>A2</b>	Sets acceleration to 2 rps <sup>2</sup>
<b>V5</b>	Sets velocity to 5 rps
<b>D12500</b>	Sets distance to 12,500 steps
<b>G</b>	Executes the move (Go)
<b>XT</b>	Ends sequence definition
> <b>Z</b>	Resets the controller and runs Sequence 100

A power-up sequence is typically used to store set-up or initialization parameters that your application requires. Some of these commands are listed below.

<u>Command</u>	<u>Description</u>
> <b>SSJ1</b>	Continuous Sequence Scan Mode
> <b>SN</b>	Scan time
> <b>JA</b>	Jog acceleration
> <b>JVL</b>	Jog velocity low
> <b>JVH</b>	Jog velocity high

You can put any commands that are not immediate (buffered) into Sequence 100 (if you want to executed them during power up).

**Using Remote Jogs and Sequence**

In some applications, you may want to move the motor manually. You can configure the JSI to allow you move the motor manually with the Configure Input (**IN**) command. You must define the jogging velocity with the Jog Velocity High (**JVH**) and Jog Velocity Low (**JVL**). You can define three different inputs for jogging: CW Jog input (**IN#J**), CCW Jog Input (**IN#K**), and Jog Speed Select High/Low (**IN#L**). You must also enable the jogging feature with the **OSE1** command. Once you set up all of these parameters, you can attach a switch to the jog inputs that you defined and perform jogging. (# represents digits 1 - 13, which you enter.)

The following example shows how you can define power-up Sequence 100 to set up jogging.

**Step 1**

Define a power up sequence.

<u>Command</u>	<u>Description</u>
> <b>XE100</b>	Erase sequence 100
> <b>XD100</b>	Define sequence 100
<b>JA2</b>	Set Jog Acceleration to 2 rps <sup>2</sup>
<b>OSE1</b>	Enables Jog function
<b>JVL.1</b>	Sets low-speed jog velocity to 0.1 rps
<b>JVH5</b>	Sets high-speed jog velocity to 5 rps
<b>IN1J</b>	Sets <b>IN 1</b> as a CW jog input
<b>IN2K</b>	Sets <b>IN 2</b> as a CCW jog input
<b>IN3L</b>	Sets <b>IN 3</b> as a speed-select input
<b>IN4B</b>	Sets <b>IN 4</b> as a sequence-select input
<b>IN5B</b>	Sets <b>IN 5</b> as a sequence-select input
> <b>XT</b>	Ends Sequence Definition

**Step 2** Define all sequences that your application may require.

<u>Command</u>	<u>Description</u>
> XE1	Erases Sequence 1
> XD1	Defines Sequence 1
A1	Sets acceleration to 1 rps <sup>2</sup>
V2	Sets velocity to 2 rps
D1000	Sets distance to 1,000 steps CW
G	Executes the move (Go)
> XT	Ends sequence definition

<u>Command</u>	<u>Description</u>
> XE2	Erase Sequence 2
> XD2	Defines Sequence 2
A1	Sets acceleration to 1 rps <sup>2</sup>
V2	Sets velocity to 2 rps
D-1000	Sets distance to 1,000 steps CCW
G	Executes the move (Go)
> XT	Ends sequence definition

**Step 3** Reset the JSI controller

<u>Command</u>	<u>Description</u>
> Z	Resets the JSI controller

**Step 4** Turn on the **IN 1** input to move the motor in the CW direction at 0.1 rps (until you turn off **IN 1** ).

**Step 5** Turn on the **IN 2** input to move the motor in the CCW direction at 0.1 rps (until you turn off **IN 2**).

**Step 6** Turn on the **IN 3** input to switch to high-speed jogging.

**Step 7** Repeat steps 4 and 5 to perform high-speed jogging.

**Step 7** Enter the **XR1** command to run the motor 1,000 steps CW.

**Step 7** Enter the **XR2** command to run the motor 1,000 steps CCW.

#### Selecting Sequences with Thumbwheels

The following example shows how the JSI is typically used with remote thumbwheels. In this example only 3 sequences are entered. As many as 100 sequences may be defined and up to 99 may be executed using remote thumbwheels. Sequence 100 is automatically executed during power up or reset (refer to the Reset (**Z**) command).

You must install your thumbwheels as shown in Figure 4-2.

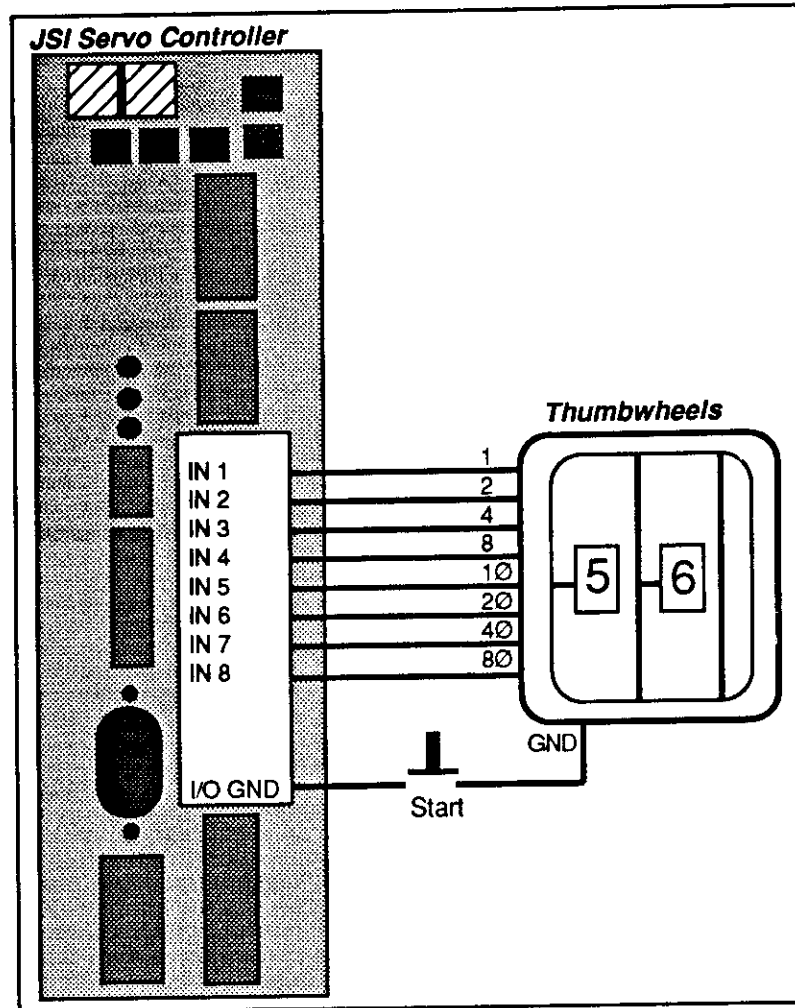


Figure 4-2. JSI Thumbwheel Installation

**Step 1** Define a power-up sequence. Set up inputs **IN 1 - IN 8** as sequence-select inputs using the Configure Input (**IN**) commands. The **IN 1** command will be the least significant bit of the 1's (ones) digit. The **IN 8** command will be the most significant digit of the 10's (tens) digit. Refer to Table 4-1.

Command	Description
> XE100	Erases sequence 100
> XD100	Defines sequence 100
IN1B	Sets up Input 1 as the sequence input
IN2B	Sets up Input 2 as the sequence input
IN3B	Sets up Input 3 as the sequence input
IN4B	Sets up Input 4 as the sequence input
IN5B	Sets up Input 5 as the sequence input
IN6B	Sets up Input 6 as the sequence input
IN7B	Sets up Input 7 as the sequence input
IN8B	Sets up Input 8 as the sequence input
INL0	Sets active input level to active low (ON = 0V applied to the input)
SSJ1	Set JSI to execute programs from remote interface
SN100	Set scan time to 100 msec
XQ1	Enable sequence hold
> XT	Ends sequence definition

The BCD values of inputs **IN 1 - IN 8** are shown in Table 4-1.

Inputs	IN8	IN7	IN6	IN5	IN4	IN3	IN2	IN1
BCD Value	80	40	20	10	8	4	2	1
	10's Digit				1's Digit			

Table 4-1. Input BCD Values

**Step 2** Define any any sequences that your application may need.

<u>Command</u>	<u>Description</u>
> XE1	Erases Sequence 1
> XD1	Starts Sequence 1 definition
MN	Sets mode to normal
A2	Sets acceleration to 2 rps
V5	Sets velocity to 5 rps
D25000	Sets distance to 25,000 steps
G	Executes the move (Go)
> XT	Ends sequence definition

<u>Command</u>	<u>Description</u>
> XE5	Erases Sequence 5
> XD5	Starts Sequence 5 definition
MN	Sets mode to normal
A2	Sets acceleration to 2 rps <sup>2</sup>
V5	Sets velocity to 5 rps
D10000	Sets distance to 10,000 steps
G	Executes the move (Go)
> XT	Ends sequence definition

<u>Command</u>	<u>Description</u>
> XE99	Erases Sequence 99
> XD99	Starts Sequence 99 definition
MN	Sets mode to normal
A2	Sets acceleration to 2 rps <sup>2</sup>
V5	Sets velocity to 5 rps
D-35000	Sets distance to -35,000 steps
G	Executes the move (Go)
> XT	Ends sequence definition

**Step 3** Connect the thumbwheels to the **IN 1 - IN 8** inputs as shown in Figure 4-2.

**Step 4** Reset the JSI controller. This will prepare the power up (Sequence 100) to be executed when you apply power to the controller.

<u>Command</u>	<u>Description</u>
> Z	Resets the JSI controller

**Step 5** Set your thumbwheel to 1 and push start to move the motor 25,000 steps in the positive direction.

**Step 6** Set your thumbwheel to 5 and push start to move the motor 10,000 steps in the positive direction.

**Step 7** Set your thumbwheel to 99 and push start to move the motor 35,000 steps in the negative direction.

*Note: If you select (with the thumbwheel) an invalid or unprogrammed sequence, nothing will happen.*



### Multi-Axis Control (Daisy-Chaining)

You may daisy chain up to 16 JSI controllers. Individual drive addresses are set with the JSI's push buttons. When daisy-chained, the units may be addressed individually or simultaneously. You should establish a unique device address for each JSI. Refer to Figure 4-4 for JSI daisy chain wiring information.

Commands prefixed with a device address command only the unit specified. Commands without a device address command all units on the daisy-chain. The general rule is: Any command that causes the drive to transmit information from the RS-232C port (such as a status or report command), must be prefixed with a device address. This prevents daisy chained units from all transmitting at the same time.

Attach device identifiers to the front of the command. The Go (G) Command instructs all units on the daisy chain to go, while 1G tells only unit one to go.

When you use a single communications port to control more than one JSI, all units in a daisy chain receive and echo the same commands. Each device executes these commands, unless this command is preceded with an address that differs from the units on the daisy chain. This becomes critical if you instruct any indexer to transmit information. To prevent all of the units on the line from responding to a command, you must precede the command with the device address of the designated unit.

No JSI executes a device-specific command unless the unit number specified with the command matches the JSI's unit number. Device-specific commands include both buffered and immediate commands.

You must use status-request commands in an orderly fashion. Commands should only be issued when the host is ready to read the response. You should not send new commands until you receive a response from the previous status-request command. In particular, you should not issue a immediate-status command until the host receives a buffered command status response. If this is not followed, the command responses will be intertwined, rendering the information useless.

If you enable the Interactive mode (SSI1), only the JSI that is set to Address 1 will respond with the >. This prevents all the JSIs from sending out > in a daisy chain. Typically, you should disable the Interactive mode when you use a host computer with the JSI controller.

### JSI Daisy-Chain Wiring

Figure 4-3 illustrates how JSIs might be wired in a daisy-chain configuration.

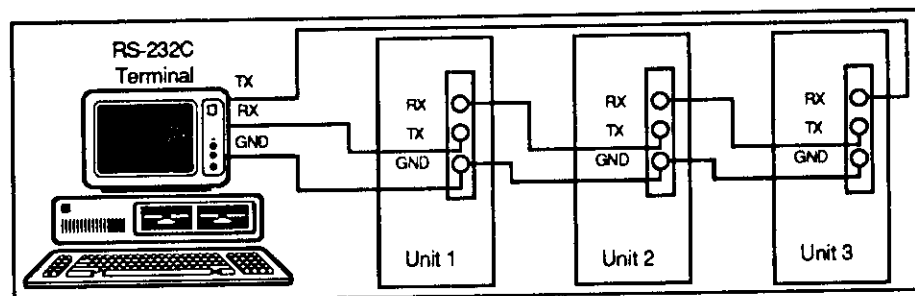


Figure 4-3. Daisy-Chained JSIs

**Sample Applications and Commands**

Example: Three indexers are on an RS-232C daisy-chain. Send the following commands:

<u>Command</u>	<u>Description</u>
> M0	Sets unit to Preset mode
> A5	Sets acceleration to 5 rps <sup>2</sup> for all three controllers
> V10	Sets velocity to 5 rps for all three controllers
> LD3	Disables limits (if they are not connected)
> 1D25000	Sets Axis 1 distance to 25,000 steps
> 2D50000	Sets Axis 2 distance to 50,000 steps
> 3D100000	Sets Axis 3 distance to 100,000 steps
> G	Moves all axes.

Unit 1 moves 25,000 steps, unit 2 moves 50,000 steps, and unit 3 moves 100,000 steps. All three units use the same acceleration and velocity rates. Units 1 and 2 move at about the same time.

**Multi-Axis Interface Program Example**

The following program is very similar to JSI.BAS, except this program controls two JSIs on a daisy-chain. This program assumes the device address of two JSIs to be #1 and #2 respectively. The program does the following.

1. Executes the first move upon user input
2. Waits for a line feed from the JSI controller, which indicates the end of the move.
3. Executes the second move upon user input.
4. Waits for a line feed from the JSI controller, which indicates the end of the move. It then begins the process again.



**PLC Operation**

You can use a PLC to execute 99 different sequences that are stored in the JSI controller. You can configure up to eight inputs as sequence-select inputs. This capability allows you to use eight outputs from the PLC to select as many as 99 sequences. You can accomplish this by changing the BCD values of the PLC outputs.

**Scanning for Sequence Execution**

Changing the BCD values of sequence input lines results in a new sequence being run that corresponds to the new value. The sum of the values issued determines which sequence the indexer will run. For example, if you set up inputs 1 - 8 as sequence-select inputs, the lowest input (Input 1) will have the least significant value and the highest input (Input 8) will have the most significant value. Refer to Table 4-1.

You must use the Input (IN) command to set up the inputs as sequence-select inputs. You must also configure the JSI controller to read sequences via BCD input (use the **SSJ1** command).

After you issue the **SSJ1** command, the JSI controller scans the sequence select inputs to find the sequence that you have specified for execution. After the system completes the execution of the desired sequence, the controller scans the sequence inputs again to find the next sequence to be executed. If you use the Interrupted Run Mode (**XQ1**) command. In the Interactive Run mode (**XQ1**), the JSI controller waits until all of the sequence inputs are turned off before selecting the next sequence to execute. Use the Input Level (**INL**) command to set up the controller's active level.

The Scan (**SN**) command determines how long the sequence-select input must be maintained before the controller executes the program. This is a debounce time.

**Sample Applications and Commands**

This section provides step-by-step procedures to run sequences from your PLC. First, you need to enter the programs into the JSI. You will need a terminal or a computer with RS-232C communication capability. You need to define the sequences before you can execute them with your PLC's BCD outputs. The JSI controller automatically saves these sequences (battery-backed-up system).

Using a terminal or a computer, key in the following commands. *The commands that you enter to define a sequence are presented vertically in the example below. This was done to help you read and understand the commands. When you are actually typing these commands into your terminal, they will be displayed horizontally.*

**Step 1** Define a power-up sequence. *Every time you power up the JSI controller, it executes these commands and enables the JSI to read up to 99 sequences from the sequence-select inputs.*

<u>Command</u>	<u>Definition</u>
> XE100	Erases sequence 100
> XD100	Defines sequence 100
SSJ1	Executes sequences via PLC input
SN20	Sets scan time to 20 msec
XQ1	Sets JSI to interrupted run mode
A10	Sets acceleration to 10 rps <sup>2</sup>
V2	Sets velocity to 2 rps
IN1B	Sets up Input 1 as a sequence-select input
IN2B	Sets up Input 2 as a sequence-select input
IN3B	Sets up Input 3 as a sequence-select input
IN4B	Sets up Input 4 as a sequence-select input
IN5B	Sets up Input 5 as a sequence-select input
IN6B	Sets up Input 6 as a sequence-select input
IN7B	Sets up Input 7 as a sequence-select input
IN8B	Sets up Input 8 as a sequence-select input
OUT1C	Sets up Output 1 as sequence-in-progress output
LD3	Disables the limits (if they are not connected)
> XT	Ends the sequence definition

**Step 2** Define any sequences that your application may require.

<u>Command</u>	<u>Description</u>
> XE1	Erases Sequence 1
> XD1	Defines Sequence 1
D2000	Sets distance to 2,000 steps
G	Executes the move (Go)
> XT	Ends Sequence 1 definition

<u>Command</u>	<u>Description</u>
> XE2	Erases Sequence 2
> XD2	Defines Sequence 2
D4000	Sets distance to 4,000 steps
G	Executes the move (Go)
> XT	Ends Sequence 2 definition

<u>Command</u>	<u>Description</u>
> XE3	Erases Sequence 3
> XD3	Defines Sequence 3
D8000	Sets distance to 8,000 steps
G	Executes the move (Go)
> XT	Ends Sequence 1 definition

<u>Command</u>	<u>Description</u>
> XE99	Erases Sequence 99
> XD99	Defines Sequence 99
D-14000	Sets distance to -14,000 steps
G	Executes the move (Go)
> XT	Ends Sequence 99 definition

**Step 3** Verify that your programs were stored properly by uploading each entered sequence (**XU**). If you receive responses that differ from what you programmed, re-enter those sequences.

**Step 4** Run each program from the RS-232C interface with the Run Sequence (**XR**) command. Make sure that the motor moves the distance that you specify.

**Step 5** Assuming your PLC accepts open-collector outputs connect the inputs and outputs as shown in Figure 4-4. If not, you will need to add pull up resistors to the outputs.

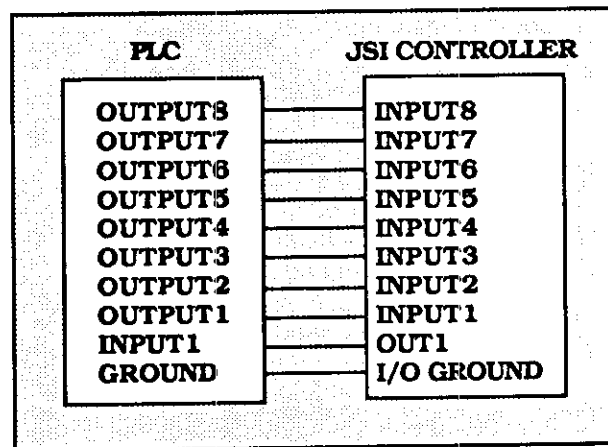


Figure 4-4. PLC Connection

**Step 6** Refer to the user guide that accompanied your PLC unit to turn on the proper combination of outputs to execute one of the four sequences programmed and stored in the JSI controller. You should program the PLC to search for sequence completion output from the JSI controller before the controller begins searching for the next sequence to execute (this type of interaction is performed through a *handshake* between the JSI and the PLC—the signal transmitted indicates that another sequence is ready to be executed).

- Turning on Output 1 only will execute Sequence 1.
- Turning on only the PLC's Output 2 will execute Sequence 2.
- Turning on only the PLC's Outputs 1 and 2 will execute Sequence 3.
- Turning on only the PLC's Outputs 1, 4, 5, and 8 will execute Sequence 99.

**Step 7** Cycle down power to the JSI. The system will execute Sequence 100.

**Step 8** Turn on the appropriate sequence-select input [set for the least Scan Time(SN)] to execute the proper sequences. Since the sequence hold feature (refer to the **XQ1** command) was enabled during the power-up sequence, your PLC program must turn off all of the sequence-select inputs before you can select another sequence.

**Tuning The JSI** Before you begin to tune the JSI, you should ensure that the analog servo valve or drive will exceed the maximum speed desired when the maximum analog output from the JSI is applied. The JSI can command  $\pm 10V$  or  $\pm 200mA$ . If this is not enough to exceed the maximum speed desired, the system will not be able to provide the desired performance.

The most important aspect of a servo system is setting the controller's gains (described in the previous sections). The controller in a conventional continuous-time (i.e., analog) servo system is an active or passive filter network, usually an op-amp with resistive and capacitive feedback. The values and configuration of the resistors and capacitors in the filter network are determined by the compensation necessary to stabilize the servo system.

The values of the resistors and capacitors can be thought of as being the gains of the controller. To stabilize the servo system, you must adjust the gains (potentiometers) to match the system being controlled (the system being the motor, its load, and the amplifier). In the case of the JSI, the gains of the controller are the constant coefficients of the recursive equation (software). The form of the recursive equation determines how many of these gains must be adjusted in order to stabilize the system.

Fortunately, there are only three user-changeable values in the equation and, better yet, it is not necessary to know the equation to tune a JSI servo system. Tuning a JSI servo system usually requires adjusting only one controller gain. The other gains are predefined and, in most cases, require no further adjustment.

There are two methods available to adjust a JSI's servo compensation network.

- The five pushbuttons on the JSI's front panel
- The RS-232C communications port

### PushButton Tuning

The JSI has five pushbuttons on the front panel that provide a simple push-button method of fine-tuning the system's performance to a specific attached load. The JSI system is factory preset with only a low proportional gain (for safety reasons). You must tune the JSI to achieve desired performance levels.

Once you have the system installed and the motor connected to its intended load, you can determine whether any fine tuning is required by observing the response of the system to commands from your indexer and by observing how stiff the system is when at rest.

When the motor is at rest, try to deflect the shaft. You should not be able to easily turn the shaft away from its rest position. If it feels very soft, the system gains probably need to be increased. A *soft* system will not respond very quickly to move commands. If the shaft feels stiff, be certain to check that the system is not vibrating. If it is vibrating, the gain may be too high. In extreme cases, the vibration grows in amplitude, producing violent motions that cause the drive to fault or break something. For this reason, you should tune the JSI and servo with some caution.

Select one of the P, I, or D buttons and hold it down. When you do, the two-digit display lights up to indicate the present value for the term you have selected from (0 - 99%). You should note this value in case you wish to return to it. While holding the button for the term you selected, push the UP button to increase the value of the term. Push the DOWN button to decrease the value of the term. You must hold a button down while using the UP and DOWN buttons. If you do not select one of the term buttons, the display will not be lighted, and no value changes occur from holding the UP and DOWN buttons.

The value that will be used is the one displayed when you release the UP or DOWN button. This value will be used by the JSI immediately. Push-button tuning allows rapid adjustments to be made to tuning parameters.

To store values in non-volatile memory, push the PROP GAIN, INT GAIN, DER GAIN and DN buttons. If you ever wish to retrieve the factory default values, all you need to do is push both the P, I, D and UP term buttons together (refer to Figure 4-5). This causes the factory values for each term to be reloaded. Table 4-2 provides a complete listing of the JSI's push-button tuning functions.

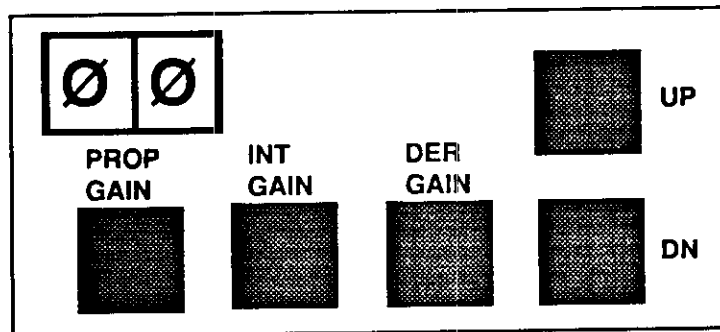


Figure 4-5. JSI Controller Push Buttons

Set integral gain: Press I to show values. Press I and UP or DN to increase/decrease the percent gain.

Set proportional gain: Press P to show values. Press P and UP or DN to increase/decrease the percent gain.

Set derivative gain: Press D to show values. Press D and UP or DN to increase/decrease the percent gain.

Set device address: Press P and D together to show values. Press P and D and UP or DN to increase/decrease the device address.

Set baud rate: Press P and I together to show values. Press P and I and UP or DN to increase/decrease the first two digits of the baud rate.

PROP GAIN	INT GAIN	DER GAIN	UP	DOWN	Function
Ø	Ø	Ø	Ø	Ø	Display Errors
X	Ø	Ø	Ø	Ø	Display Proportional Gain
X	Ø	Ø	X	Ø	Increase Proportional Gain
X	Ø	Ø	Ø	X	Decrease Proportional Gain
Ø	X	Ø	Ø	Ø	Display Integral Gain
Ø	X	Ø	X	Ø	Increase Integral Gain
X	X	Ø	Ø	X	Decrease Integral Gain
Ø	Ø	X	Ø	Ø	Display Derivative Gain
Ø	Ø	X	X	Ø	Increase Derivative Gain
Ø	Ø	X	Ø	X	Decrease Derivative Gain
X	X	Ø	Ø	Ø	Display RS-232 Baud Rate
X	X	Ø	X	Ø	Increase RS-232 Baud Rate
X	Ø	X	Ø	X	Decrease RS-232 Baud Rate
X	Ø	X	Ø	Ø	Display RS-232 Device Address
X	Ø	X	X	Ø	Increase RS-232 Device Address
X	Ø	X	Ø	X	Decrease RS-232 Device Address
Ø	X	X	Ø	Ø	Reserved
Ø	X	X	X	Ø	Reserved
Ø	X	X	Ø	X	Reserved
X	X	X	Ø	Ø	Save Values to Non-volatile Memory
X	X	X	X	Ø	Return to Factory Settings
Ø	Ø	Ø	X	X	Reset
X	X	X	X	Ø	Display Software Revision Level

Table 4-2. JSI PushButton Tuning Functions

- Miscellaneous Functions**
- Return to factory settings: P, I, D, and UP together
  - Save tuning values to non-volatile memory: P, I, D, and DN together
  - Reset: UP and DN together
  - Display Software revision level: P, I, and D together

**General Tuning Considerations** You should remember that all of the terms are interactive and it may take considerable experimentation to find the exact combination of values to get the best performance in your application. In a very general sense, the proportional term should be set first, followed by the integral term, and finally the derivative term.

<b>Proportional Gain</b>	Proportional gain affects system stiffness and accuracy. As the proportional gain is adjusted higher, the influence of the error between the commanded position and the actual feedback position becomes greater. If the gain is sufficiently high, the system will oscillate. This happens because very small position changes are amplified into very large error signals, and the inertia of the servo and load will not allow the system to follow the electronic commands fast enough. The system lag time eventually reaches a point where the feedback and the command signals are in phase. At this point, oscillation results. When this occurs, you must reduce the gain below the oscillation point.
<b>Integral Gain</b>	Integral gain allows the system to compensate for position errors (due, for example, to friction). Integral gain reduces velocity ripple. It does this by slowing down the electronic response time so that it more closely resembles the response of the mechanical components of the loop. The integral gain factor is applied to a summation of all errors that occur from the time zero (when you first enable the command output <b>CMD+</b> & <b>CMD-</b> ). Hence, this gain reduces the steady-state error in the system.
<b>Derivative Gain</b>	Derivative gain adds damping effects to the system. In the case of a system oscillating at the end of a move, or around a change in velocity, increasing the derivative gain will reduce the oscillation. The derivative gain adds phase lead to compensate for the systems natural phase lag. The derivative gain is applied to the rate of change of the position error. Thus, if a constant error occurs, the derivative gain will not affect the system. A Changing position error will allow the derivative gain to affect the system.
<b>Position Errors Due to Load Inertia</b>	Load is the inertia (mass times the radius of rotation squared) seen by the shaft of the motor measured in (Oz-In-In). The amount of inertia affects the torque required. The torque required is a function of acceleration and inertia. If you find that your load inertia is larger than 10 times the motor inertia, you may want to trade some speed performance for accuracy at the final position. In this case, increase the integral gain.
<b>Slow Response Due to Load Inertia</b>	In a system where the load inertia is larger than the factory setting allowed, it may be possible to increase the system response time by allowing the overshoot to increase. In this case, increase the proportional gain, and decrease the integral gain.
<b>Position Errors Due to Friction</b>	Significant friction in the load may cause the end-of-move position to be unacceptably in error. In this case, you can trade off system response for final positional accuracy. To do this, increase the integral gain, and increase the derivative gain.
<b>Shaft and Coupling Vibration</b>	If the load is coupled to the shaft through a non-rigid coupler, it is possible for the shaft and coupler to oscillate at a frequency greater than would be possible for the system as a whole. In this case, it may be possible to trade system response for system stability. To do this increase the integral and derivative gains, while decreasing the proportional gain.
<b>Inadequate Response Time (Frequency Response)</b>	Some systems may involve very little inertia, but need the torque for very high acceleration. In this case, you may want to narrow the range of values that the system can handle to optimize the move profile for a light load. To do this, increase the proportional gain while decreasing the integral term.

- Overshoot** If the system exhibits excessive overshoot (when making the transition from acceleration to constant velocity or from deceleration to a stop), it may be caused by a commanded acceleration that is much higher than what the system can actually produce. If this occurs, reduce the commanded acceleration [refer to the Acceleration (**A**) command in Chapter 5, Software Reference] until this overshoot is minimized. Reducing the integral gain will also limit overshoot in most cases.
- Integral Limit** The integral portion of the PID control algorithm provides an additional factor for tuning purposes. The integrator portion of the control is the integral gain of the position error from the time you enable the servo to the present time. Digitally, this is a summation of all the position errors from each sample period from the time you enable the servo up to the present time. You can limit the maximum value that the summation of errors can achieve with the Define Integral Sum Maximum Sum Limit (**CIL**) command. If you prevent the summation of the errors to become too large, you can reduce the overshoot that is caused by the integral gain *and* achieve steady-state accuracy.