

Application Design

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
- ❑ Use examples to help you develop your application

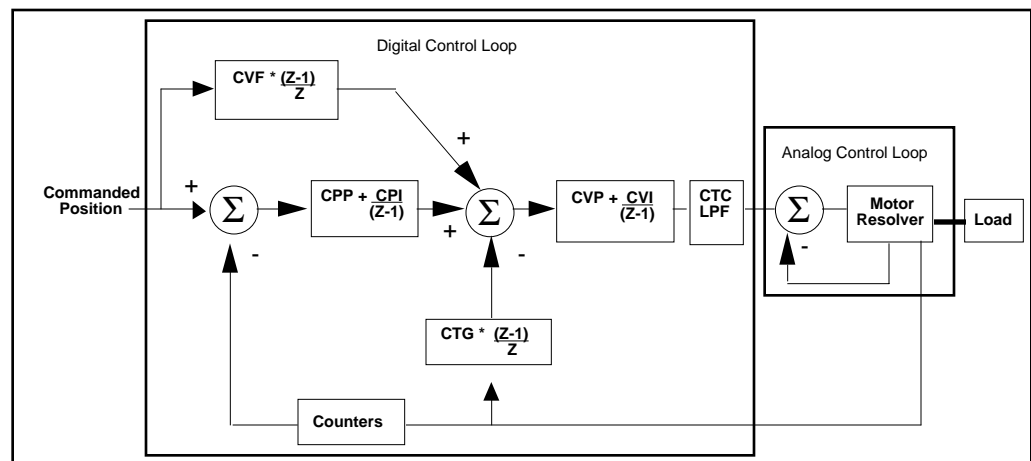
Basic Servo Tuning Theory

The ZX employs two basic control loops.

- ❑ Digital Control Loop
- ❑ Analog Control Loop

The *digital control loop* uses information from the resolver and user inputs to determine what the commanded motor currents should be.

The *analog control loop* takes current commands from the digital control loop and pulse width modulates the bus voltage to achieve these currents in the motor.



Digital and Analog Control Loops

The digital control loop has 15 parameters that you can adjust to obtain optimal shaft performance. You cannot adjust the analog control loop. It is configured to run all sixteen ZX motors (605, 606, 610, 620, 630, 640, 805, 806, 810, 820, 830, 840, 910, 920, 930, and 940) at optimum performance without modification. To ensure that the system operates properly, you must select the correct motor size with the Configure Motor (CMTR) command (refer to the *ZX Indexer/Drive Software Reference Guide*).

Tuning parameters can vary significantly in each operating mode (Position mode, Velocity mode, or Torque mode). To simplify the task of tuning, default tuning parameters are stored for each motor size in each mode. If the default parameters do not provide adequate performance, you can manually tune the drive with the front panel interface or the RS-232C interface. All tuning parameters are accessible via the RS-232C interface; however, only some are accessible via the front panel.

Command	RS-232C	Front Panel	Tuning Commands
CPD	yes	yes	Configure Position Derivative
CPDM	yes	no	Configure Position Derivative Maximum
CPI	yes	yes	Configure Position Integral
CPIM	yes	no	Configure Position Integral Maximum
CPP	yes	yes	Configure Position Proportional
CPPM	yes	no	Configure Position Proportional Maximum
CTC	yes	no	Configure Time Constant
CTG	yes	yes	Configure Tach Gain
CTGM	yes	no	Configure Tach Gain Maximum
CVF	yes	yes	Configure Velocity Feed-Forward
CVFM	yes	no	Configure Velocity Feed-Forward Maximum
CVI	yes	yes	Configure Velocity Integral
CVIM	yes	no	Configure Velocity Integral Maximum
CVP	yes	yes	Configure Velocity Proportional
CVPM	yes	no	Configure Velocity Proportional Maximum

Tuning Parameter Commands

Tuning Procedure

If you are using the ZX for the first time, Compumotor recommends that you use the RS-232C interface. This interface provides access to all of the tuning parameters and gives you real-time access to some of the control variables. Two basic commands—**DDI** (Display Drive Information) and **DSP** (Display Servo Picture)—are designed to help you tune the drive. The **DDI** command lists all the tuning parameters as well as motor's resolution, drive configuration, etc. You can use this command to verify the drive's current operating mode.

Helpful Hint:
Sample **DDI** command response

```

PP  PI  PD  VP  VI  VF  TG
50  05  00  10  00  60  60
*PERCENT
*MAXIMUM 10000 00400 32000 10000 00000 32000 32000

*TIME_CONSTANT=00005 (*100 MICROSECONDS)
*AVERAGE_CURRENT_LIMIT=20.00_AMPS
*PEAK_CURRENT_LIMIT=40.00_AMPS
*MOTOR_RESOLUTION=05090
*RESOLVER_RESOLUTION=AUTO
*MOTOR_TYPE=2620

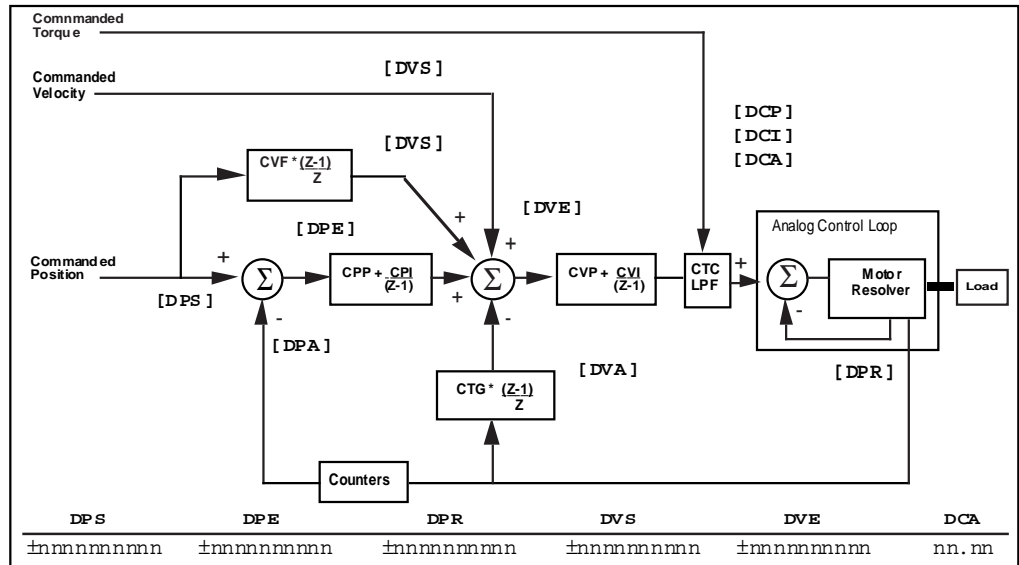
```

All the gain commands have a maximum limit. This provides a wide dynamic range. For example, the gain term **PP** (Position Proportional) can vary from 0-99% using the **CPP** command. The maximum **CPPM** value can vary from 0 - 32,767. The equation below illustrates the number that is actually used in the control loop assuming that **CPP** = 50 and **CPPM** = 10000.

$$\begin{aligned}
 \text{Position Proportional gain} &= (\text{CPP}/100) * \text{CPPM} \\
 &= 50\% * 10000 \\
 &= 5000
 \end{aligned}$$

The **DSP** command gives near real-time servo parameters. Use **DSP** to get an approximate real-time preview of what the control loop is doing and how changing the parameters will affect the system.

Helpful Hint:
 The figure illustrates the response to a **DSP** command and the different parameters that you can display while the drive is operating. The software commands in brackets [] are the actual data in the loop. These variables are continuously updated.



Display Servo Picture Command (DSP) Response

The following display commands will help you tune the ZX.

DCA: Display Current Average

This command displays the ZX's average current.

DCP: Display Current Peak

This command displays the ZX's peak current.

DPS: Display Position Setpoint

This command displays the actual number of steps received from an indexer or pulse generator. This display is inactive in velocity and torque mode operation.

DPA: Display Position Actual

This command displays the motor shaft's actual position.

DPE: Display Position Error

This command displays the difference between the commanded and actual position in user-defined resolution.

DPR: Display Position Resolver

This command displays the position of the resolver. It *rolls over* numerically every mechanical revolution.

DVS: Display Velocity Setpoint

This command displays the desired velocity. In Position mode, this would correspond to the rate of change in steps.

DVA: Display Velocity Actual

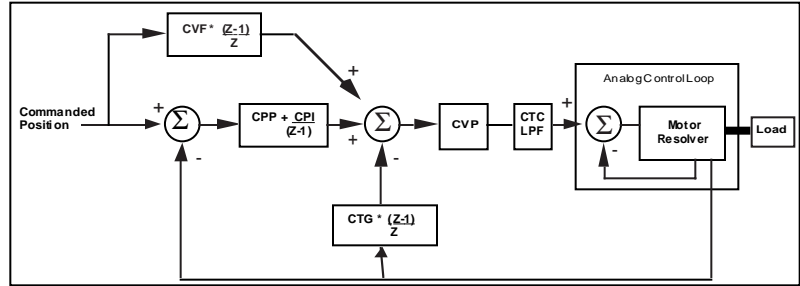
This command displays the actual motor shaft velocity in rpm.

DVE: Display Velocity Error

This command displays the difference in the commanded velocity and the actual velocity in rpm.

Servo Tuning

The ZX's microprocessor-based indexer internally generates position setpoint commands while the ZX's **DSP** closes the position loop.



ZX Tuning

The following table contains the ZX's tuning commands.

Command	Tuning Commands
CPD	Configure Position Derivative
CPDM	Configure Position Derivative Maximum
CPI	Configure Position Integral
CPIM	Configure Position Integral Maximum
CPP	Configure Position Proportional
CPPM	Configure Position Proportional Maximum
CTC	Configure Time Constant
CTG	Configure Tach Gain
CTGM	Configure Tach Gain Maximum
CVF	Configure Velocity Feed-Forward
CVFM	Configure Velocity Feed-Forward Maximum
CVP	Configure Velocity Proportional
CVPM	Configure Velocity Proportional Maximum

Tuning Commands

- CPP: Configure Position loop Proportional Gain** This command directly reflects the *stiffness* of the system. Generally, you want this gain as high as possible without causing the system to oscillate.
- CPI: Configure Position Loop Integral Gain** This command directly influences the final position accuracy. In the default mode, it is turned on, but only slightly. It is error-limited to prevent integral windup.
- CPD: Configure Position Loop Derivative Gain** This command sets both the digital tach gain and the velocity feed-forward gain to the same value. It has the effect of *damping* the system response. This gain is increased if the motor oscillates at zero commanded position.
- CVP: Configure Velocity Loop Proportional Gain** This command directly reflects the *stiffness* of the system similar to the **CPP** command. Generally, you want this gain as high as possible without causing the system to oscillate. The only difference with this command relative to **CPP** is that it takes into account the velocity tach gain.
- CVF: Configure Velocity Feed-forward Gain** This term reduces the position loop following error only when the shaft is turning. It does not affect the system's dynamics.
- CTG: Configure Tach Gain** This term allows additional damping. If you increase this term, the system will become sluggish, but you will be able to stabilize large inertias.
- CTC: Configure Torque Time Constant** This command filters the digital controller's output response. The motor is *commutated* every 100 μ s and the servo loop is updated every 500 μ s. In between each servo update, the commutation can use an average torque commanded value. The default is set to 500 μ s (**CTC5**). This effectively low-pass filters the torque command signal with a -3dB frequency of 2000 Hz. You can change this value to decrease the low-pass filter frequency. This will lower the drive's bandwidth.

Position Mode Tuning Procedure

Use the following steps to tune the ZX.

- Step ①** Set motor resolution to the proper number of steps/rev you desire (refer to the **CMR** command in the [ZX Indexer/Drive Software Reference Guide](#)). The default is 5000 steps/rev.
- Step ②** Check to make sure the **CMTR** command reports back the actual motor you are using. If it is wrong, change it with the **CMTR** command (refer to the [ZX Indexer/Drive Software Reference Guide](#)).
- Step ③** Attach the load and make your desired move with the default settings. Pay careful attention to the response time, end-of-position overshoot, following error, etc.
- Step ④** Vary parameters to improve your performance if needed. Some common performance problems and suggested tuning procedures on how to improve performance are listed below.
- Shaft Seems Spongy** *Solution Procedure:*
1. Increase **CPP**
 2. Increase **CVP**
 3. Decrease **CTG**
 4. Increase **CPI**

Shaft Oscillates

Solution Procedure:

1. Increase **CTG**
2. Decrease **CVP**
3. Decrease **CPP**
4. Decrease **CPI**

Shaft Overshoots at End of Move

Solution Procedure:

1. Decrease **CPI**
2. Increase **CTG**

Shaft Has Too Much Following Error During Move

Solution Procedure:

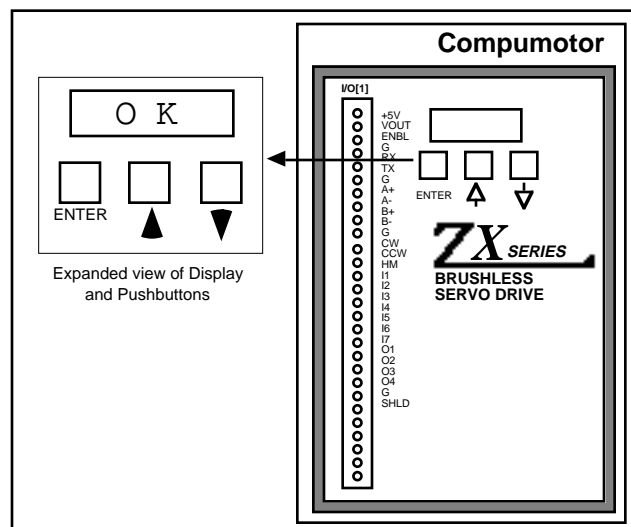
1. Increase **CVF**
2. Decrease **CTG**
3. Increase allowable **CPE**

Step ⑤

When performance is acceptable, you can save your gain parameters with a Save (**SV**) command over RS-232C or with the front panel display (press the **ENTER** pushbutton in the SAVE display).

Alphanumeric Display and Pushbuttons

The ZX has a four-character, dot-matrix, alphanumeric display. All error messages are scrolled across the display when a fault occurs. You can modify many drive parameters with the three pushbuttons.



ZX Drive Display and Pushbuttons

Fault Messages

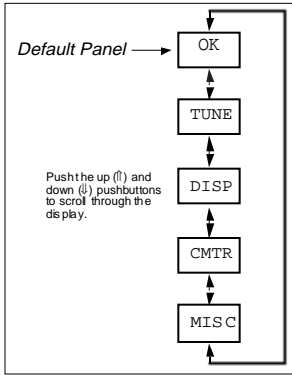
When a fault occurs, the appropriate ZX fault messages will be displayed. A fault code followed by a description of the fault scroll across the display.

Example: **ERROR #04 > OVER_VOLTAGE**

Fault messages are displayed continually until the fault is removed and the ZX is turned on again. Refer to *Chapter ⑧ Maintenance & Troubleshooting* for a complete list of error messages and troubleshooting methods.

Pushbutton Operation

You can use the ZX's pushbuttons to modify drive parameters and to display several drive variables. The figure to the left is an overview of the Main menu panels and sub-panels. Although only one panel is shown on the display at a time, the ZX's display operates in a menu-driven format.



Main Menu Panel (Overview)

The default value for the ZX's Configure Pushbuttons (**CPB**) command is **CPB1**. This fully enables the ZX's front panel. **CPB0** provides you with access to all front panel displays. **CPB0**, however, will not let you activate any of the menus.

OK is the default message. It indicates that you are in the main menu. Use the up and down pushbuttons to view the menu items in the following order:

- OK** Default user message, the home panel
- TUNE** Tune Menu
- DISP** Display Menu
- CMTR** Configure Motor Menu
- MISC** Miscellaneous Menu

To choose a menu, press the up and down pushbuttons to display the menu you want. Press **ENTER** to access the sub-panel menu. The table below shows the main menus and sub-menus.

O K	T U N E	D I S P	C M T R	M I S C
<i>Home Panel</i>	P P n n	D V E L	6 0 5	S A V E
	P I n n	D E R R	6 0 6	R F S
	P D n n	D C A	6 1 0	B R m m
	T G n n		6 2 0	A D p p
	V P n n		6 3 0	F O L L / N T F L
	V I n n		6 4 0	S E Q U
	V F n n		F M C A	R E V #
				J O G

Main Menu Panel for a ZX600 System (Overview)

Pressing the up and down pushbuttons at the same time will return the display to the *Home* panel, regardless of the sub-menu that you are currently using. If you do not press any pushbuttons for several seconds, the display will also return to the home panel.

If an error message is scrolling when the front panel is accessed, the scrolling will be interrupted. When no pushbutton is pressed for several seconds, the scrolling message will return.

If you hold a pushbutton, the selected feature will repeat automatically. If you hold a pushbutton for several seconds, the selected feature will repeat automatically at an accelerated pace. To reset the ZX, press the **UP**, **DOWN**, and **ENTER** pushbuttons together (works like the Reset [**Z**] command).

TUNE Menu

You can select the **TUNE** menu to adjust the system gains for optimum performance. *The ZX is factory-configured for typical user loads. Hence, many applications do not require tuning.* The following gains are available.

- PPnn** Position Proportional Gain
- PInn** Position Integral Gain
- PDnn** Position Derivative Gain
- TGnn** Tachometer Gain
- VPnn** Velocity Proportional Gain
- VI nn** Velocity Integral Gain
- VFnn** Velocity Feed-forward Gain

The variable **nn** represents a percentage ranging from **00** to **99**. Use the **UP** and **DOWN** pushbuttons to locate the desired gain parameter on the display panel. To change the gain value, press and hold the **ENTER** pushbutton while using the **UP** or **DOWN** pushbuttons to increase or decrease the gain. When the desired value is reached, release the **UP** or **DOWN** pushbutton and the **ENTER** pushbutton. After you modify the gain, you can now change another gain or press the **UP** and **DOWN** pushbuttons together to return to the main menu. *To change the maximum gain values, you must use a terminal and communicate via RS-232C.*

DISP Menu

Select the **DISP** menu to display ZX parameters on the front panel. To review the respective numerical values, press the **ENTER** pushbutton. The following parameters are may be displayed:

Helpful Hint:
To return to the Main menu, press the **UP** and **DOWN** pushbuttons simultaneously.

- DVEL** Display Actual Shaft Velocity in rpm
- DERR** Display Position error in steps (-999 to +999)
- DCA** Display Average Current X 100 (0234=2.34 amps)

CMTR Menu


Select the **CMTR** menu to configure the motor type (**CMTR** command). The following choices are available:

- 605 805 910
- 606 806 920
- 610 810 930
- 620 820 940
- 630 830
- 635 840
- 640
- FMCA** **Find Motor Commutation Angle**

To select a motor size, locating the desired motor size with the up and down pushbuttons and press the **ENTER** pushbutton. The preset motor size is designated by an asterisk. Changing motor sizes also changes some of the tuning parameters. The ZX has been configured at the factory for the motor type that you ordered. *If you change motor sizes, be sure to enter the proper **CMTR** value.* To change from one series to another (i.e., 600, 800, or 900) you must use the **CMTR** command via the RS-232C interface.

WARNING

Disconnect the load prior to re-commutating the motor. System damage and/or personal injury can occur during re-commutation if the load is attached.

 **Helpful Hint:**
All of Compumotor's resolvers are pre-aligned to the rotors at the factory, so this procedure is not usually necessary.

CMTR recalculates the mechanical offset between the rotor poles and the stator poles. The offset is factory-set to zero, but you can recalibrate the offset if you select the **FMCA** panel and press the **ENTER** pushbutton to select the **FMCA** command. This command locates the rotor magnets relative to the stator windings and allows you to properly commutate the motor.

MISC Menu

Selecting the **MISC** menu allows you to perform a variety of functions. The following section explains the submenu choices and their functions.

SAVE

Saves the servo tuning parameters to battery-backed RAM. To use, press the **ENTER** pushbutton. ***SV*** will be displayed when this function is executed.

RFS

This option returns all servo parameters to factory settings. To use this command, press the **ENTER** pushbutton. **FSET** will be displayed after the command is executed.

BRnn

This option allows you to change the baud rate (**mm** = 03, 06, 12, 24, 48, and 96—these values represent baud rates 300, 600, 1200, 2400, 4800, and 9600 respectively). To change the baud rate, press the **ENTER** and **UP** or **DOWN** pushbuttons simultaneously (as appropriate).

ADpp

This option allows you to change the device address (**pp** represents a device address from 01 to 99). To change the device address, press the **ENTER** and **UP** or **DOWN** pushbuttons simultaneously (as appropriate).

FOLL or NTFL

If you have a ZX, **NTFL** will be displayed in this menu. No further access is granted.

If you have a ZXF, **FOLL** will be displayed. To change the following ratio on the fly, press the **ENTER** pushbutton. The four least significant digits of the **FOL** command will be displayed. The least significant digit represents the following ratio in tenths (it should be blinking). This indicates that any changes to the following ratio will be in 1/10 increments. To select a higher magnitude of ratio change, press either the **UP** or **DOWN** pushbuttons. This will respectively move the blinking cursor to the left or right. Any changes to the following ratio, will now be at this new digit's magnitude.

100s	10s	1s	0.1s	FOL weighting
a	b	c	d	Blinking digit location

To change the ratio at the blinking digits magnitude, simultaneously press the enter key with the appropriate arrow key. *You must press the **UP** and **DOWN** pushbuttons simultaneously to return to the Main menu.*

SEQU

This option allows you to select and run any of 99 sequences. Press the **ENTER** pushbutton. **XSnn** will be displayed (the variable **nn** represents the current sequence selected). To select a new sequence, press the **ENTER** and **UP** or **DOWN** pushbuttons simultaneously (as appropriate). To run the selected sequence, press only the **ENTER** pushbutton. The display should show **XRnn** and run sequence **nn** repetitively. Pressing only the **ENTER** pushbutton again will return you to the **XSnn** display, where you can select a new sequence. *You must press the **UP** and **DOWN** pushbuttons simultaneously to return to the Main menu.*

REV#

When you press the **ENTER** pushbutton, this menu displays the current microprocessor and

DSP software revision levels (respectively).

JOG

Use the following steps to execute jogging from the front panel:

- ① Enable the Jog function with the **OSE1** command via RS-232C.
- ② Enable the pushbuttons with the **CPB1** command via RS-232C.
- ③ Press ENTER—**HI** will appear. This indicates that the axis will jog at the high jog velocity (**JVH**). Press ENTER again to change **HI** to **LO** (**JVL**).
- ④ Press ENTER and the up or down pushbuttons (*simultaneously*) to begin jogging. **UP** selects CW motion. **DOWN** selects CCW motion. If no motion occurs, check the status of your limits.
- ⑤ To return to the Main menu, press the **UP** and **DOWN** pushbuttons simultaneously.

Motion Profile Application Considerations

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

Positional Accuracy vs. Repeatability

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.

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. The ZX has minimal calculation-delay time associated with a Go (**G**) command. This delay can be as low as 500 μ s. The ZX has an internal timer that allows you to monitor the elapsed time of your move. The response of the **TM** command shows you the previous move's execution time.

Predefined Gos

For the fastest possible calculation move times, predefined gos can be used. The **GDEF** command is used to execute predefined gos. The predefined gos are faster because any calculations are performed ahead of time and stored in memory. Refer to the **GDEF** command to execute a predefined go. Perform the following commands to make a predefined go. You can have up to 16 predefined moves.

Command	Description
> GDEF1, A100, AD100, V4, D50000	Defines predefined move #1
> GD1	Execute predefined move #1

Preset Mode Moves

A preset move is a move distance that you specify in motor steps. You can select preset moves by putting the ZX into Normal mode (**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 with the Mode Position Incremental (**MPI**) command. You can select absolute moves with the Mode Position Absolute (**MPA**) command. At any time, you can change the mode you are in and request the state in which the ZX is configured by issuing the **DR** command.

Continuous Mode Moves

The Continuous mode (**MC**) command accelerates the motor to the velocity that was last specified with the Velocity (**V**) command. The motor continues to move at the specified velocity until Stop (**S**) or Kill (**K**) is issued (or a velocity change is specified). To change velocity while the motor is moving, use the instantaneous velocity command (**IV**). Another way to change velocity while moving is to enter Motion Profiling mode (**MPP**).

In Motion Profiling mode, all buffered commands are executed immediately—therefore you only have to enter the **V** command to change the velocity. No **G** is needed following the **V**.

Continuous mode is useful for the following applications:

- ❑ Applications that require constant movement of the load and motion is not based on distance, but on internal variables or external inputs
- ❑ When the motor must be synchronized to external events such as trigger input signals.

 **Helpful Hint:**

In this example, velocity is changed based on external inputs

<u>Command</u>	<u>Description</u>
> PS	Pauses motion until a C command is reached
> MPP	Places the ZX in MPP mode
> IN1A	Sets up I1 (Input 1) as trigger bit 1
> IN2A	Sets up I2 (Input 2) as trigger bit 2
> LD3	Disables the CW and CCW limits (<i>This command is not necessary if the limits are installed</i>)
> MC	Sets unit to the Continuous mode
> A25	Sets acceleration to 25 rps ²
> AD25	Sets acceleration to 25 rps ²
> V1	Sets velocity to 1 rps
> G	Executes the move (Go)
> T1	Waits 1 sec after the motor reaches constant velocity
> V5	Sets velocity to 5 rps
> TR10	Waits for trigger bit 1 to go on and bit 2 to go off
> STOP	Stops the motor
> NG	Ends Profiling mode
> C	Continues execution of commands

These commands cause the ZX to run in Continuous mode. The motor reaches 1 rps, waits for 1 second, changes velocity to 5 rps, waits for you to turn **I1** (Input 1) on and turn **I2** (Input 2) off, and then stops. *The **V0** and **STOP** commands 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).* The **DIN** command (an immediate command) simulates the state you want the inputs to be in. In the example above, you could simulate the activation of the trigger state without physically toggling the inputs, by using the **DIN** command as follows.

> **DINEEE10EEEEE**

E means *do not affect the input*. A **1** makes the input one, a **0** makes the input zero. Each **E**, **1**, or **0** represents an input bit. There are 10 inputs. The **1** and **0** in this example correspond to **I1** and **I2** on the front panel. The first 3 **E**'s correspond to the CCW, CW, and Home limits.

Incremental Mode Preset Moves

When you are in Incremental mode (**MPI**), a preset move moves the motor the specified distance from its starting position. You can specify the direction with the optional sign (**D+8000** or **D-8000**), or you can define it separately with the Change Direction (**H+** or **H-**) command.

Command	Description
> LD3	Disables CW & CCW limits (<i>not needed if limits are installed</i>)
> MPI	Sets unit to Incremental Position Mode
> MN	Places the ZX in the preset mode
> PZ	Zeroes the position counter
> A25	Sets acceleration to 25 rps ²
> AD25	Sets deceleration to 25 rps ²
> V5	Sets velocity to 5 rps
> D8000	Sets distance to 8,000 steps
> G	Executes the move (Go)
> D12000	Sets distance to 12,000 steps
> G	Initiates motion
> 1PR	Reports the setpoint (commanded) position Response: 20000

Absolute Mode Preset Moves

A preset move in the Absolute mode (**MPA**) moves the motor to the distance in an absolute coordinate system that you specify relative to an absolute zero position. You can set the absolute position to zero with the Position Zero (**PZ**) command or by cycling the power to the indexer. 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. 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 ZX 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 ZX 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 performs the same commands as the incremental position example. In this case, the **PR** command will report a different position because it is working in an absolute coordinate system.

Helpful Hint:

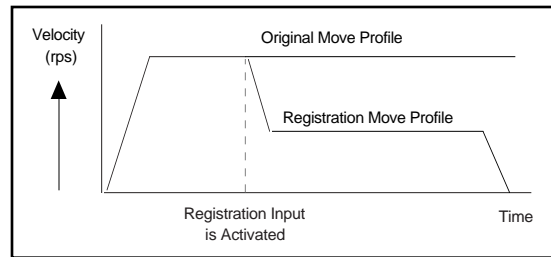
The motor will move to absolute position 8,000. The second move is 4,000 more steps to the absolute position of 12,000 steps. The **PR** command reports a setpoint (commanded position) of 12,000 steps.

Command	Description
> LD3	Disables the CW & CCW limits (<i>not needed if limits are installed</i>)
> MPA	Sets unit to Incremental Position Mode
> PZ	Zeroes the position counter
> A25	Sets acceleration to 25 rps ²
> AD25	Sets deceleration to 25 rps ²
> V5	Sets velocity to 5 rps
> D8000	Sets distance to 8,000 steps
> G	Executes the move (Go)
> D12000	Sets distance to 12,000 steps
> G	Initiates motion

> **1PR** Reports the setpoint(commanded) position
Response: **12000**

Registration

Registration with the ZX provides the ability to change the move profile which is being executed to an unrelated move profile defined as a registration move. **This unrelated registration move is executed when an input to the ZX transitions from a high to a low level.** You can only define input #7 (**I7**) as a registration input (refer to the **IN** command [**IN7Q**] in the *ZX Indexer/Drive Software Reference Guide*). The registration input has the highest priority of any input. Upon receiving the registration input the motor's current position is captured within 50 microseconds. The registration move profile is executed during the next update period. The registration move profile uses the actual position captured (within 50 microseconds) as its starting or zero reference point.

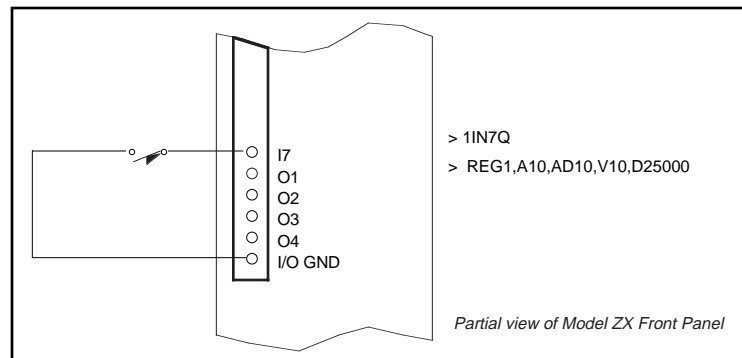


Registration Move

The interrupt is *edge-sensitive* to the voltage high-to-low transition (if you have a bouncy switch for the registration input, use the debounce [**TDR**] command to ensure that false registration interrupts do not occur). With the ZX, a registration interrupt can interrupt another registration move.

The ZX has 7 programmable inputs labeled **I1** through **I7**. Only **I7** can be defined as registration input. **REG1** defines the move when input **I7** is configured as a registration input.

Helpful Hint:
When **I7** is defined as a registration input, it **cannot** be toggled at a frequency higher than 1 kHz, or once every millisecond.



ZX Registration

The syntax for defining a registration move using input **I7** is:

> **REG1,A10,AD10,V10,D25000**

The registration move **REG1** will be performed when **I7** is activated. The acceleration (**A10**) will be 10 rps², the deceleration will be 10 rps², the velocity will be 10 rps, and the distance traveled will be 25,000 steps.

Bouncy Registration Inputs

The switch for the registration may be bouncy or noisy and may take a few milliseconds to settle (since the registration inputs to the ZX are interrupts that are edge sensitive). Hence, a bouncy switch each edge appears like a registration interrupt and the registration move is made from the distance position at which the latest edge was detected. The ZX allows you to debounce the inputs (with a software command). You can ignore any bouncing transitions from your switch after the initial registration interrupt occurred. The time in which the interrupts or false edges are ignored is determined by the number you enter for the **TDR** command. **TDR** is the debounce time in milliseconds that you specify so the inputs cannot cause another registration interrupt until the switch settles.

In the following example, an application needs a registration move. The motor has a resolution of 5,000 steps per revolution. The move must turn the motor 1 revolution at 10 rps. If an input does not occur, the move will be a 500,000-step move. The ZX is

configured as follows:

Helpful Hint:

If **I7** is toggled (voltage high to low), the corresponding registration move is run. The **DIN** command cannot be used to activate the registration input (the registration inputs are hardware oriented).

<u>Command</u>	<u>Description</u>
> 1IN7Q	Defines I7 as a registration input
> REG1, A1Ø, AD1Ø, V1Ø, D5ØØØ	Defines the registration move
> D5ØØØØØ	Sets distance to 500,000 steps
> V5	Sets velocity to 5 rps
> G	The preset move is initiated

Jogging the Motor

In some applications, you may want to move the motor manually. You can configure the ZX to allow you to move the motor manually with the Configure Input (**IN**) command. Define the jogging velocity with the Jog Velocity High (**JVH**) and Jog Velocity Low (**JVL**) commands. 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 **OSE1**. Once you set up these parameters, you can attach a switch to the jog inputs that you defined and perform jogging (# represents digits 1 - 7, which you enter). The following example shows how to define power-up sequence 100 to set up jogging.

Step ①

Define a power-up sequence.

<u>Command</u>	<u>Description</u>
> XE1ØØ	Erase sequence #100
> XD1ØØ	Define sequence #100
> LD3	Disables the limits (<i>not needed if you have limit switches installed</i>)
> JA25	Set jog acceleration to 25 rps ²
> JAD25	Set jog deceleration to 25 rps ²
> OSE1	Enables Jog function
> JVL.5	Sets low-speed jog velocity to 0.5 rps
> JVH5	Sets high-speed jog velocity to 5 rps
> IN1J	Sets I1 as a CW jog input
> IN2K	Sets I2 as a CCW jog input
> IN3L	Sets I3 as a speed-select input
> XT	Ends sequence definition

Step ②

Reset the ZX.

<u>Command</u>	<u>Description</u>
> Z	Resets the ZX

Step ③

Turn on **I1** to move the motor CW at 0.5 rps (until you turn off **I1**).

Step ④

Turn on **I2** to move the motor CCW at 0.5 rps (until you turn off **I2**).

Step ⑤

Turn on **I3** to switch to high-speed jogging.

Step ⑥

Repeat steps ③ and ④ to perform high-speed jogging.

Backlash Compensation

The ZX has the capability to compensate for backlash in the gearing of your system. You can specify different compensations depending on the direction the motor is moving. You will use the **BL** command for backlash compensation. Refer to the [ZX Indexer/Drive Software Reference Guide](#) for a detailed description of the backlash command. The syntax of the command is as follows:

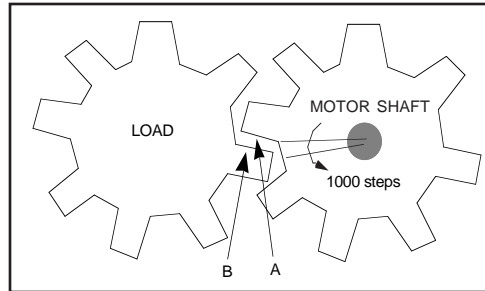
<u>Command</u>	<u>Description</u>
> BLn,m	Variable n is the amount of CCW steps that should be compensated. Variable m is the number of CW steps that will be compensated.

If a CW move is made, an extra **m** steps will be made to account for the backlash. The concept is that the load will not begin to move until the motor moves enough to make contact with the gearing. The same is true for the opposite direction. The backlash compensation may be different in either direction so you can program them independently. The load will not move until point A contacts point B. This is the backlash associated with changing direction. Assuming this distance to be 1,000 motor steps the Backlash command will be typed as follows:

<u>Command</u>	<u>Description</u>
----------------	--------------------

> **BL1000,2000** **1000** is the amount of CCW steps that should be compensated.
 2000 is the number of CW steps that will be compensated.

For a 25,000-step CCW move, the motor will actually move 26,000 steps to remove the backlash but the position counter will only change by 25,000 steps. This is done because the load is expected to move whenever the motor moves. The load should move a certain distance when the motor moves a certain distance. If there is backlash in the system, the load will not move the correct distance when the motor is commanded to move in the opposite direction from its previous move. In this example, the motor was moving in the CW direction. The gearing was flush and the teeth were touching. When the direction changes, the teeth must move 1,000 steps before they are again in contact with the load gear. Thus for the load to move 25,000 steps the motor will have to move 1,000 steps until the teeth are in contact then move 25,000 steps so that the load will actually move 25,000 steps. This mode allows you to compensate for the system error between the motor and the actual load position. *The compensation only occurs when you change direction.*



Backlash Compensation

Defining a Home Location

The ZX's go home function brings the motor to a home reference position. The homing function allows backs the motor to a home switch and stops it on a specific edge of the switch. You can program the active level of the switch. The homing function also allows you to home to the Z channel of the ZX motor's resolver. This occurs at the rollover between resolver position 0 and 65535. Use the Display Position Resolver (**DPR**) command to locate this position.

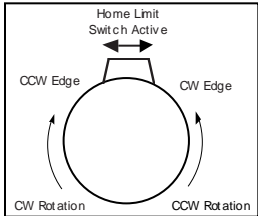
Homing to a Switch

The ZX home position is located where the edge selected with the **OS** command of the Home Limit input occurs (i.e., the ZX 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 stopping, the ZX positions the servo motor away from the selected edge of home limit signal in the opposite direction of the final approach direction of the go home move. After coming to a stop a second time, the ZX 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 ZX's resolver Z Channel pulse are active.

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 CW 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 PLC for example). In such a situation, initiate homing operations from the opposite side of the selected edge of home. When the homing operation is complete, the indexer resets its internal position counter.



Go Home Description

Go Home

You can use the home limit input in conjunction with the resolver's internal Z Channel input to select a final home position. To enable Z Channel homing, 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 ZX's internal resolver roll-over position is used for final Home positioning. *The internal Z Channel pulse and the Home Enable input 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 go home velocity can also be set by the **GHV** command. The last specified value will be used for the go home.

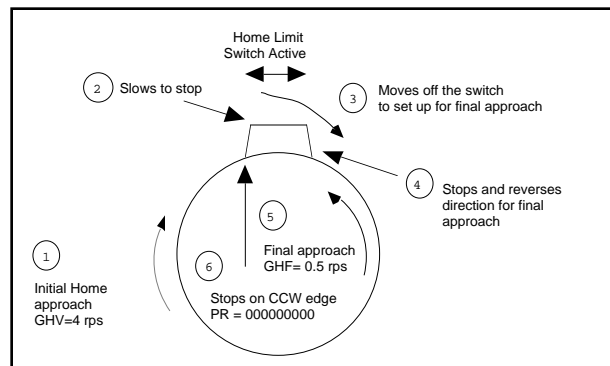
If you use an input to implement a go home move, the velocity provided by the **GHV** command is used to execute the homing function. The acceleration and deceleration parameters for this move are the home value for acceleration (**GHA**) and deceleration (**GHAD**). If an end-of-travel limit is activated before home is found, the indexer reverses direction and attempts to find the home position again. If the other limit is activated before the indexer finds home, the indexer will stop trying to go Home. The indexer can indicate whether or not the homing process was successful by responding to the Request Indexer Status (**R**) and Go Home Status (**RG**) commands. After a successful home move, the position counter is reset to zero.

Go Home Example

Command	Description
> LD3	Disables positive and negative limit inputs
> OSB1	Back up to home limit
> OSG1	The final approach direction is CCW
> OSH1	The reference edge is the CCW edge
> GHA1	Sets go home acceleration to 1 rps ²
> GHAD1	Sets go home deceleration to 1 rps ²
> GH.5	Sets go home in the positive direction at 0.5 rps

The motor starts to move (CW) toward the home position. Upon reaching the home switch, the motor stops. It continues to move CW at the final go home speed until it is off of the home switch. It will then be in position to reverse direction and make its final go home approach (CCW). It will encounter the home switch on the CW edge first and will continue to move until the switch is inactive which will be the CCW edge.

Helpful Hint:
This figure illustrates the homing procedure for these set up commands.



Go Home Example

Creating Motion Programs and Sequences

You must program the ZX to perform motion functions. A motion program consists of initialization (or ZX set-up), move profiles, and an I/O or RS-232C interface to execute motion instructions.

Sequence Commands

Sequences are the building blocks of motion programs for the ZX. A sequence can be one command or up to 8 K bytes of commands. The sequences are stored in battery backed RAM. A sequence is a set of commands that is executed by issuing that sequence number. The ZX has programming capabilities that send program control from one sequence to another (i.e., a **GOTO** statement). The ZX also allows you to transfer program control to another sequence and return to the point of transfer (i.e., a subroutine [**GOSUB**] call). The following

commands define, erase, and run sequences as well as other specialized sequence functions. Refer to the *ZX Indexer/Drive Software Reference Guide* for detailed descriptions and syntax of the following commands.

Sequence Status Commands	<u>Command</u>	<u>Description</u>
	> XBS	Reports the number of bytes available for sequence programming
	> XC	Sequence checksum report
	> XDIR	Reports defined sequences and the bytes of memory they occupy
	> XSD	Sequence status definition report
	> XSR	Sequence status run report
Sequence Programming Commands	<u>Command</u>	<u>Description</u>
	> XD	Starts sequence definition
	> XT	Ends sequence definition
	> XE	Erases a sequence
	> XEALL	Deletes all sequences from battery-backed RAM
Sequence Execution Commands	<u>Command</u>	<u>Description</u>
	> XQ	Sets/resets Interrupted Run mode
	> XRP	Runs a sequence with a pause
	> XR	Runs a sequence
Sequence Branching Commands	<u>Command</u>	<u>Description</u>
	> XG	Exits current sequence and moves to execute another sequence
	> GOTO	Exits current sequence and moves to execute another sequence
	> XR	When used within a sequence it jumps to execute another sequence then returns to the sequence from which it was called
Sequence Debugging Commands	<u>Command</u>	<u>Description</u>
	> XTR	Sequence Trace mode
	> XST	Sequence Single Step mode
	> XS	Sequence Execution status
	> #	Step sequence command
	> DIN	Simulate input state command
Special Sequence Commands	<u>Command</u>	<u>Description</u>
	> WHEN	Special condition command
	> XWHEN	Special condition sequence
	> XFK	Fault sequence

A sequence is a series of commands. These commands are executed in the order in which they are programmed when 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 ZX has 8,000 bytes of nonvolatile memory to store up to 100 sequences. You can use the **XBS** command to determine how many bytes are available in the sequence buffer and the **XDIR** command to determine what sequences have been programmed. 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 sequence definition, enter the Define Sequence (**XD**) command immediately followed by a sequence number (1 to 100) and a delimiter. The Terminate Sequence (**XT**) command ends sequence definition. All commands that you enter after **XD** and before **XT** will be executed when the sequence is run (see example below). Type **DR** to see the state of the ZX.

<u>Command</u>	<u>Description</u>
> 1DR	Displays the present state of the ZX

Perform the following commands.

<u>Command</u>	<u>Description</u>
> MPI	Places the ZX in the incremental mode
> MN	Places the ZX in the preset mode
> FSIØ	Places the ZX in the indexer mode
> LD3	Disables the ZX's limits

Command	Description
> XE1	Erases Sequence #1
> XD1	Begins definition of Sequence #1
> A25	Sets acceleration to 25 rps ²
> AD25	Sets deceleration to 25 rps ²
> V10	Sets velocity to 10 rps
> D5000	Sets distance to 5,000 steps
> G	Executes the move (Go)
> H	Reverses direction
> G	Executes the move (Go)
> XT	Ends definition of sequence
> XR1	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.

Once you define a sequence, it cannot be redefined until you delete it. You can delete a sequence with the **XE** command immediately followed by a sequence number (1 to 100) and a delimiter, then redefine the sequence. *You can use **XEALL** to delete all defined sequences from battery-backed RAM. Use **XEALL** with extreme caution—erased sequences cannot be retrieved.*

Sequence #100 is a power-up sequence (if you have defined it). 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 ZX's nonvolatile memory. The only way to erase these sequences is by using the Erase Sequence (**XE**) or Erase All Sequences (**XEALL**) commands.

Creating and Executing Sequences

You can create sequences via RS-232C. Before you create sequences, you must understand the types of motion and the required user interfaces. To determine the proper user interface, you should be familiar with the methods of selecting sequences within your application.


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. See *Stand Alone Operation* in this chapter for more on this feature.
- Computer Interface:* Use the Execute Sequence (**XR**) command to run sequences. See *Stand Alone Operation* in this chapter for more on this feature.
- PLC (Programmable Logic Controller):* Use the sequence select inputs to run a sequence. See *Stand Alone Operation* in this chapter for more on this feature.
- Front Panel Pushbutton Execution:* Refer to the *Pushbutton Operation* section earlier in this chapter.
- Remote Panel:* Use a remote panel to select programs. See *Standalone Operation* in this chapter for more on this feature.

Subroutines

When you use the **GOTO** Sequence (**XG**) and execute a sequence (**XR**) command, you can execute different sequences from within a sequence. These commands can be substituted for the **GOTO** and **GOSUB** commands respectively. If you use **XG** or **GOTO**, the program will move to the sequence that you specified in **XG** or **GOTO**. 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/GOTO** or **XR/GOSUB**). However, if you use the **XR** or **GOSUB** command, the program will return control to the original sequence that contained **XR** or **GOSUB**. Program control will return to the original sequence when a Terminate Sequence (**XT**) command is reached. This prompts the program to return to the sequence that initiated the move to another sequence.

 **Helpful Hint:** The **XG** command has no limit since the program will not return control to the original sequence.

You can nest up to 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 Sequence #3 from Sequence #2. This nesting procedure can be repeated 16 times.

 **Helpful Hint:**

Command	Description
----------------	--------------------

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. Trace mode was enabled to help you see how the sequence was executed.

```

> XE2           Erases Sequence #2
> XD2           Defines Sequence #2
> A100         Sets acceleration to 100 rps2
> AD100        Sets deceleration to 100 rps2
> V5           Sets velocity to 5 rps
> D25000       Sets distance to 25000 steps
> G            Executes the move (Go)
> XT           Ends Sequence 2 definition
> XE3           Erases Sequence #3
> XD3           Defines Sequence #3
> A10          Sets acceleration to 10 rps2
> V5           Sets velocity to 5 rps
> D-25000      Sets distance to 25000 steps
> G            Executes the move (Go)
> XT           Ends Sequence #3 definition
> XE1           Erases Sequence #1
> XD1           Defines Sequence #1
> XR2           Executes Sequence #2
> GOSUB3       Executes Sequence #3 (same as an XR command)
> XT           Ends Sequence #1 definition
> 1XTR1        Enables Trace mode
> XR1           Executes Sequence #1

```

Asynchronous Events—FAULT and WHEN

The ZX has special sequences that can run when a certain condition occurs. One such sequence is the *power-up sequence* (sequence 100). The **FAULT** and **WHEN** sequences operate similarly.

FAULT Sequence

The fault sequence is automatically executed when a fault condition or a Kill (**K**) occurs. Any condition that faults the ZX (error flashes) activates the fault sequence—if one is defined. If a kill command is issued, the sequence will also run. You can use the fault sequence to place the ZX in a safe state and turn off outputs that may be harmful to the rest of the system. You can use an **IF** command to determine what condition caused the fault so the fault can be remedied. The following steps illustrate the use of a fault sequence. The **IF** statement section explains more about the **ERnnnnn** flag, which indicates what fault condition occurred.

Step ①

Define an input as a user fault input. You can use this input to indicate that a fault has occurred somewhere external to the system. This input will cause a fault condition.

<u>Command</u>	<u>Description</u>
> IN1U	Defines input #1 as a user fault input

Step ②


Designate Sequence #10 as the fault sequence.

<u>Command</u>	<u>Description</u>
> XFK10	Designates sequence 10 as the fault sequence
> XE10	Erases sequence #10
> XD10	Defines sequence #10
> 1"External_Fault	Quote command
> XT	End definition of sequence #10

The quote command sends a message over the RS-232C link to the terminal to tell the operator that a fault has occurred. You can use the Quote command to write statements to the terminal. Sequence #10 will now be executed whenever a fault occurs. You may now program. In the following example, an alternating loop will be performed.

Step ③

The normal *state* of this example application will be an alternating loop.

 **Helpful Hint:**
The motor will alternate back and forth continuously.

<u>Command</u>	<u>Description</u>
> A50	Acceleration is 50 rps ²
> AD40	Deceleration is 40 rps ²
> D50000	Distance is 50000 steps
> V7	Velocity is 7 rps
> L	Loop command
> G	Initiates motion
> H	Change direction
> N	End the loop

Step ④

You will now cause a system user fault error. Input states can be simulated with the **DIN**

command.

Helpful Hint:

Sequence #10 should automatically execute when the fault occurs.

<u>Command</u>	<u>Description</u>
> DINEEE1	Input #1 is activated

Error #66 should scroll across the ZX's display. To clear the fault, enter the following command.

<u>Command</u>	<u>Description</u>
> ON	Clears fault message

An **IF** statement could have been used to determine what fault condition occurred, then branched to a sequence that handled that fault condition appropriately. For example, the fault sequence will run for several faults. These faults are *limits reached*, *general servo fault*, and *user faults*. Each bit in the error flag (**ERnnnnnnn**) corresponds to one of these faults. Use the **IF** statement to determine which one occurred. Retype sequence #10 (the fault sequence) as follows:

Helpful Hint:

Depending on which error caused the program to branch to the fault sequence, the appropriate message will be displayed. Disable the fault sequence with the **XFKØ** command.

<u>Command</u>	<u>Description</u>
> 1XE1Ø	Erases sequence 10
> 1XD1Ø	Defines sequence 10
> IF (ER1)	IF the CCW limit is hit then
> 1"CCW_LIMIT_WAS_HIT	Display a message
> NIF	End the IF (ER1) statement
> IF (ERX1)	IF CW limit is hit then
> 1"CW_LIMIT_WAS_HIT	display a message
> NIF	End the IF (ERX1) statement
> IF (ERXXXXXX1)	IF the user fault occurred, display a message
> 1"USER_FAULT	
> NIF	Ends the IF (ERXXXXXX1) statement
> XT	Ends fault sequence statement

WHEN Sequence

The **WHEN** sequence runs when a specific condition is true. This could be a variable having a certain value, the inputs being in a specific state or the user flag having a set state. Whatever sequence or program is currently running will be interrupted when the condition is true and the sequence designated by the **XWHEN** command will be executed. In the following example, when *Variable 3 > 50*, or *Input #1* is on, the ZX will execute the **XWHEN** sequence.

<u>Command</u>	<u>Description</u>
> WHEN (VAR3>5Ø_OR_INXXX1)	The when statement that must evaluate true in order for the XWHEN sequence to be run is defined.

Step ①

The **WHEN** sequence is now defined. Sequence #8 is the **WHEN** sequence.

<u>Command</u>	<u>Description</u>
> XWHEN8	Sequence 8 is designated as a WHEN sequence
> XE8	Erases sequence #8
> XD8	Begin definition of Sequence #8
> A5Ø	Acceleration is 50 rps ²
> AD4Ø	Deceleration is 40 rps ²
> D3ØØØØ	Distance is 30000 steps
> V5	Velocity is 5 rps
> G	Initiates motion
> XT	Ends the sequence definition

Step ②

The normal program to be executed is defined and executed here.

<u>Command</u>	<u>Description</u>
> VAR3=Ø	Variable 3 is initialized to 0
> A5Ø	Acceleration is 50 rps ²
> AD4Ø	Deceleration is 40 rps ²
> D5ØØØØ	Distance is 50000 steps
> V7	Velocity is 7 rps
> L	Loop command
> G	Initiates motion
> VAR3=VAR3+1	Variable 3 is increased
> N	End the loop

Step ③

You can cause the **WHEN** sequence to occur using either the **DIN** command to activate the input that will satisfy the **WHEN** condition or the current program will run 50 times, then the **WHEN** sequence will be executed.

Step ④

Enter **XWHENØ**. You can use the **WHEN** statement to change the mode of operation. This command can also preview the effect of multiple **WHEN** statements. Using an input to execute the **XWHEN** sequence can change or interrupt the program. Within the **XWHEN** sequence, you can use an **IF** statement to check the state of one or more inputs. Based on the state of the inputs, different sequences can be executed. Disable the **XWHEN** sequence before continuing with this procedure by entering **XWHENØ**.

Power-Up Sequence Execution

The ZX can be programmed to execute a sequence of commands on power up (sequences can be used as subroutines). Refer to the ***ZX Indexer/Drive Software Reference Guide*** for detailed descriptions and syntax of the following commands.

 **Helpful Hint:**

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.

Command	Description
> XE1ØØ	Erases sequence #100
> XD1ØØ	Begins definition of sequence #100
> LD3	Disables limits
> A2Ø	Sets acceleration to 20 rps ²
> AD2Ø	Sets deceleration to 20 rps ²
> V5	Sets velocity to 5 rps
> D125ØØ	Sets distance to 12,500 steps
> MN	Sets to Normal mode
> MPI	Sets into the incremental mode
> FSIØ	Sets unit to the indexer mode
> XG1	Go to sequence #1
> XT	Ends sequence definition
> XE1	Erases sequence #1
> XD1	Defines sequence #1
> G	Executes a go command
> XT	Ends definition of the sequence
> Z	Resets the indexer and runs sequence #100

A power-up sequence typically stores set-up or initialization parameters that your application requires. *Having motion in your power up sequence is not recommended.* Examples of set-up commands are listed below.

 **Helpful Hint:**

You can put any buffered commands into sequence #100 (if you want to execute them during power up).

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

Sequence Debugging Tools

After sequences are created, you may need to debug them to ensure that they perform the functions properly. The ZX provides several debugging tools.

- In Trace mode, you can trace a sequence as it is executing.
- You can set the state of the ZX 's inputs & outputs via software commands.
- You can enable error messages to explain why the ZX has stopped execution due to a programming error.

Trace Mode

You can use the Trace mode to debug a sequence or a program of sequences. Trace mode tracks, command-by-command, the entire sequence as it runs. It displays to your terminal, over the RS-232C serial link, all commands as they are executed. The following example demonstrates Trace mode.

Step ①

Create the following sequence:

<u>Command</u>	<u>Description</u>
> XE1	Erases sequence #1
> XD1	Defines sequence #1
> A1Ø	Acceleration is 10 rps ²
> AD1Ø	Deceleration is 10 rps ²
> V5	Velocity is 5 rps
> L5	Loop 5 times
> GOSUB3	Jump to sequence #3
> N	Ends the loop
> XT	Ends the definition of sequence #1

Step ②

Define sequence #3.

<u>Command</u>	<u>Description</u>
> XE3	Erases a sequence
> XD3	Defines sequence #3
> D5ØØØØ	Sets the distance to 50000 steps
> G	Initiates motion
> XT	Ends the definition of sequence #3

Step ③

Enter the following command to enable Trace mode.

<u>Command</u>	<u>Description</u>
> 1XTR1	Enables Trace mode

Step ④

Execute the sequence. The commands will be displayed on the terminal as each command in the sequence is executed. Enter the following command.

<u>Command</u>	<u>Description</u>
> XR1	Run sequence #1—response:
*SEQUENCE_ØØ1_____COMMAND_A1Ø	
*SEQUENCE_ØØ1_____COMMAND_AD1Ø	
*SEQUENCE_ØØ1_____COMMAND_V5	
*SEQUENCE_ØØ1_____COMMAND_L5	
*SEQUENCE_ØØ1_____COMMAND_GOSUB3_____LOOP_COUNT_1	
*SEQUENCE_ØØ3_____COMMAND_D5ØØØØ_____LOOP_COUNT_1	
*SEQUENCE_ØØ3_____COMMAND_G_____LOOP_COUNT_1	
*SEQUENCE_ØØ3_____COMMAND_XT_____LOOP_COUNT_1	
*SEQUENCE_ØØ1_____COMMAND_N_____LOOP_COUNT_1	
*SEQUENCE_ØØ1_____COMMAND_GOSUB3_____LOOP_COUNT_2	
*SEQUENCE_ØØ3_____COMMAND_D5ØØØØ_____LOOP_COUNT_2	
*SEQUENCE_ØØ3_____COMMAND_G_____LOOP_COUNT_2	
*SEQUENCE_ØØ3_____COMMAND_XT_____LOOP_COUNT_2	
*SEQUENCE_ØØ1_____COMMAND_N_____LOOP_COUNT_2	
*SEQUENCE_ØØ1_____COMMAND_GOSUB3_____LOOP_COUNT_3	
*SEQUENCE_ØØ3_____COMMAND_D5ØØØØ_____LOOP_COUNT_3	
.	
.	
*SEQUENCE_ØØ3_____COMMAND_G_____LOOP_COUNT_5	
*SEQUENCE_ØØ3_____COMMAND_XT_____LOOP_COUNT_5	
*SEQUENCE_ØØ1_____COMMAND_N_____LOOP_COUNT_5	
*SEQUENCE_ØØ1_____COMMAND_XT	

The format for Trace mode display is: **Sequence Number_Command_Loop Count**

- Step ⑤** To exit Trace mode, enter the following command:
- | <u>Command</u> | <u>Description</u> |
|----------------|--------------------|
| > XTRØ | Exits Trace mode |
- Single-Step Mode** Single-Step mode is another level of debugging. This mode allows you to execute one command at a time as desired. Use **XST** to enable Single-Step mode. To execute a command, use the # sign. By entering # followed by a delimiter, you will execute the next command in the sequence. If you follow the # with a number (*n*) and a delimiter, you will execute the next *n* commands. The following steps demonstrate Single-Step mode.
- Step ①** Enable Single-Step mode.
- > XST1
- Step ②** Begin execution of sequence #1
- > XR1
- Step ③** You will not execute any commands until you use the # command.
- | <u>Command</u> | <u>Description</u> |
|----------------|----------------------|
| > # | Executes one command |
- The response will be:
- ```
*SEQUENCE_ØØ1_____COMMAND_A1Ø
```
- Step ④** To execute more than one command at a time, follow the # with the number of commands you want to execute.
- | <u>Command</u> | <u>Description</u>                                  |
|----------------|-----------------------------------------------------|
| > #3           | Executes 3 commands, then pauses sequence execution |
- To complete the sequence, use the # sign until all the commands are completed. To exit Sequence-Step mode, type:
- > XSTØ
- Simulating I/O Activation** If your application has inputs and outputs that integrate the ZX with other components in your system, you can simulate the activation of these inputs and outputs so that you can run your sequences without activating the rest of your system. Thus, you can debug your program independently of the rest of your system. This is the same way in which a PLC program can be debugged by simulation of input and output states to run various portions of the program. The ZX uses two commands that allow you to simulate the input and output states desired. The **DIN** command controls the inputs and the **DOUT** command controls the outputs.
- You will generally use the **DIN** command to cause a specific input pattern to occur so that a sequence can be run. Use the **DOUT** command to simulate the output patterns that are needed to prevent an external portion of your system from operating. You can set the outputs in a state that will be the inactive state of your external system. When you execute your program, a part in the program that will activate the outputs will not actually turn the outputs on to their active state because the **DOUT** command overrides this output and holds the external portion of the machine in an inactive state. When the program is running smoothly without problems you can activate the outputs and the ZX will affect the external system.
- Outputs** The following steps describe the use and function of the **DOUT** command.
- Step ①** Display the state of the outputs with the **OUT** and **O** commands.
- | <u>Command</u> | <u>Description</u>                |
|----------------|-----------------------------------|
| > 1OUT         | Displays the state of the outputs |
- The response will be:
- ```
*1_A_PROGRAMMABLE_OUTPUT_____ (STATUS_OFF)
*2_A_PROGRAMMABLE_OUTPUT_____ (STATUS_OFF)
*3_A_PROGRAMMABLE_OUTPUT_____ (STATUS_OFF)
*4_A_PROGRAMMABLE_OUTPUT_____ (STATUS_OFF)
```
- | <u>Command</u> | <u>Response</u> |
|----------------|-----------------|
| > 1O | *ØØØØ |
| > DOUT11EE | |
- Step ②** Change the output state using the **O** command.

Step ③

<u>Command</u>	<u>Description</u>
> O1110	
Display the state of the outputs with the OUT and O commands.	

<u>Command</u>	<u>Description</u>
> 1OUT	Displays the state of the outputs

The response will be:

```
*1_A_PROGRAMMABLE_OUTPUT_____ (DISABLED_ON)
*2_A_PROGRAMMABLE_OUTPUT_____ (DISABLED_ON)
*3_A_PROGRAMMABLE_OUTPUT_____ (STATUS_ON)
*4_A_PROGRAMMABLE_OUTPUT_____ (STATUS_OFF)
```

Step ④

<u>Command</u>	<u>Response</u>
> 1O	*1110

You can now disable the outputs into the inactive state using the **DOUT** command. An **E** does not affect the output.

> DOUT00EE

Step ⑤

Display the state of the outputs with the **OUT** and **O** commands.

<u>Command</u>	<u>Description</u>
> 1OUT	Displays the state of the outputs

The response will be:

```
*1_A_PROGRAMMABLE_OUTPUT_____ (DISABLED_OFF)
*2_A_PROGRAMMABLE_OUTPUT_____ (DISABLED_OFF)
*3_A_PROGRAMMABLE_OUTPUT_____ (STATUS_ON)
*4_A_PROGRAMMABLE_OUTPUT_____ (STATUS_OFF)
```

<u>Command</u>	<u>Response</u>
> 1O	*0010

Inputs

The following steps describe the use and function of the **DIN** command. You can use it to cause an input state to occur. The inputs will not actually be in this state but the ZX treats them as if they are in the given state and will use this state to execute its program.

Step ①

This sequence will wait for a trigger state to occur and will then begin moving in Continuous mode. An input that is configured as a stop (**S**) input will stop motion.

<u>Command</u>	<u>Description</u>
> 1IN1A	Input #1 is a trigger input
> 1IN2A	Input #2 is a trigger input
> 1IN3D	Input #3 is a stop input
> 1INL0	The active input level is low
> 1XE1	
> 1XD1	Define sequence #1
> TR11	Waits for the input trigger state to be 11
> A100	Acceleration is set to 100 rps ²
> AD100	Deceleration is 100 rps ²
> V5	Velocity is 5 rps
> MC	The continuous mode is activated
> TR10	Waits for a trigger input state of 10
> G	Begins motion
> XT	Ends sequence definition

Step ②

Turn on the Trace mode so that you can view the sequence as it is executed.

<u>Command</u>	<u>Description</u>
> 1XTR1	Turns on Trace mode

Step ③

Execute the sequence.

<u>Command</u>	<u>Description</u>
> XR1	Runs sequence #1

Step ④

When the **TR11** command is encountered, program execution will pause until the trigger condition is satisfied. Simulate the input state with the **DIN** command. *Inputs with an E value are not affected.*

Step ⑤

> 1DINEEE11EEEEE


When the **TR10** trigger is encountered, program execution will pause for the new input pattern. Use the **DIN** command to simulate the desired input state.

> 1DINEEE10EEEEE

Step ⑥

The motor will move continuously until a the stop input is activated. Activate the stop input with the **DIN** command.

> 1DINEEE101EEEEE

 **Helpful Hint:**
To deactivate the I/O simulation commands:

> 1DINEEEEEEEEEEE

> 1DOUTEEEEE

Error Messages

The ZX has an Error Message mode that displays an error message when an invalid command is attempted. This error message can be useful in debugging a sequence. The **SSN** command enables Error Message mode. The following commands demonstrate this mode.

<u>Command</u>	<u>Description</u>
> 1SSN1	Enters the error message mode
> 1D1000000000	

The ZX will respond with an error message: ***INVALID_DATA_FIELD**

<u>Command</u>	<u>Description</u>
> 1DK10000	


The ZX will respond with an error message: ***INVALID_COMMAND**

The error message mode can be exited by typing:

<u>Command</u>	<u>Description</u>
> 1SSN0	Exits the error message mode

Data

Inputs can be defined as data inputs to allow for external entry of motion data, loop counts, sequence select, time delays, and variable values. The following commands will read and enter data from the inputs.

 **Helpful Hint:**
The recommended method of controlling the input lines to read data is through thumbwheels. A PLC may also be used to enter data.

<input type="checkbox"/> VAR	Variable Read	<input type="checkbox"/> TRD	Time Delay
<input type="checkbox"/> DRD	Distance	<input type="checkbox"/> XRD	Sequence Number
<input type="checkbox"/> VRD	Velocity	<input type="checkbox"/> FRD	Following Ratio
<input type="checkbox"/> LRD	Loop Count		

The input lines when configured as data inputs are weighted differently than sequence select inputs. The weighting for data inputs is *binary coded decimal* or BCD. This weighting allows you to enter data via thumbwheels. To use the data inputs to enter data, the outputs must also be used. Outputs 1 - 3 must be configured as data strobe outputs. They are used to select or strobe the appropriate while reading data. Up to 16 digits of data and one sign bit may be entered.

Delays


You can use the Time (**T**) command to halt the operation of the indexer function for a preset time. In *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 *Preset mode*, the motor finishes the move before the indexer executes the time delay.

<u>Command</u>	<u>Description</u>
> PS	Waits for the indexer to receive a C command before executing the next command
> D25000	Sets distance to 25000 steps
> G	Moves motor 25,000 steps
> T5	Waits 5 seconds after the move ends
> H	Changes motor direction
> G	Moves motor 25,000 steps in the opposite direction
> C	Continues execution

High-Level Programming Tools

The Model ZX's X-language includes some commands that are common in most high-level programming languages (Pascal, Fortran or BASIC). In addition to these commands, 30 variables **VAR1-VAR30** are provided for performing mathematical functions and Boolean comparisons. You can access some system variables—**POS** (Commanded Position or Setpoint) and **FEP** (Following Encoder Position).

Branching commands evaluate condition statements (see below) to make branching decisions. If the condition is true, one set of commands is processed—if the condition is false, another set of commands will be executed.

 **Helpful Hint:**
The condition statements that are evaluated can be very complex. The condition statements support all of the following decisions.

IF (condition true)—*execute these commands*
ELSE—*execute these commands*
ENDIF
WHILE (condition true)—*execute these commands*
NWHILE
REPEAT—*Execute these commands*
UNTIL (condition true)

Variables

The ZX has up to 30 variables that can perform multiplication, division, addition, and subtraction. You can assign these variables to various motion parameters. These parameters and the syntax of assigning a variable to them are listed here.