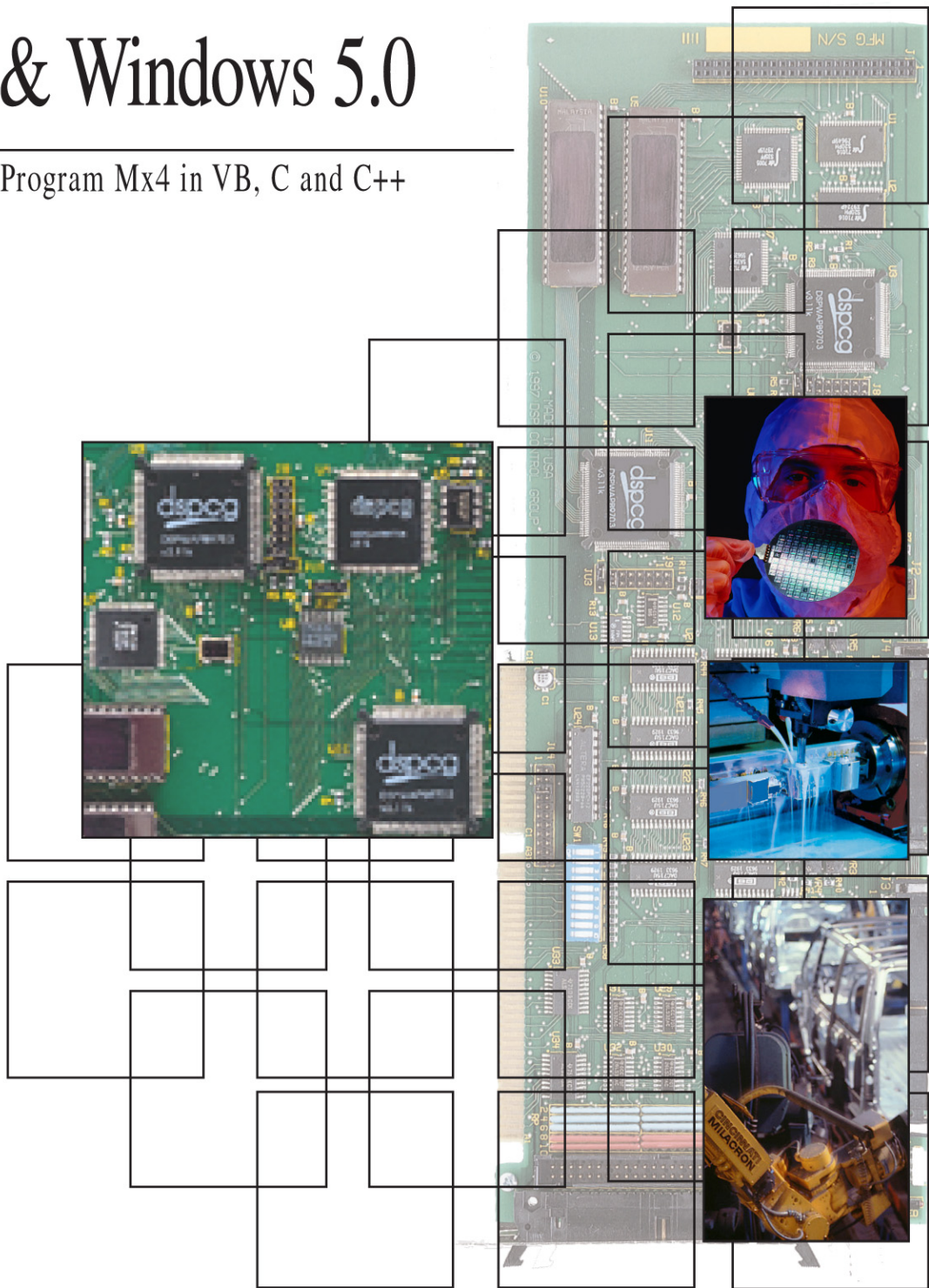


# Mx4 & Windows 5.0

A Guide to Program Mx4 in VB, C and C++



# CONTENTS

<b>1</b>	<b>INTRODUCTION.....</b>	<b>1-1</b>
	C vs. Visual Basic.....	1-1
	Syntax .....	1-2
	Data Types .....	1-2
	Installation.....	1-3
	Other Documentation.....	1-3
<b>2</b>	<b>STEP-BY-STEP APPLICATION DEVELOPMENT.....</b>	<b>2-1</b>
	A Very Simple Motion Control Application .....	2-1
	Setup .....	2-2
	Creating the User Interface .....	2-6
	Supplying Code.....	2-10
	Make An Executable File.....	2-12
	A Two Axis Jog Control.....	2-13
	Introduction.....	2-13
	Getting Started With 3D Panel Controls.....	2-14
	Jog Speed Control.....	2-16
	The Jog Control .....	2-18
	Position Display.....	2-23
	Initialization .....	2-24
<b>3</b>	<b>USING MOTION COMMANDS IN VISUAL BASIC.....</b>	<b>3-1</b>
	Obtaining Access To The DLL Functions .....	3-1
	Real Time Commands .....	3-2
	State Variables .....	3-2
	DSPL Variables .....	3-3

*Contents*

DSPL Program Functionality.....	3-3
User-Defined Units .....	3-4
Inputs / Outputs.....	3-5
Data Table Downloading .....	3-5
Mx4 Dual Port RAM Access .....	3-6
Bus Communication .....	3-6
Serial Communication .....	3-7
Visual Basic Examples .....	3-7

## 4 FUNCTION REFERENCE..... 4-1

Reference .....	4-1
Function Summary.....	4-5
Control Law & Initialization.....	4-6
Simple Motion .....	4-6
Input / Output Control.....	4-7
State Variables, DP RAM.....	4-7
DSPL Variables .....	4-8
System Diagnostic .....	4-8
Multi-Axis RTCs .....	4-8
DLL Synchronization.....	4-9
DSPL & Tables .....	4-9
Contouring .....	4-10
Bus Communication .....	4-10
Motor, Power, Sensors and Drive.....	4-11
Coordinated Motion - Gearing.....	4-11
Coordinated Motion - Cam .....	4-12
Serial Communication .....	4-12
Interrupt Control .....	4-13
Filtering (optional).....	4-13
Function Listing .....	4-14

Mx4 & WINDOWS v5.0  
A GUIDE TO PROGRAMMING Mx4  
IN VISUAL BASIC AND C

This documentation may not be copied, photocopied, reproduced, translated, modified, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of DSP Control Group, Inc.

© Copyright 1998-2001 DSP Control Group, Inc.  
PO Box 39331  
Minneapolis, MN 55435  
Phone: (952) 831-9556  
FAX: (952) 831-4697

All rights reserved. Printed in the United States.

The authors and those involved in the manual's production have made every effort to provide accurate, useful information.

Use of this product in an electro mechanical system could result in a mechanical motion that could cause harm. DSP Control Group, Inc. is not responsible for any accident resulting from misuse of its products.

DSPL, Mx4, Acc4, Vx4++, and Vx8++ are trademarks of DSP Control Group, Inc.

Other brand names and product names are trademarks of their respective holders.

DSPCG makes no warranty or condition, either expressed or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding the licensed materials.

# 1 INTRODUCTION

The VB & C Mx4 DLL (Dynamic Link Library) allows Mx4 to be programmed directly from both Visual Basic and C development platforms on the Windows operating system (both Windows 95/98 and Windows NT/2000 DLLs are available). The DLL contains functions and subroutines which permit the Visual Basic or C programmer to configure Mx4, send commands, read the value of state variables such as position and velocity, and much more.

## C VS. VISUAL BASIC

---

Windows applications can be developed with both C and Visual Basic development tools. The C examples included with the installation were developed using Microsoft Visual C++ v4.0, and the Visual Basic examples were developed using Microsoft Visual Basic v5.0.

Visual Basic is a programming system designed to permit a rapid development of Windows applications. It allows the programmer to interactively create a graphical user interface. Command buttons, labels, scroll bars, and other components of the user interface can be positioned graphically. Each of these objects has a large number of easily modifiable properties which control its appearance and behavior. The programmer must supply Visual Basic code to be executed in response to user interface events. Such an event might be the user clicking on a command button or selecting a command from a menu.

**This manual is written as a Visual Basic Mx4 Programmer's Guide, however, the information herein can be easily adapted for the C programmer as well. Chapters 2 and 3 are intended for the Visual Basic programmer, but the Chapter 4 function reference information, although written from the Visual Basic perspective, is also useful for the C programmer as is described below.**

## Introduction

The same DLL supports both Visual Basic and C application development. The difference lies within the syntax and data types which differ between Visual Basic and C.

## SYNTAX

Chapter 4 of this manual, *Function Reference*, presents the Visual Basic syntax and data types for all of the commands. Of course, the programmer can retrieve this information from the “header” files which are required by both Visual Basic and C. These files, .bas modules for Visual Basic, and .h header files for C, serve as function definitions and prototypes for the Visual Basic / C development tools. The C programmer can use the supplied .h file for obtaining the C function syntax for the command set.

An important distinction between the syntax for Visual Basic and C is the use of both subroutines and functions within Visual Basic. That is, the majority of the command set may be used as either subroutines (no return argument) or functions (return argument). For example, the following lines will illustrate the use of POS\_PRESET as both a subroutine and a function,

```
POS_PRESET 1, 1000           `Visual Basic subroutine
Temp = POS_PRESET (1, 1000) `Visual Basic function
```

In C, the function call would be,

```
POS_PRESET (1, 1000);      /* C function */
```

## DATA TYPES

The Visual Basic (.bas module) and C (.h) header files include information regarding the data types of function arguments and return values. Again, the function listings in Chapter 4 use Visual Basic data types. The C programmer can simply use the .h header file supplied with the DLL for C function data type information.

## INSTALLATION

---

The VB & C DLL distribution media contains an automatic setup program which will install the DLL, associated support files, as well as both Visual Basic and C example applications on the user's system. To install the DLL, simply type a:\setup.exe at the Windows File Run prompt and follow the instructions. The installation will prompt the user for bus communication / serial communication parameters for registry initialization.

## OTHER DOCUMENTATION

---

In conjunction with this manual you may find the following items of assistance:

### MX4 USER'S GUIDE, MX4 OCTAVIA USER'S GUIDE

This manual includes comprehensive information on Mx4's hardware, software, system tuning, memory organization, trouble shooting and more. The *Mx4 User's Guide* is the focal point in learning the technical details of Mx4. All other Mx4 manuals assume that users have already read the *Mx4 User's Guide*.

### MX4PRO: MX4 TUNING EXPERT

This manual describes Mx4Pro - a testing and tuning software used with Mx4. Mx4Pro includes features such as a signal generator oscilloscope and live block diagram which make this software useful for testing and performance optimization.

### DSPL PROGRAMMER'S GUIDE

This manual will assist you with DSPL, DSPCG's high level programming language for the Mx4. DSPL has its own compiler and downloader, which are included in the Mx4pro Development Tools.

## *Introduction*

### Vx4++ USER'S GUIDE

This manual includes information on the add-on drive control option. Vx4++ is DSPCG's multi-DSP based drive controller that provides complete drive signal processing for all industrial DC and AC machines. The capabilities of Vx4++ include that normally offered by servo control amplifiers.

### ACC4 USER'S GUIDE

This manual includes information on the Mx4 Serial Adapter and Mx4 ADC options. The Acc4 daughterboard allows the use of RS-232/485 serial communication to facilitate host-serial-Mx4 based communication.

# 2 STEP-BY-STEP APPLICATION DEVELOPMENT

The purpose of this chapter is to explain how to use Visual Basic and the VB & C DLL to create Windows motor control applications. The intended audience for this section is a control engineer who has some familiarity with Mx4 but knows little about Visual Basic. More experienced Visual Basic programmers may skip ahead to Chapter 3, *Using Mx4 In The Visual Basic Environment*; however, this chapter may serve well as a reference for building an application from the ground up in Visual Basic.

The following text is based upon developing Visual Basic applications using Microsoft Visual Basic v5.0 in the Windows 95/98 operating system.

## A VERY SIMPLE MOTION CONTROL APPLICATION

The example application illustrated in Figure 2-1 displays the position, following error, and velocity of a servo motor connected to Mx4 in real-time. It also allows the user to start and stop the motor by clicking on command buttons.

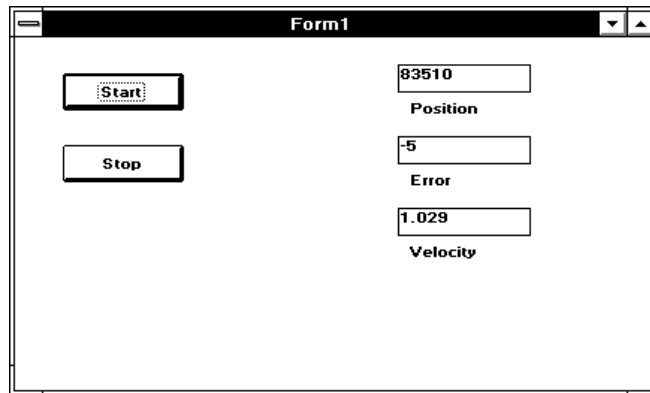


Figure 2-1 : Simple Motion Control Application

The following describes the steps involved in the development of the application program illustrated in Figure 2-1. These steps have been categorized under Setup, Creating The User Interface, Supplying Code, and finally, Making An Executable File.

## SETUP

Several steps are required to create a new Visual Basic program. A Visual Basic program ( or project) consists of a number of different files. The best way to organize these files is to create a separate folder for them.

### STEP 1: Create a project directory

Invoke the Window's File Manager. Create a folder for your project by selecting "New" "Folder" from the File menu and enter the name of the new folder.

### STEP 2: Start Visual Basic.

Once the Visual Basic environment has been opened, select "New Project" from the File menu. Select "Standard EXE" from the New Project window (see Figure 2-2).

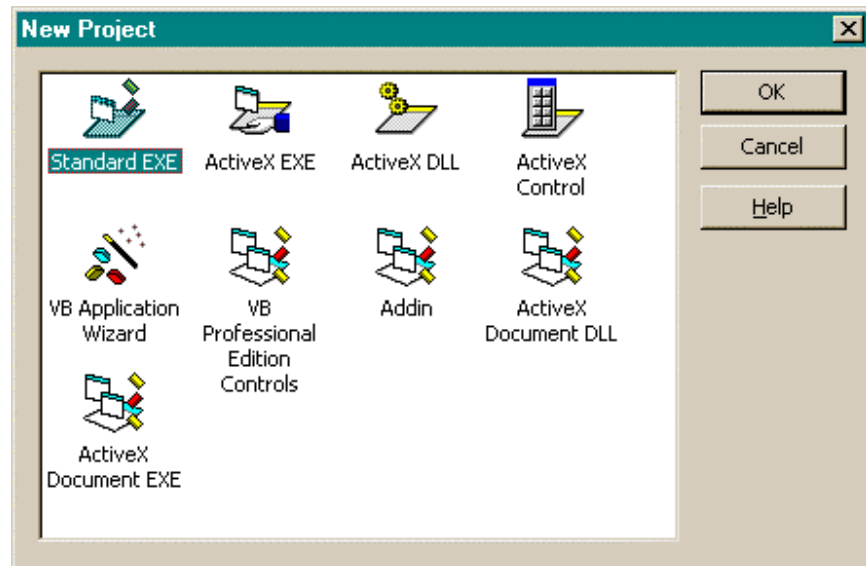


Figure 2-2 : Visual Basic Environment, New Project

**STEP 3: Save the project and form**

After opening a new standard EXE project, the Visual Basic environment displays a blank form, titled Form1 (see Figure 2-3). Use the "Save Project As..." and "Save Form1 As..." File menu commands to save the default form, FORM1.FRM, in the project folder which was created in Step 1.

## Step-by-Step Application Development

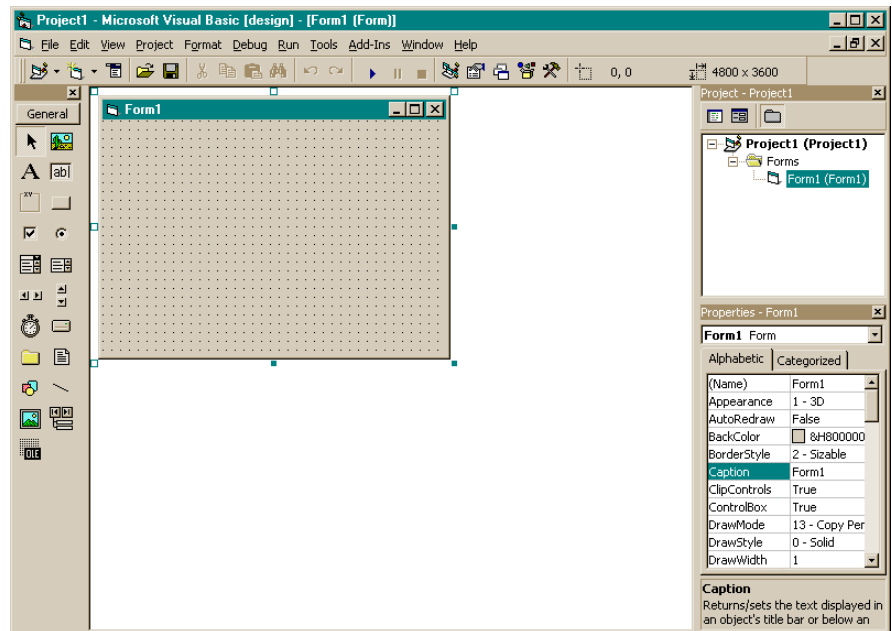


Figure 2-3 : Default form Form1

In order to use the functions contained in the VB & C Mx4 DLL, a module must be incorporated into the the project (the module provides function declarations and constant definitions from the DLL to the current Visual Basic project). The code for this module is provided with the DLL installation (see the MX495\_VB.TXT file in the installation root directory).

### STEP 4: Create declaration module

To keep projects organized and self-contained, it is advisable to keep a copy of the declaration module in each project folder. Copy the MX495\_VB.TXT file from the DLL installation directory to the current project folder. Declaration modules must have a .BAS extension, so rename the file in the project folder to MX4.BAS. Select "Add Module" from the Visual Basic Project menu (see Figure 2-4), and then select "Existing" since the declaration module desired is in the project folder. Select the MX4.BAS module. The module is now part of the project and is displayed in the Project Explorer window (Figure 2-5).

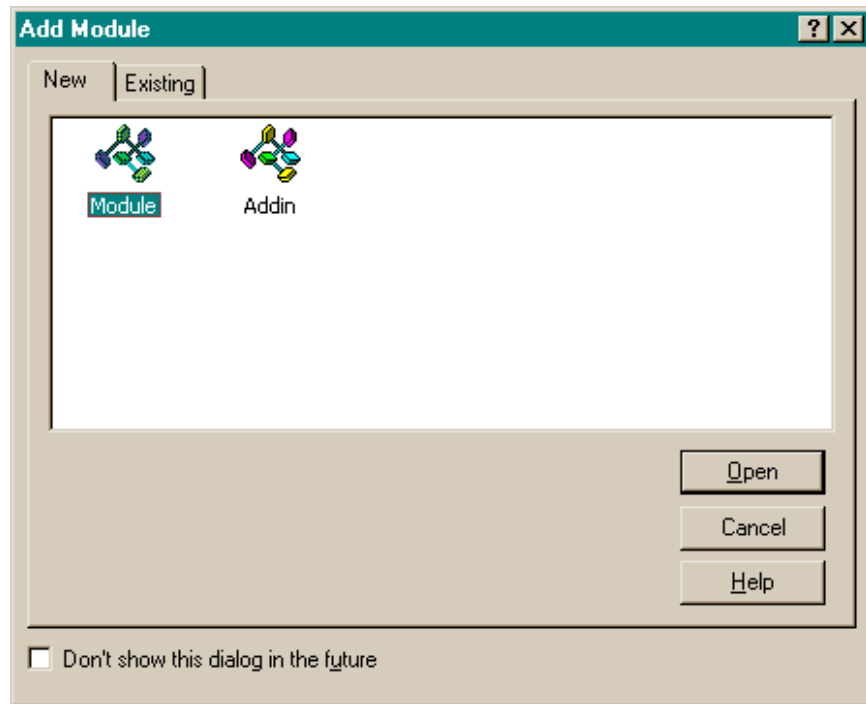


Figure 2-4 : Adding a module to the project

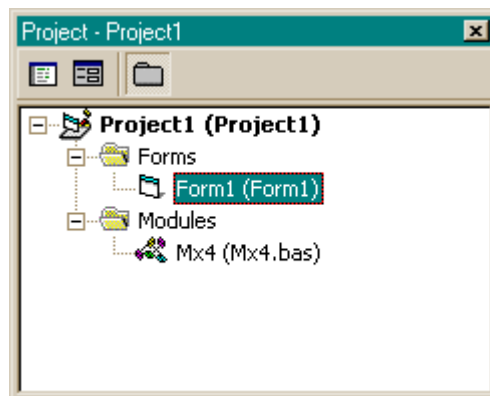


Figure 2-5 : Project explorer window

## CREATING THE USER INTERFACE

We are now ready to start creating the user interface. A Visual Basic application consists of a number of forms ( or Windows). Our project will have only one form, the default form. This form is automatically loaded when the application is started. We create the user interface by adding controls to this form. A control is an object on a form such as a command button. Controls are used to display information and to allow the user to input data and make selections. The Toolbox (Figure 2-6) is used to add controls to the form. We will use a label control to display the current position of axis-1.

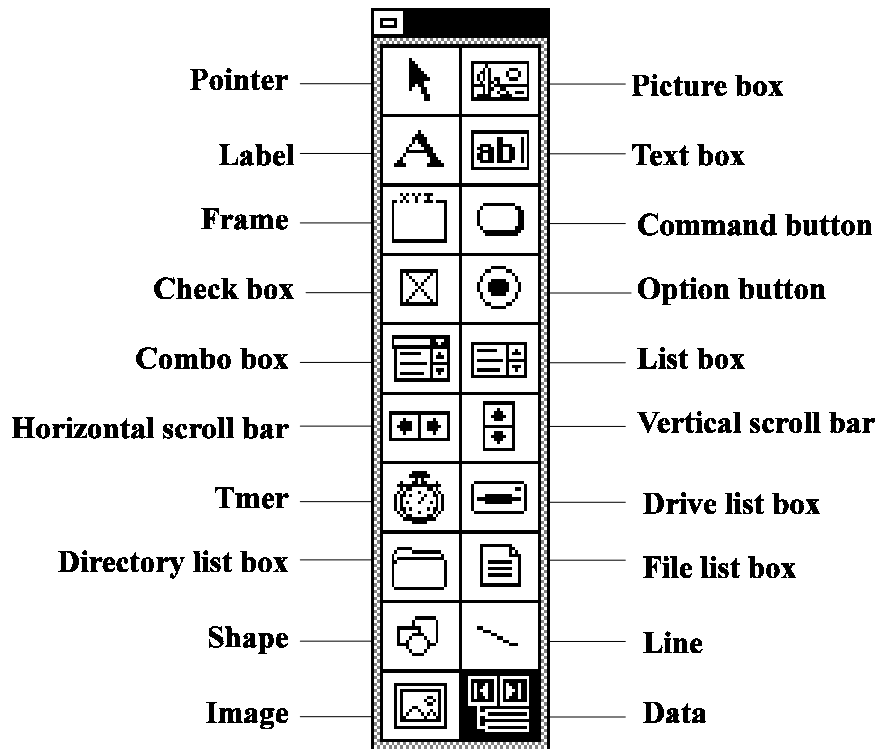


Figure 2-6 : Visual Basic Toolbox

**STEP 5: Create a label control**

Click on the label tool. Position the cursor over the location on the form where you want to place the label. The cursor should now be a cross hair. Click and drag the cross hair to create a label control with the appropriate dimensions.

Controls have properties which effect their appearance and behavior. The properties of a control are modified using the properties window (Figure 2-7).

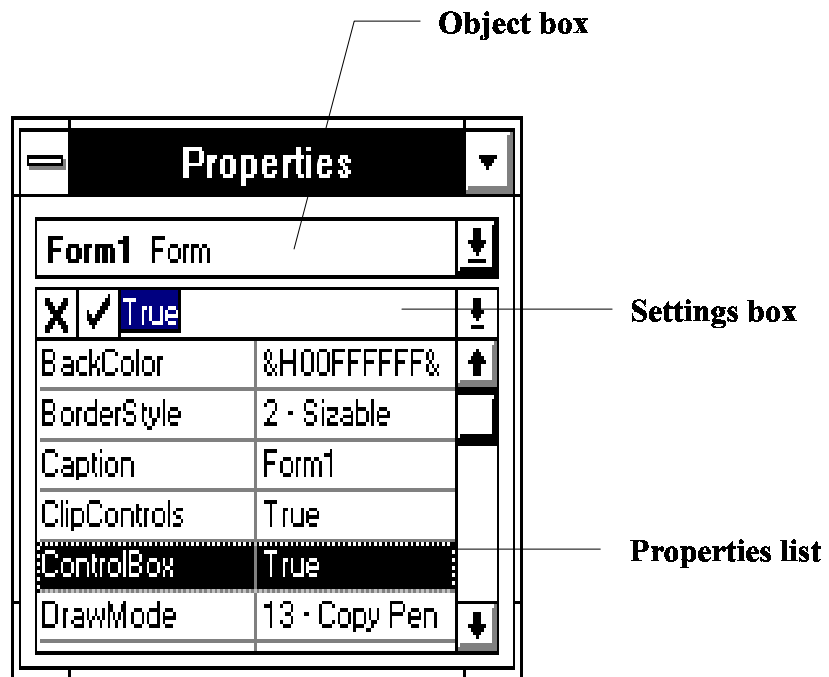


Figure 2-7 : Properties Window

**STEP 6: Bring up the Properties window**

Click on the label control you just created to select it. If you have been following this procedure exactly, it will be selected already. Press the F4 function key to bring up the properties window. If necessary, reposition the properties window so that the label control is visible.

**STEP 7: Select border style**

Use the scroll-bar at the right of the properties list to scroll through the properties of the label control. Find the BorderStyle property and click on it to select it.

**STEP 8: Choose border style**

The default value of the BorderStyle property, "0", is shown in the Settings Box. Click on the arrow to the right of the Settings Box to see a list of all valid border styles. Select "Fixed-Single". Note how this property affects the appearance of the label control.

All controls have a "name" property which is assigned by default when the control is created. This name is used to access the control and its properties from Visual Basic code. It is a good programming practice to assign a more meaningful name to important controls.

**STEP 9: Select a name for label**

Set the name property of the label control created in Step 7 to "PositionLabel".

**STEP 10: Create two more labels**

To display the following error and velocity two more label controls are required. Create these in just the same manner that you created the label control for displaying position. Set their name properties to "ErrorLabel" and "VelocityLabel".

The "Caption" property of a label control specifies the text of the label. The initial value for this property can be specified by the programmer at design-time. The property can also be modified from Visual Basic code while the program is running to create a real-time display. This is how the "PositionLabel", "ErrorLabel", and "VelocityLabel" controls will be used. Additional controls are needed to label the position, error, and velocity displays (i.e. place the text "position" below the "PositionLabel" control).

**STEP 11: Set caption properties**

Create a label control below the "PositionLabel" control and set its Caption property to "Position". Provide labels for the "ErrorLabel" and "VelocityLabel" controls in the same way.

Several more controls are necessary to complete the user interface: a start command button, a stop command button, and a timer. Timers are a special type of control which we will discuss later.

**STEP12: Add Start command button**

Add a command button control to the form. This will be the start button. Set its Caption property to "Start" and its name property to "StartButton".

**STEP13: Add Stop command button**

Add a second command button to the form. This will be the stop button. Set its Caption property to "Stop" and its Name property to "StopButton".

**STEP14: Add a timer control to the form**

The timer control won't be visible when the program is running. It can be placed anywhere on the form. Set its Interval property to 100. (This property specifies the timer's timeout interval in milliseconds.)

The graphical portion of the user interface is now complete. Figure 2-8 shows what it should look like.

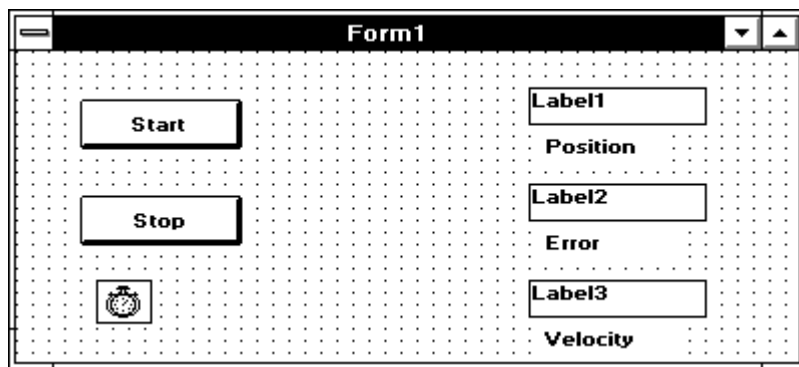


Figure 2-8 : Completed Form

## SUPPLYING CODE

The next set of steps is to supply code to give the application the desired functionality. Each object in the user interface has a number of events associated with it. Event procedures are used to respond to these events. "Load" is an important event associated with a form. This event occurs when the form is first displayed. Our application has only one form, the default form. This form is displayed automatically when the application is invoked. The event procedure for responding to the "Load" event is a good place to put code for initializing Mx4.

### STEP 15: **Bring up the Code window**

Double click on the background (dotted) area of the form to bring up the code window. A template for the Form\_Load event procedure is displayed (Figure 2-9).

### STEP 16: **Supply initialization code**

Add the following Mx4 initialization code to the Form-Load form.

```
Dim sBuffer As String           ` String for the signature

` The buffer must be initialized to a length of at least 11
sBuffer = Space (11)

` This code will use the DLLs signature function to make sure
` the Mx4 card is present at the address specified in the
` Registry, default address is 0xD0000
If (Left$(signature(sBuffer), 3) <> "Mx4") Then
    MsgBox "Mx4 Card NOT found", vbOK, "Mx4 Error"
End
End If

` The next two functions will set the time and position units
` of the DLL into 200 usec, position will be in units of encoder
` counts, velocity will be in units of encoder counts/200usec,
` and so on.
time_unit 1# / 5000#
position_unit 1

` The next 2 function calls will set up the control gains and
` maximum acceleration for axis 1. Notice the error checking,
` a common error that could occur is a parameter may be out of
` range.
If (ctrl (1, 100, 5208, 5796, 1432) <> ERR_OK) Then
```

```
MsgBox "Error occurred", vbOK, "CTRL Error"  
End If  
If (maxacc (1,1#) <> ERR_OK) Then  
MsgBox "Error occurred", vbOK, "MAXACC Error"  
End If
```

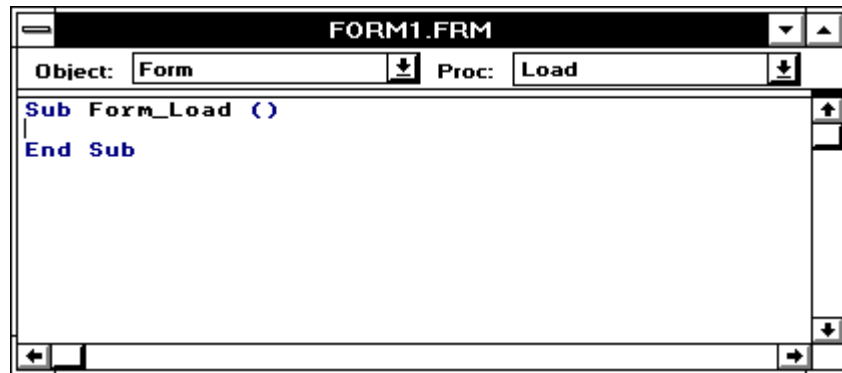


Figure 2-9 : Code Window

The timer control will generate a timeout event every 100ms. These events can be used to update the real-time position, following error, and velocity display. For example, to update the position display, the program must periodically read the position of axis-1 and update the Caption property of the label control for position.

#### STEP17: Supply timeout code

Double click on the timer control. Add the statements shown below to the Timer1\_Timer event procedure.

```
Sub Timer1_Timer  
PositionLabel.Caption = pos(1)  
ErrorLabel.Caption = ferr(1)  
VelocityLabel.Caption = format(vel(1), "###0.000")  
End Sub
```

When the user clicks on the start command button the program must issue a "velocity mode" command to Mx4. When the user clicks on the stop button the program must issue a "stop" command. This functionality is easily implemented by supplying code for the start and stop button's "Click" event procedure.

#### STEP 18: Supply Start button code

## *Step-by-Step Application Development*

Double click on the start command button. Add the statement shown below to theStartButton\_Click event procedure

```
Sub StartButton_Click()  
    velmode 1, 1  
End Sub
```

### **STEP 19: Supply Stop button code**

Double click on the stop command button. Add the statement show below to the StopButton\_Click event procedure.

```
Sub StopButton_Click()  
    stop_axis 1  
End Sub
```

The application is now complete. It's time to see how it works.

### **STEP 20: Run the program**

Start the program by selecting "Start" from the Run Menu. Check it out. If an encoder is hooked up, you should be able to see the position display change as the shaft of the motor is spun. When done experimenting, select "End" from the Run Menu to terminate the application.

## MAKE AN EXECUTABLE FILE

The final step in the creation of a Windows applicaiton using Visual Basic is to generate an executable. The executable can be invoked just like any other windows application, and is independent of the Visual Basic development environment.

### **STEP 21: Create executable**

Select "Make filename.exe..." from the File menu. By default this command will create an executable in the project directory with the same name as the project. Your application is now complete.

## A TWO AXIS JOG CONTROL

---

### INTRODUCTION

This section describes a virtual jog control implemented using Visual Basic and Mx4. This Windows application allows the user to control the position of a two axis servo system by depressing up, down, left, and right buttons. The intended audience for this section is a control engineer who has some familiarity with Mx4 but knows little about Visual Basic.

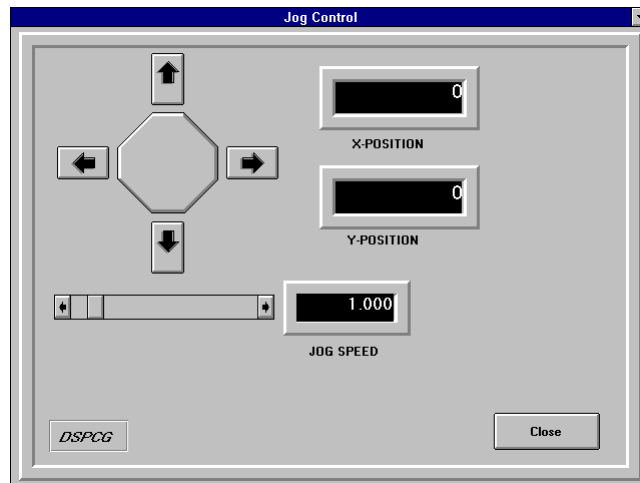


Figure 2-10. Jog Control Application

The application's display is shown in Figure 2-10. The shaft positions of the two servo motors are displayed to the right of the jog control. The horizontal scroll bar below the jog control is used to select a jog speed. To rotate the shaft of the axis-1 servo-motor clockwise the user positions the mouse over the "right" button and presses the mouse button. While the button is held down the shaft rotates at the specified jog speed. When the user releases the mouse button, the motion stops. When the user clicks on the central octagon, the system returns to its home position. To exit the application, the user clicks on the close button.

The VB & C Mx4 DLL is used to program Mx4 in Visual Basic. This section assumes the reader is familiar with accessing the routines in this DLL from Visual Basic.

## GETTING STARTED WITH 3D PANEL CONTROLS

The jog control application achieves its realistic "look" through the extensive use of 3D panel controls. We assume the reader is unfamiliar with these controls and we will provide a brief introduction to them here. Readers already familiar with 3D panel controls may skip this section.

The 3D panel control is one of a number of controls that are made available by incorporating Sheridian Software System's 3D Widgets VBX (THREED.VBX) into a project. This VBX is distributed with the professional edition of Visual Basic. Figure 2-11 shows the additional controls that this VBX adds to the toolbar.

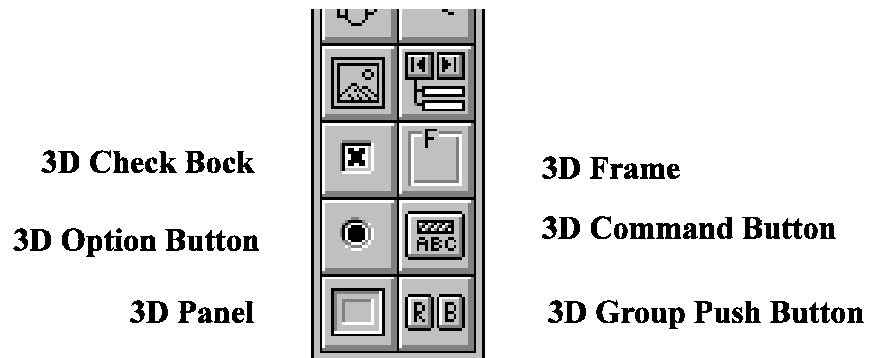


Figure 2-11. Controls provided by THREED.VBX

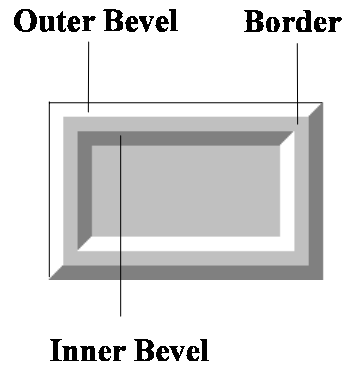


Figure 2-12 Features of a 3D panel control

A 3D panel control can be used to group other controls on a raised background or to lend a three dimensional appearance to a standard control such as a label. A 3D panel has an outer bevel, a border, and an inner bevel. These features are shown in Figure 2-12. The properties `BevelInner`, `BevelOuter`, `BevelWidth`, and `BorderWidth` control the appearance of these features. For example, by setting the `BevelInner` property to 0 (none) the inner bevel can be eliminated.

A 3D panel control is similar to a frame control in that other controls can be placed on it. When the panel control is moved, these controls move with it. When the panel control is deleted, these child controls are deleted as well. In our jog control application, a 3D panel control is used to provide the overall background for the display. 3D panel controls are also used to construct the x-position, y-position, and jog speed displays. A label control is used to provide the numeric display. This label control is placed on a 3D panel control which provides the border for the display. The border panel is itself placed on the background panel. These relationships are illustrated in Figure 2-13.

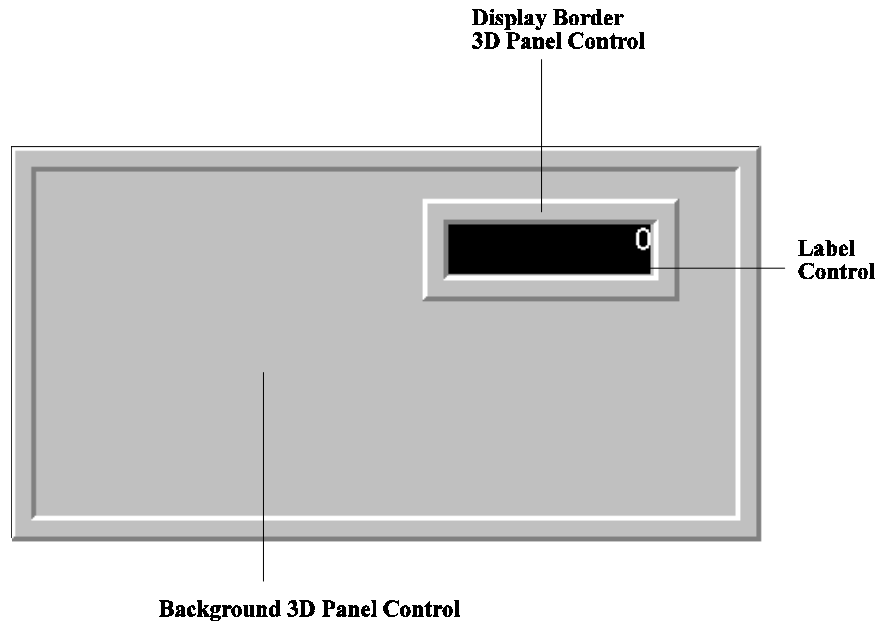


Figure 2-13. 3D panel controls used to provide an application background and a display border.

## JOG SPEED CONTROL

The user selects a jog speed using a horizontal scroll bar. The jog speed is displayed numerically in a label control to the left of the scroll bar. The components of the jog speed control are shown in Figure 2-14. Horizontal scroll bar controls (Figure 2-15) have a `value` property which gives the current position of the scroll box on the scroll bar. The scroll bar's scale can be specified using the `Min` and `Max` properties. `Min` specifies the value of the `value` property when the scroll box is at the left side of the scroll bar. `Max` specifies the value of the `value` property when the scroll box is at the right side of the scroll bar. We have set the `JogSpeedScroll` control's `Min` property to zero and its `Max` property to 32767. Because the `value` property is an integer, we can't use it directly to obtain the jog speed. Some scaling is required. The constant `MaxJogSpeed` is used to specify the maximum jog speed the user can select. Jog speeds must be between 0 and `MaxJogSpeed`. The equation below is used to convert the scroll bar's `value` property into a jog speed.

```
<speed> = JogSpeedScroll.Value * MaxJogSpeed / 32768.
```

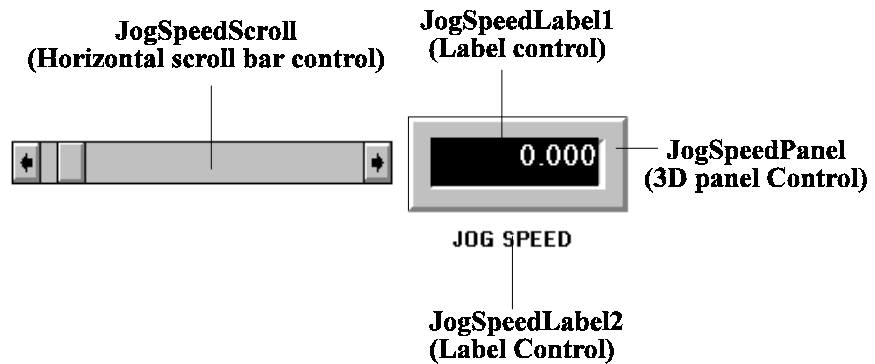


Figure 2-14. Components of the jog speed control

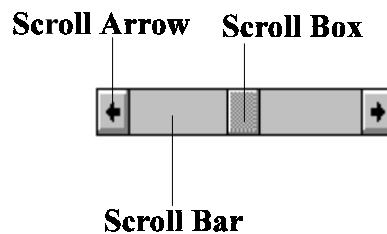


Figure 2-15. Elements of a horizontal scroll bar

There are two events associated with the scroll bar: `Scroll` and `Change`. While the user drags the scroll box, the scroll event is generated repeatedly and the `JogSpeedScroll_Scroll()` event procedure is executed. The user can also change the position of the scroll box by clicking on the left or right arrows or by clicking on the scroll bar itself. The `Change` event is generated whenever the position of the scroll box is changed. The `JogSpeedScroll_Change()` event procedure handles this event. The code for these two event procedures is given in Listing 2-1. Both procedures update the `Caption` property of the label control `JogSpeedLabel1`. This label control is used to display the selected jog speed.

## *Step-by-Step Application Development*

```
Sub JogSpeedScroll_Change ()
    JogSpeedLabel1.Caption = Format(MaxJogSpeed *
JogSpeedScroll.Value / 32768, "####0.0000")
End Sub

Sub JogSpeedScroll_Scroll ()
    JogSpeedLabel1.Caption = Format(MaxJogSpeed *
JogSpeedScroll.Value / 32768, "####0.0000")
End Sub
```

Listing 2-1: Event procedures for JogSpeedScroll  
horizontal scroll bar control

## THE JOG CONTROL

The components of the jog control are shown in Figure 2-16. The operation of the up, down, left, and right buttons is straight forward. The event procedures associated with the four buttons are all very similar. We will describe only the event procedures for the `RightButton` control (Listing 2-2). When the user positions the mouse over the control and presses the mouse button, a `MouseDown` event is generated and the `RightButton_MouseDown()` event procedure is executed. This event procedure issues a `VELMODE` command for axis-1 to Mx4. This command causes the shaft of the axis-1 servo-motor to begin rotating in the clockwise direction. The rotation speed is obtained from the `JogSpeedScroll` horizontal scroll bar discussed earlier. When the user releases the mouse button, a `MouseUp` event is generated and the `RightButton_MouseUp()` event procedure is executed. This procedure issues a `stop_axis` command to Mx4 for axis-1 causing the shaft of the axis-1 servo-motor to stop rotating.

## *Step-by-Step Application Development*

```
Sub RightButton_MouseDown (Button As Integer, Shift As Integer, X
As Single, Y As Single)

    'Issue velmode command to Mx4 to make axis-1 spin in the
positive
'direction at the speed specified by the the current value
'of the Horizontal scroll bar.
    velmode 1, MaxJogSpeed * JogSpeedScroll.Value / 32768

End Sub

Sub RightButton_MouseUp (Button As Integer, Shift As Integer, X As
Single, Y As Single)

    'Issue a stop command to Mx4
    stop_axis 1

End Sub
```

Listing 2-2: Event procedures for the RightButton control.

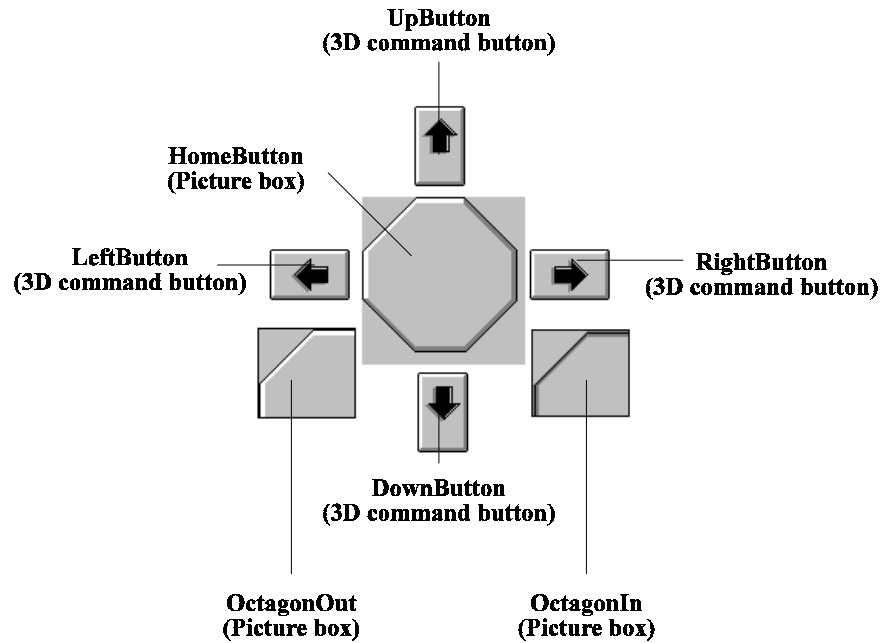


Figure 2-16. Components of the jog control

The operation of the `HomeButton` control is a little more complicated. This is because Visual Basic does not provide an "Octagonal Button" control. We had to use a picture control and emulate some of the behavior of a standard command button using Visual Basic code. The appearance of a command button control changes when it is depressed. To emulate this behavior, we needed two bitmap images: one showing the octagonal button in the up position and the other showing it in the down position. (We created these images using Corel Draw, although any other drawing program could have been used.) The idea is to change the bitmap displayed by the `HomeButton` picture box control while the program is running. To do this, two additional picture box controls (`OctagonIn` and `OctagonOut`) are necessary. The `Visible` property of these controls is set to `False` to prevent them from being displayed while the program is running. `OctagonIn` holds the bitmap image of the button in the down position. `OctagonOut` holds the image of the button in the up position. These bitmaps are setup at design time by assigning the appropriate .BMP file to their `Picture` property. To change the image displayed, we assign the `Picture` property of either `OctagonIn` or `OctagonOut` to the `Picture` property of the `HomeButton` control. The statement below, for example, will cause the `HomeButton` control to display the image of the button in the down position.

```
HomeButton.Picture = OctagonIn.Picture
```

The event procedures for the `HomeButton` control are shown in Listings 3 and 4. When the user positions the mouse over the home button and presses the mouse button, a `MouseDown` event is generated and the `HomeButton_MouseDown()` event procedure is executed. This procedure selects the button-down image. When the user releases the mouse button, a `MouseUp` event is generated and the `HomeButton_MouseUp()` event procedure is executed. This procedure selects the button-up image. When the user presses and releases the mouse button over a control, a `Click` event is generated. When the `HomeButton_Click()` event procedure is executed, an `AXMOVE` command is issued to Mx4 causing the servomotors to return to their home positions. The procedure must determine whether to use a positive or a negative velocity to return each axis to its home position. The procedure selects a positive axis-1 velocity, for example, only if the shaft must rotate in the positive direction to get to the home position. Low level Mx4 motion commands permit the desired motion for several axes to be specified at once. All of the motion control procedures in the DLL are single axis. The `BEGIN_RTC()` and `END_RTC()` procedures permit a multi-axis command to be built up from multiple calls to a single axis motion control procedure. We did this with `AXMOVE()` in `HomeButton_Click()` to illustrate the technique.

## *Step-by-Step Application Development*

```
Sub HomeButton_Click ()

    Dim x_velocity, y_velocity

    'Determine whether a positive or a negative
    'velocity is required to return axis-1 to the
    'home position
    If pos(1) < 0 Then
        x_velocity = MaxJogSpeed
    Else
        x_velocity = -MaxJogSpeed
    End If

    'Determine whether a positive or a negative
    'velocity is required to return axis-2 to the
    'home position
    If pos(2) < 0 Then
        y_velocity = MaxJogSpeed
    Else
        y_velocity = -MaxJogSpeed
    End If

    'Issue an axmove command to Mx4 to get both
    'axes back to the home position
    begin_RTC
        axmove 1, 1.9999, 0, x_velocity
        axmove 2, 1.9999, 0, y_velocity
    end_RTC

End Sub
```

**Listing 2-3:** HomeButton\_Click() event procedure

## *Step-by-Step Application Development*

```
Sub HomeButton_MouseDown (Button As Integer, Shift As Integer, X As
Single, Y As Single)

    'While mouse button is down display the "Octagon In" picture
    HomeButton.Picture = OctagonIn.Picture

End Sub

Sub HomeButton_MouseUp (Button As Integer, Shift As Integer, X As
Single, Y As Single)

    'The user has release the mouse button. Display the "Octagon
Out"
    'picture
    HomeButton.Picture = OctagonOut.Picture

End Sub
```

Listing 2-4: HomeButton\_MouseDown() and HomeButton\_MouseUp() event procedures.

## POSITION DISPLAY

The shaft positions of the x and y servo-motors are displayed in real-time using label controls. The controls making up the position display are illustrated in Figure 2-17. The timer control is configured to generate a `Timer` event every 100ms. When this event occurs, the `Timer1_Timer()` event procedure (Listing 2-5) is executed. This procedure updates the `Caption` properties of the label controls (`Axis1PositionLabel1` and `Axis2PositionLabel2`) used to display the shaft positions of the two motors.

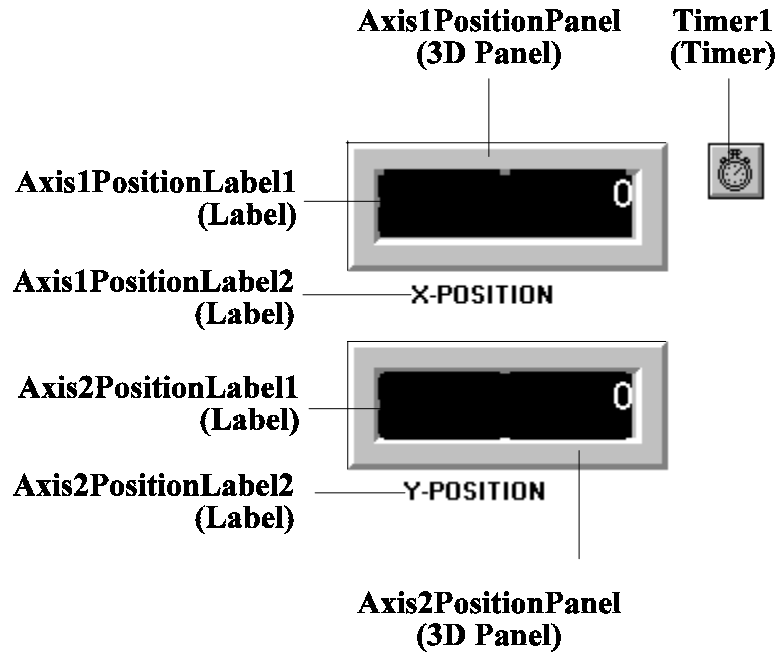


Figure 2-17 Components of the shaft position display

```
Sub Timer1_Timer ()  
  
    'The timer has expired. Update the axis-1 and axis-2  
    'position display  
    Axis1PositionLabel1.Caption = pos(1)  
    Axis2PositionLabel1.Caption = pos(2)  
  
End Sub
```

Listing 2-5: The Timer1\_Timer() event procedure

## INITIALIZATION

This application has only one form. This form is loaded automatically at startup. When the form is loaded, a Load event is generated and the Form\_Load() event procedure (Listing 2-6) is executed. This procedure is used to initialize Mx4 and set up the application. The listing is self-explanatory. A number of properties

of the JogSpeedScroll horizontal scroll bar control are initialized using the MaxJogSpeed constant. The location of the background panel on the form is computed here as well. This is done to allow the background panel to be in the center of the screen regardless of the screen's resolution or size.

```
Sub Form_Load ()

    Screen.MousePointer = 11 'Turn mouse pointer into hourglass

    'Make sure Mx4 is present
    If Left$(signature(), 3) <> "MX4" Then
        MsgBox "Mx4 Not Found"
    End
End If

reset_MX4 'reset Mx4

time_unit 1 'Let time unit be one Mx4 tick (200us)

'Setup Control Law for Axis-1
ctrl 1, 100, 5208, 5796, 1432 'Ki, Kp, Kf, Kd
maxacc 1, 1.9999
estop_acc 1, 1.9999
outgain 1, 0
KiLimit 1, 0

'Setup Control Law for Axis-2
ctrl 2, 100, 5208, 5796, 1432 'Ki, Kp, Kf, Kd
maxacc 2, 1.9999
estop_acc 2, 1.9999
outgain 2, 0
KiLimit 2, 0

'make sure axis 1 and 2 are stopped
begin_RTC
    stop_axis 1
    stop_axis 2
end_RTC

'Initialize the Jog Speed Control
JogSpeedScroll.Value = 1 / MaxJogSpeed * 32768
JogSpeedLabel1.Caption = Format(1, "####0.000")
JogSpeedScroll.SmallChange = .1 / MaxJogSpeed * 32768
JogSpeedScroll.LargeChange = 1 / MaxJogSpeed * 32768
```

Listing 2-6: Form\_Load() event procedure

*Step-by-Step Application Development*

```
'Position the background panel in the
'center of the window
Jog.Width = Screen.Width
Jog.Height = Screen.Height
Jog.Top = 0
Jog.Left = 0
BackgroundPanel.Top = (Jog.ScaleHeight -
BackgroundPanel.Height) / 2
BackgroundPanel.Left = (Jog.ScaleWidth - BackgroundPanel.Width)
/ 2

Screen.MousePointer = 0 'Return the mouse pointer to its normal
shape
```

Listing 2-6 (continued): `Form_Load()` event procedure.

# 3 USING MOTION COMMANDS IN VISUAL BASIC

This chapter provides a reference to the most commonly asked questions on how to use the Mx4 DLL commands in Visual Basic. Detailed information on all of the functions and commands are located in chapter 4 of this manual, *Function Reference*.

## OBTAINING ACCESS TO THE DLL FUNCTIONS

Before beginning any programming in Visual Basic which utilizes the functions of the DLL (MX495.DLL or MX4NT.DLL), the Visual Basic project must have access to the DLL. This access comes in the way of a special module that must be incorporated in the user's project. This module contains a Visual Basic declaration for each function in the DLL. It also defines a number of useful constants.

After installing the Mx4 DLL, the root directory of the installation will contain a file (MX495\_VB.TXT or MX4NT\_VB.TXT). This file contains the code for the required Visual Basic module.

To create the module in a Visual Basic project, the .txt file should be copied to the project directory with the .bas extension. For example, copy MX495\_VB.TXT to the project directory as MX4.BAS. In the Visual Basic environment, select Add Module from the Project menu. Select Existing, then select the MX4.BAS module, and Open. The module has been added to the project and should be visible in the Project Explorer window. Any of the functions discussed in this manual (see chapter 4, *Function Reference*) are now available to the project.

## REAL TIME COMMANDS

---

One of the primary purposes of the DLL is to allow the Visual Basic programmer to conveniently issue real time commands to Mx4. One category of Mx4 commands requires special consideration: multi-axis commands. Many motion control commands permit the desired motion for several axes to be specified at once. Such commands are useful for synchronizing the motion of two or more axes. All of the motion control functions in the DLL are, however, single axis. The `BEGIN_RTC` and `END_RTC` functions permit a multi-axis command to be built up from multiple calls to a single axis DLL function. The `AXMOVE` function illustrates this. A call to `AXMOVE` by itself will generate an `AXMOVE` RTC for the specified axis. To generate a two-axis `AXMOVE` RTC two calls to `AXMOVE` would be bracketed between calls to `BEGIN_RTC` and `END_RTC` as shown below.

```
BEGIN_RTC
    AXMOVE 1, 0.005, 234567, 4.00
    AXMOVE 2, 0.005, -3000, -3.50
END_RTC
```

While a multi-axis command is being built up, no commands are issued to Mx4. An RTC is issued only after the call to the `END_RTC` function. Keep in mind that the purpose of the `BEGIN_RTC` and `END_RTC` functions is to allow specific commands to be generated. Their generality is limited. All of the function calls bracketed by `BEGIN_RTC` and `END_RTC` must be of the same type. Not all commands are multi-axis.

## STATE VARIABLES

---

The state variables actual position [`POS`], actual velocity [`VEL`], and following error [`FERR`] are available to the programmer for all [`Mx4 : 4`][`Mx4 Octavia : 8`] axes. For example, to display the position, velocity, and following error of axis 6 to label controls `lblPos`, `lblVel`, and `lblEr`,

```
lblPos.Caption = POS (6)
lblVel.Caption = Format(VEL (6), "####0.000")
lblErr.Caption = FERR (6)
```

Note that the velocity display is formatted to include fractional numbers.

## DSPL VARIABLES

---

The DLL allows DSPL variables to be easily monitored as well as written to through the functions `CHANGE_VAR`, `MONITOR_VAR`, and `VAR`.

To change the value of DSPL variable `VAR82` to `10000.5`,

```
CHANGE_VAR 82, 10000.5
```

DSPL variables are monitored through a window in the Dual Port RAM which allows up to 4 variables to be “viewed” at the same time. Reading or monitoring a variable involves associating the DSPL variable with the specific DPR window (`MONITOR_VAR`), and then reading the DPR window (`VAR`). For example, set monitor window number 1 to `VAR44`, set monitor window number 2 to `VAR71`. Then read the variable values into temp storage `Temp1` and `Temp2`.

```
MONITOR_VAR 1, 44  
MONITOR_VAR 2, 71  
Temp1 = VAR (1)  
Temp2 = VAR (2)
```

After the above code is executed, `Temp1` contains the value of `VAR44`, and `Temp2` contains the value of `VAR71`.

## DSPL PROGRAM FUNCTIONALITY

---

With regards to DSPL programming applications, the Visual Basic programmer may require any or all of the following functionality,

- clearing a DSPL program [`CLEAR_DSPL`]
- downloading a DSPL program [`DOWNLOAD_DSPL`]
- setting a DSPL program for autostart execution [`AUTOSTART_DSPL`]
- initiating execution of a DSPL program [`START_DSPL`]
- signaling to a DSPL program from the host [`SIGNAL_DSPL`]
- terminating DSPL program execution [`STOP_DSPL`]

As an example, download a compiled DSPL program, test (remember, compiling a DSPL .hll file will create a .lod download file), which is located in the `c:\work` directory. Prior to downloading, clear the DSPL storage area. After the program has been downloaded, enable autostart execution.

### *Using Motion Commands In Visual Basic*

```
Dim sFileName As String
sFileName = "c:\work\test.lod"
CLEAR_DSPL
DOWNLOAD_DSPL sFileName
AUTOSTART_DSPL 1
```

To initiate execution of a downloaded DSPL program,

```
START_DSPL
```

Similarly, to signal to a DSPL program waiting at a WAIT\_UNTIL\_RTC line, or to terminate execution of a DSPL program,

```
SIGNAL_DSPL
STOP_DSPL
```

## USER-DEFINED UNITS

---

Not every motion application lends itself to pre-defined position and/or time units, and for that reason, the DLL allows the Visual Basic programmer to define both position [POSITION\_UNIT] and time [TIME\_UNIT] units. The default DLL position unit is counts, and the default DLL time unit is seconds.

Consider the case of an engraving application which lends itself to units of inches (where 1 inch is equal to 7390 counts) and milliseconds (msec); that is position in units of inches, velocity in units of inches/msec, and acceleration in units of inches/msec<sup>2</sup>. The following code sets these units for all subsequent accesses through the DLL,

```
POSITION_UNIT 7390
TIME_UNIT 1#/1000#
```

## INPUTS / OUTPUTS

---

The DLL includes functions which allow the user to manipulate Mx4's digital I/O. Inputs may be read via the `MX4_INPUT` function while outputs can be controlled with the `OUTP_ON` and `OUTP_OFF` functions.

For example, if IN19 is on, then turn on OUT4, otherwise turn off OUT20 and OUT21.

```
If MX4_INPUT (19) Then
    OUTP_ON 4
Else
    BEGIN_RTC
        OUTP_OFF 20
        OUTP_OFF 21
    END_RTC
End If
```

## DATA TABLE DOWNLOADING

---

Downloading data tables to Mx4 is an important feature of the DLL for many applications. Data tables include cam tables [`DOWN_CAM`], cubic spline tables [`DOWN_CUBIC`], position and velocity compensation tables [`DOWN_POS` and `DOWN_VEL`], and DSPL table\_p / table\_v tables [`DOWN_POINTS`]. Downloading a table is fundamentally the same, regardless of the type.

For example, download to Mx4 a cam table consisting of 10 master position, slave position pairs to the cam table starting at cam index 100. The cam table `camex.dat` is located in the `c:\work` directory.

```
Dim mdata(10) As Single
Dim sdata(10) As Single
Dim I As Integer
Open "c:\work\camex.dat" For Input As #1
For I = 0 To 9
    Input #1, mdata(I)
    Input #1, sdata(I)
Next I
DOWN_CAM mdata(0), sdata(0), 10, 100
```

## Mx4 DUAL PORT RAM ACCESS

---

Even with all of the functionality provided by the DLL, the Visual Basic programmer may want to read from and write directly to the Dual Port RAM. The `R_1BYTE`, `R_2BYTE`, and `R_4BYTE` functions provide the read access while the `W_1BYTE`, `W_2BYTE`, and `W_4BYTE` functions provide the write access to the DPR.

As an example of the DPR access functions, consider the application which requires reading the ADC1 value from the Mx4 (see *Acc4*, and *Mx4 User's Guide* for information regarding ADC1-4 analog feedback values updated to the DPR). The application will need to monitor the Mx4 access byte until it is cleared, set the host DPR access byte for the ADC DPR window, as well as read the ADC1 value from DPR addresses 502h, 503h.

```
While (R_1BYTE (&H500) <> 0)
Wend
W_1BYTE &H501, 1
Temp = R_2BYTE (&H502)
W_1BYTE &H501, 0
```

## BUS COMMUNICATION

---

When the ISA bus Mx4 controller is used in a system, it (the Dual Port RAM) resides at a specified bus address, such as 0xD0000. If multiple Mx4 controllers are used in the same bus address space, it is necessary for the host application to address each of the Mx4's at their unique bus address. The DLL functions `CURRENTCARDADDRESS` and `CHANGECARDADDRESS` allow the Visual Basic programmer to accomplish these tasks.

For example, read back the current card address which the DLL is set to, then change the address to 0xD8000.

```
MsgBox "communicating at 0x" & Hex(CURRENTCARDADDRESS)
CHANGECARDADDRESS &HD8000
```

## SERIAL COMMUNICATION

---

Rather than across a bus, serial communication to RS-232/RS-485 equipped Mx4 controllers takes place across a comm port. The comm port setting is made when the DLL (or Mx4pro) is first installed. To change the comm port setting, the `CHANGECOMMPORTSETTING` function is used. For example, to change the comm port used to communicate with a serial Mx4 to comm port 3,

```
CHANGECOMMPORTSETTING 3
```

RS-485 serial communication allows multiple Mx4 controllers to be connected to the line as different nodes (up to a maximum of 16 nodes per line). If the host program must communicate with more than one Mx4 per line, the node address must be changed. To change the serial communication node address to node 10,

```
CHANGESLAVENODEADDRESS 10
```

The complete set of serial communication-related functions are listed in chapter 4, *Function Reference*.

## VISUAL BASIC EXAMPLES

---

Included with the DLL installation are three complete Visual Basic example applications. The examples are located in the `VB_Exam` folder in the installation root directory.

### EXAMPLE 1

This example demonstrates the operation of two motors in a velocity controlled mode (`VELMODE`). The operation is initiated and ended by a Start and a Stop button. The operating speed for both motors is 5 encoder counts per 200  $\mu$ s. By means of this program you will learn how to:

- 1) Setup servo loops by specifying their gains and maximum acceleration
- 2) Specify the time and distance units for speed and other motion dynamics

### *Using Motion Commands In Visual Basic*

- 3) Check for out-of-range parameter errors
- 4) Operate motors in velocity mode
- 5) Display the positions, velocities and errors for both motors
- 6) Use Multi-axis motion commands in Visual Basic.

## EXAMPLE 2

This example demonstrates the operation of two motors in Jog mode. The speed for both motors is adjusted by a horizontal slide bar. Motion in each direction is initiated by depressing an arrow key in a proper direction. Depressing the middle key brings both motors to their starting positions. Finally, positions of both axes are numerically displayed. By means of this program, you will learn how to:

- 1) Use the slide bar in conjunction with real time change of motion parameters (e.g. speed)
- 2) Use the mouse in conjunction with the real time commands.

## EXAMPLE 3

The third example application allows the user to download a DSPL program, and start and stop the execution of the program. The application also demonstrates changing the Mx4 card address for multi-card applications. Axis 1 and axis 2 state variables position, velocity, and error are also displayed to the screen.

# 4 FUNCTION REFERENCE

## REFERENCE

---

ALLPOS .....	4-15
ALLVEL .....	4-16
ALLERR .....	4-17
ALLVAR .....	4-18
AUTOSTART_DSPL .....	4-19
AXMOVE .....	4-20
AXMOVE_S .....	4-23
AXMOVE_T .....	4-25
BEGIN_RTC .....	4-27
BEGINDLLCRITICALSECTION .....	4-28
BTRATE .....	4-29
CAM .....	4-31
CAM_OFF .....	4-34
CAM_OFF_ACC .....	4-35
CAM_POINT .....	4-36
CAM_POS .....	4-38
CAM_PROBE .....	4-40
CHANGECARDADDRESS .....	4-43
CHANGECOMMPORTSETTING .....	4-44
CHANGESLAVENODEADDRESS .....	4-45
CHANGE_VAR .....	4-46
CLEAR_CUBIC .....	4-47
CLEAR_DSPL .....	4-48
CLEAR_POINTS .....	4-49
CLEAR_POS_TABLE .....	4-50
CLEAR_VEL_TABLE .....	4-51
COMMUNICATIONSLOST .....	4-52
CTRL .....	4-53
CTRL_KA .....	4-56
CUBIC_INT .....	4-57

*Function Reference*

CUBIC_RATE .....	4-59
CUBIC_SCALE .....	4-63
CURRENTCARDADDRESS .....	4-65
CURR_LIMIT .....	4-66
CURR_OFFSET .....	4-68
CURR_PID .....	4-69
DDAC .....	4-70
DISABL_INT .....	4-72
DISABL2_INT .....	4-74
DOWNLOAD_DSPL .....	4-76
DOWN_CAM .....	4-77
DOWN_CUBIC .....	4-78
DOWN_POINTS .....	4-79
DOWN_POS .....	4-80
DOWN_VEL .....	4-81
ENCOD_MAG .....	4-82
END_RTC .....	4-84
ENDDLLCRITICALSECTION .....	4-85
EN_BUFBRK .....	4-86
EN_ENCFLT .....	4-88
EN_ERR .....	4-90
EN_ERRHLT .....	4-92
EN_INDEX .....	4-94
EN_MOTCP .....	4-96
EN_POSBRK .....	4-98
EN_PROBE .....	4-100
ESTOP_ACC .....	4-102
FERR .....	4-104
FLUX_CURRENT .....	4-105
GEAR .....	4-107
GEAR_OFF .....	4-108
GEAR_OFF_ACC .....	4-109
GEAR_POS .....	4-110
GEAR_PROBE .....	4-112
GETCOMMINSTCOUNT .....	4-114
GETCOMMTYPE .....	4-115
GETCURRENTNODEADDRESS .....	4-116
GETNUMBEROFAXES .....	4-117
INP_STATE .....	4-118
INT5MS .....	4-119
KILIMIT .....	4-120

*Function Reference*

LOW_PASS (option).....	4-122
MAXACC.....	4-125
MONITOR_VAR.....	4-127
MOTOR_PAR.....	4-128
MOTOR_TECH.....	4-129
MX4_CLEAR.....	4-130
MX4_INPUT.....	4-132
MX4_ISTAT.....	4-133
NOTCH (option).....	4-135
OFFSET.....	4-138
OUTGAIN.....	4-140
OUTP_OFF.....	4-142
OUTP_ON.....	4-143
OVERRIDE.....	4-144
PARREAD.....	4-145
POS.....	4-148
POSBRK_OUT.....	4-149
POSITION_UNIT.....	4-153
POS_PRESET.....	4-154
POS_SHIFT.....	4-155
PWM_FREQ.....	4-156
REL_AXMOVE.....	4-157
REL_AXMOVE_S.....	4-158
REL_AXMOVE_T.....	4-160
REL_AXMOVE_SLAVE.....	4-162
RESET_MX4.....	4-164
RESETCOMMUNICATIONS.....	4-165
R_1BYTE.....	4-166
R_2BYTE.....	4-167
R_4BYTE.....	4-168
R_NBYTE.....	4-169
SIGNAL_DSPL.....	4-170
SIGNATURE.....	4-171
START.....	4-172
START_DSPL.....	4-174
STEPPER_ON.....	4-175
STOP_AXIS.....	4-176
STOP_DSPL.....	4-178
SYNC.....	4-179
TABLE_SEL.....	4-181

*Function Reference*

TIME_UNIT.....	4-182
TRQ_LIMIT.....	4-183
VAR.....	4-184
VEC.....	4-185
VECCHG.....	4-186
VEL.....	4-188
VELMODE.....	4-189
VIEWVEC.....	4-191
VX4_BLOCK.....	4-192
W_1BYTE.....	4-193
W_2BYTE.....	4-194
W_4BYTE.....	4-195
W_NBYTE.....	4-196

## FUNCTION SUMMARY

---

The Mx4 Visual Basic programming function set includes many commands and programming tools. The functions consist of sixteen major command categories. Each category extends the power and flexibility of Mx4 in general areas of motion control.

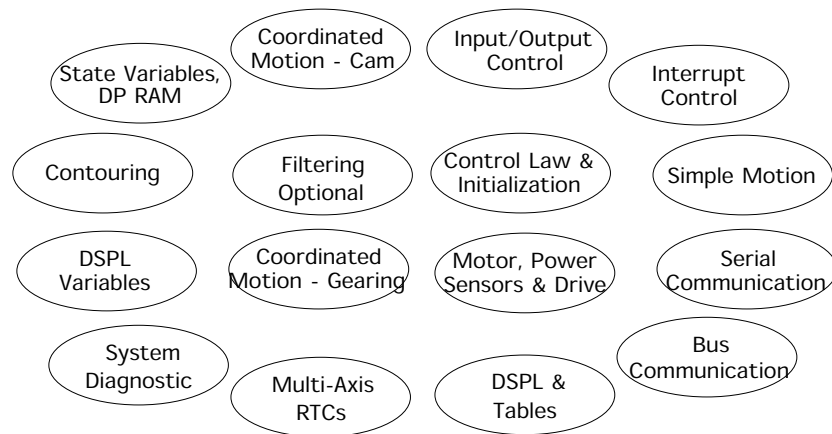


Fig. 4-1: Function Categories

## CONTROL LAW & INITIALIZATION

Control gains, system parameters, time, position, and velocity units all fall in this category.

COMMAND	DESCRIPTION
CTRL	Position, velocity loop control law parameters
CTRL_KA	Program an acceleration feed-forward gain
ESTOP_ACC	Specify emergency stop maximum acceleration
KILIMIT	Integral gain limit
MAXACC	Specify maximum acceleration
OFFSET	Amplifier offset cancellation
OUTGAIN	Position loop output gain
POS_PRESET	Preset position counters
POS_SHIFT	Position counter reference shift
POSITION_UNIT	Specify user-position units
RESET_MX4	Reset Mx4 controller card
SIGNATURE	Check Mx4 controller signature
STEPPER_ON	Select stepper / servo axes
SYNC	Define Mx4 master/slave status
TIME_UNIT	Specify user-time units
TRQ_LIMIT	Specify a torque limit

## SIMPLE MOTION

The instructions within this category control the torque, velocity, and position of one or multiple axes with a trapezoidal profile. The commands in this category may be classified as open and closed loop.

COMMAND	DESCRIPTION
AXMOVE	Trapezoidal axis move
AXMOVE_S	s-curve axis move
AXMOVE_T	Time based axis move
DDAC	Direct DAC command (open loop)
REL_AXMOVE	Relative position axis move
REL_AXMOVE_S	Relative s-curve axis move
REL_AXMOVE_T	Time based relative axis move
STOP	Stops the motion
VELMODE	Velocity mode

## INPUT / OUTPUT CONTROL

These functions are used to control and query the status of the Mx4 discrete inputs and outputs.

COMMAND	DESCRIPTION
INP_STATE	Configure logic state of inputs
MX4_INPUT	Read status of Mx4 inputs
OUTP_OFF	Set status of outputs to low logic level
OUTP_ON	Set status of outputs to high logic level
POSERK_OUT	Set outputs after position breakpoint interrupt

## STATE VARIABLES, DP RAM

These functions provide Dual Port RAM utilities for reading from and writing to the Mx4 controller.

COMMAND	DESCRIPTION
ALLPOS	Read 4 actual position state variables
ALLVEL	Read 4 actual velocity state variables
ALLERR	Read 4 following error state variables
ALLVAR	Read 4 DSPL variables
FERR	Read Mx4 following error state variables
POS	Read Mx4 actual position state variables
R_1BYTE	Read single byte from specified DPR address offset
R_2BYTE	Read single word from specified DPR address offset
R_4BYTE	Read long word from specified DPR address offset
R_NBYTE	Read n bytes from specified DPR address offset
VEL	Read Mx4 actual velocity state variable
W_1BYTE	Write single byte to specified DPR address offset
W_2BYTE	Write single word to specified DPR address offset
W_4BYTE	Write long word to specified DPR address offset
W_NBYTE	Write n bytes to specified DPR address offset

## DSPL VARIABLES

The following functions allow read/write functionality to DSPL variables (VAR1-VAR128).

COMMAND	DESCRIPTION
CHANGE_VAR	Write value to specified DSPL variable
MONITOR_VAR	Select DSPL variable to be read
VAR	Read DSPL variable

## SYSTEM DIAGNOSTIC

In addition to Mx4's full diagnostic reporting via the DPR, the host may examine internal Mx4 parameters and provide debug support with the PARREAD RTC.

COMMAND	DESCRIPTION
PARREAD	Mx4 system parameter readback

## MULTI-AXIS RTCs

These functions make it possible for multi-axis commands to be executed simultaneously from Visual Basic.

COMMAND	DESCRIPTION
BEGIN_RTC	Begin multi-axis command
END_RTC	End multi-axis command

## DLL SYNCHRONIZATION

These functions make it possible to synchronize access to the DLL in a multi-threaded program.

COMMAND	DESCRIPTION
BEGINDLLCRITICALSECTION	Acquire mutex lock for DLL
ENDDLLCRITICALSECTION	Release mutex lock for DLL

## DSPL & TABLES

These functions are used to set up and control the execution of a DSPL program on the Mx4.

COMMAND	DESCRIPTION
AUTOSTART_DSPL	Start DSPL execution at power-up/reset
CLEAR_DSPL	Clear DSPL program from Mx4 memory
CLEAR_POINTS	Clear points table
CLEAR_POS_TABLE	Clear position compensation table
CLEAR_VEL_TABLE	Clear velocity compensation table
DOWNLOAD_DSPL	Download compiled DSPL program to Mx4
DOWN_POINTS	Download points table
DOWN_POS	Download position compensation table
DOWN_VEL	Download velocity compensation table
SIGNAL_DSPL	Signal the DSPL program
START_DSPL	Begin execution of DSPL program
STOP_DSPL	Halt DSPL program execution
TABLE_SEL	Select compensation table

## CONTOURING

The Mx4 includes contouring commands for users who need to generate arbitrary motion profiles. In these applications, a host computer generates position and velocity data points for a complex contouring path in a periodic basis. In CNC and robotics applications, motion trajectories may be computed in real time. These trajectories are transmitted to Mx4 in blocks of position/(velocity) points. The ring buffer area of Mx4's dual port RAM is the storage area for these motion blocks. Mx4 performs high order interpolation on all these points and executes the trajectory path on a point to point basis.

COMMAND	DESCRIPTION
BTRATE	Block transfer rate for 2 <sup>nd</sup> order contour
CLEAR_CUBIC	Clear cubic data table
CUBIC_INT	Start the internal cubic spline table
CUBIC_RATE	Set cubic spline point transfer rate
CUBIC_SCALE	Scales, shifts position points
DOWN_CUBIC	Download cubic data table
OVERRIDE	Set feedrate override for LINEAR / CIRCLE
START	Start contouring motion
VECCHG	Contouring vector change

## BUS COMMUNICATION

These functions allow bus (ISA) communication parameters to be modified.

COMMAND	DESCRIPTION
CHANGECARDADDRESS	Sets the Mx4 bus address pointer
CURRENTCARDADDRESS	Returns the current Mx4 bus address pointer

## MOTOR, POWER, SENSORS AND DRIVE (available with Vx4++ option only)

Mx4 allows the option of an add-on multi-DSP drive control card called Vx4++. The drive control option performs all of the signal processing functions of servo amplifier control boards. Vx4++ controls include commutation, current loops, field current, torque current, current limiting, pulse-width modulation frequency, etc. This board makes the Mx4 control unit compatible with all power devices, industrial motors, and a majority of sensors on the market.

COMMAND	DESCRIPTION
CURR_LIMIT	Current limit setting
CURR_OFFSET	Current loop offset adjustment
CURR_PID	Program current loop control law parameters
ENCOD_MAG	Specify encoder lines, motor poles, comm. option
FLUX_CURRENT	Bipolar field flux value
MOTOR_PAR	Set the motor parameter
MOTOR_TECH	Define the motor technology
PWM_FREQ	Set output PWM signal frequency
VEC	read Vx4++ VIEWVEC parameter
Vx4_BLOCK	Block further instructions to Vx4++
VIEWVEC	Specify Vx4++ parameters to view

## COORDINATED MOTION - GEARING

Multi-axis motion control applications require synchronization of two or more axes in a coordinated task. In addition to the electronic gearing master-slaving technique, compensation tables also help users specify their own application specific "slaving function".

COMMAND	DESCRIPTION
GEAR	Unconditional 'electronic' gearing
GEAR_OFF	Disengage 'electronic' gearing
GEAR_OFF_ACC	Turns electronic gearing off and halt slave(s)
GEAR_POS	'electronic' gearing based on position value
GEAR_PROBE	'electronic' gearing based on external interrupt
REL_AXMOVE_SLAVE	Superimposes a relative axis move onto a slave engaged in gearing

## COORDINATED MOTION - CAM

Multi-axis motion control applications require synchronization of two or more axes in a coordinated task. A subset of table oriented master/slaving is what is known as "electronic cam".

COMMAND	DESCRIPTION
CAM	Turns electronic cam on
CAM_OFF	Turns only electronic cam off
CAM_OFF_ACC	Turns electronic cam off and halts slave(s)
CAM_POINT	Place cam point into cam table
CAM_POS	Turns electronic cam on at a specified position
CAM_PROBE	Turns electronic cam on after EN_PROBE is set
DOWN_CAM	Download cam data points

## SERIAL COMMUNICATION

These functions allow serial (RS-232, RS-485) communication parameters to be modified.

COMMAND	DESCRIPTION
CHANGECOMMPORTSETTING	Change comm port setting
CHANGESLAVENODEADDRESS	Change Mx4 slave node address
COMMUNICATIONSLOST	Check for communication lost
GETCOMMINSTCOUNT	Return number of DLL instances connected
GETCOMMTYPE	Return comm type
GETCURRENTNODEADDRESS	Return current Mx4 slave node address
RESETCOMMUNICATIONS	Reset serial communications

## INTERRUPT CONTROL

The Mx4 DSPL includes a comprehensive set of instructions to handle interrupts. There are many system conditions that require the host's and/or DSPL program's immediate attention for an executive (or system-level) decision. Some interrupts will be issued concurrently requiring immediate action by the Mx4. The complete set of interrupts provided by Mx4 facilitates data reporting to the host for issues of system-level significance.

COMMAND	DESCRIPTION
DISABL_INT	Disable the interrupts
DISABL2_INT	Disable the interrupts
EN_BUFBRK	Contouring buffer breakpoint interrupt enable
EN_ENCFLT	Encoder fault interrupt
EN_ERR	Following error interrupt enable
EN_ERRHLT	Following error / halt interrupt enable
EN_INDEX	Index pulse interrupt enable
EN_MOTCP	Motion complete interrupt enable
EN_POSBRK	Position breakpoint interrupt enable
EN_PROBE	General purpose ext probe interrupt enable
INT5MS	5msec sampling interrupt
MX4_CLEAR	Clear interrupt conditions
MX4_ISTAT	Test for interrupt conditions

## FILTERING (OPTIONAL)

COMMAND	DESCRIPTION
LOW_PASS	Implement low pass filter at controller output
NOTCH	Implement notch filter at controller output

## FUNCTION LISTING

---

The function reference is listed with syntax and data types specific to Visual Basic. The command listing follows this format:

<b>FUNCTION</b>	indicates the command function
<b>SYNTAX</b>	proper command syntax
<b>ARGUMENTS</b>	command arguments (if any) are defined
<b>DESCRIPTION</b>	explanation of command operation, functionality
<b>SEE ALSO</b>	listing of related commands
<b>APPLICATION</b>	some helpful suggestions as to for which applications a command may be useful
<b>EXAMPLE</b>	an example illustrating the command in use

## ALLPOS

---

**FUNCTION** Read four state variables at one time

**SYNTAX** ALLPOS flag, pos1, pos2, pos3, pos4

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

flag int value selecting which set of axis will be read

flag = 1 reads axis 1-4

flag = 2 reads axis 5-8

state1 double return values of positions for axis set

state2

state3

state4

### DESCRIPTION

The function reads a set of four positions at a time. The first argument selects the set to be read. The remaining arguments are passed by reference, and the position values are returned in ascending order.

**SEE ALSO** ALLVEL, ALLERR, ALLVAR

### EXAMPLE

Read axis 1-4.

```
ALLPOS 0, Axis1, Axis2, Axis3, Axis4
```

## ALLVEL

---

**FUNCTION** Read four state variables at one time

**SYNTAX** ALLVEL flag, vel1, vel2, vel3, vel4

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

flag int value selecting which set of axis will be read

flag = 1 reads axis 1-4

flag = 2 reads axis 5-8

state1 double return values of velocity for axis set  
state2  
state3  
state4

### DESCRIPTION

The function reads a set of four velocities at a time. The first argument selects the set to be read. The remaining arguments are passed by reference, and the position values are returned in ascending order.

**SEE ALSO** ALLPOS, ALLERR, ALLVAR

### EXAMPLE

Read axis 1-4.

```
ALLVEL 0, Axis1, Axis2, Axis3, Axis4
```

## ALLERR

---

**FUNCTION** Read four state variables at one time

**SYNTAX** ALLERR flag, pos1, pos2, pos3, pos4

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

flag int value selecting which set of axis will be read

flag = 1 reads axis 1-4

flag = 2 reads axis 5-8

state1 double return values of errors for axis set

state2

state3

state4

### DESCRIPTION

The function reads a set of four errors at a time. The first argument selects the set to be read. The remaining arguments are passed by reference, and the position values are returned in ascending order.

**SEE ALSO** ALLPOS, ALLVEL, ALLVAR

### EXAMPLE

Read axis 1-4.

```
ALLPOS 0, Axis1, Axis2, Axis3, Axis4
```

## ALLVAR

---

**FUNCTION**     Read four state variables at one time

**SYNTAX**        ALLVAR var1, var2, var3, var4

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

var1            double return values of variables

var2

var3

var4

### DESCRIPTION

The function reads the set of four DSPL variables at the same time. The DSPL variables to be read are selected by the monitor\_var function.

**SEE ALSO**        ALLPOS, ALLVEL, ALLERR, MONITOR\_VAR

### EXAMPLE

Read monitored variables

```
ALLVAR VAR1, VAR2, VAR3, VAR4
```

## AUTOSTART\_DSPL

---

**FUNCTION** Start DSPL Execution at Power-Up/Reset

**SYNTAX** AUTOSTART\_DSPL flag

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

Flag long value flag enabling or disabling the autostart option

Flag = 0 autostart is disabled

Flag = 1 autostart is enabled

### DESCRIPTION

The DSPL autostart feature requires an Mx4 controller with the optional battery-backup memory. The autostart feature allows Mx4 to begin DSPL program execution immediately after power-up. A DSPL program must have previously been loaded into Mx4's battery-backup memory before the AUTOSTART\_DSPL command is used. Once an AUTOSTART\_DSPL command has been executed by Mx4, Mx4 will remain in the specified (enable / disable) autostart state until it executes another AUTOSTART\_DSPL command; even after power-down. The Mx4 (with the battery-backup memory option) is shipped from the factory with the autostart feature disabled.

**SEE ALSO** CLEAR\_DSPL, DOWNLOAD\_DSPL, SIGNAL\_DSPL, START\_DSPL, STOP\_DSPL

### EXAMPLE

Enable the DSPL autostart option.

```
AUTOSTART_DSPL (1)
```

## AXMOVE

---

**FUNCTION** Axis Move with Trapezoidal Trajectory

**SYNTAX** AXMOVE axis, acc, pos, vel

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

axis	long value specifying the axis
acc	positive double precision value specifying the maximum halting acceleration (deceleration) $0 \leq acc_x \leq 1.999969 \text{ counts}/(200\mu\text{s})^2$
pos	double precision target position $-2147483648 \leq pos_x \leq 2147483647 \text{ counts}$
vel	positive double precision target velocity $0 \leq vel_x \leq 255.99998 \text{ counts}/200\mu\text{s}$

### DESCRIPTION

The AXMOVE command allows for trapezoidal command generation with specified endpoint position, slew rate velocity, and acceleration for each axis. This command is suitable for linear moves.

**SEE ALSO** AXMOVE\_S, AXMOVE\_T, REL\_AXMOVE, REL\_AXMOVE\_S, REL\_AXMOVE\_T, STOP

## AXMOVE cont.

---

### APPLICATION

This command can be used in almost any imaginable motion control application. Applications may benefit from this command any time there is a need for a linear move from point A to point B in a multi-dimensional space. To name a few applications: pick and place robots (e.g., in component insertion), rapid traverse (e.g., in machining), and master/slaving (e.g., in paper processing and packaging) applications.

#### **Command Sequence Example**

```

MAXACC ( ) ;set the maximum accel. so system can be stopped
CTRL ( ) ;set the gain values
KILIMIT ( )
AXMOVE ( ) ;run system in axis move (linear trapezoidal) mode
:
EN_MOTCP ( ) ;enable motion complete
;upon the completion of this (command) trajectory
;Mx4 generates motion complete interrupt

```

### EXAMPLE 1

Assuming current positions of zero for axes 1 and 2, we want to move axis 1 to the target position of 234567 and axis 2 to the target position of -3000 counts. Let's also assume that we want this move to be accomplished with the slew rate velocity of 4.0 counts/200 $\mu$ s for axis 1 and 3.50 counts/200 $\mu$ s for axis 2, and an acceleration of 0.005 counts/(200  $\mu$ s)<sup>2</sup> for both axes.

```

BEGIN_RTC
    AXMOVE 1, 0.005, 234567, 4.0
    AXMOVE 2, 0.005, -3000, 3.5
END_RTC

```

## **AXMOVE cont.**

---

### **EXAMPLE 2**

The user can issue a new axis move command before the motion of the previous `AXMOVE` command is completed. For example, assume the `AXMOVE` command of Example 1 is executed. Now, the DSPL Motion program 'decides' to stop axis two at a new target position of `-50000` counts with a new slew rate of `8.0` counts/200 $\mu$ s and a new acceleration of `0.035` counts/(200 $\mu$ s)<sup>2</sup>. While the `AXMOVE` of Example 1 is in progress, the DSPL Motion program issues the new command.

```
AXMOVE 2, 0.035, -50000, 8.0
```

## AXMOVE\_S

---

**FUNCTION** S-Curve Axis Move with Trapezoidal Trajectory

**SYNTAX** AXMOVE\_S axis, acc, pos, vel

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

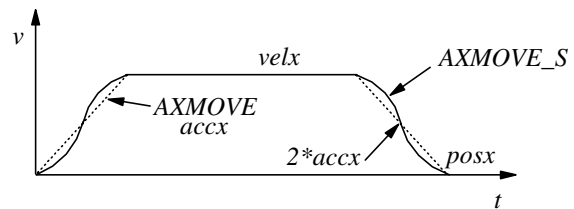
axis	long value specifying the axis
acc	positive double precision value specifying the maximum halting acceleration (deceleration) $0 \leq acc_x \leq 1.999969 \text{ counts}/(200\mu\text{s})^2$
pos	double precision target position $-2147483648 \leq pos_x \leq 2147483647 \text{ counts}$
vel	positive double precision target velocity $0 \leq vel_x \leq 255.99998 \text{ counts}/200\mu\text{s}$

### DESCRIPTION

The AXMOVE\_S command allows for s-curve command generation with specified endpoint position, slew rate velocity, and acceleration for each axis. This command is suitable for linear moves where s-curve acceleration is desired.

## AXMOVE\_S cont.

---



The figure above illustrates the velocity profile of the AXMOVE\_S along with the linear velocity ramp of the AXMOVE command. With AXMOVE\_S, the acceleration will reach a value of 2\*accx for a maximum (see above figure).

**SEE ALSO**    AXMOVE,    AXMOVE\_T,    REL\_AXMOVE,    REL\_AXMOVE\_S,  
REL\_AXMOVE\_T, STOP

### APPLICATION

Refer to *DSPL Application Programs*.

### EXAMPLE 1

Assuming current positions of zero for axes 1 and 2, we want to move axis 1 to the target position of 200000 counts and axis 2 to the target position of -3000 counts. Let's also assume that we want this move to be accomplished with the slew rate velocity of 4.0 counts/200  $\mu$ s for axis 1 and 2.0 counts/200  $\mu$ s for axis 2. Use an acceleration reference of 0.05 counts/(200  $\mu$ s)<sup>2</sup> for both axes.

```
BEGIN_DSPL
  AXMOVE_S 1, 0.05, 200000, 4.0
  AXMOVE_S 2, 0.05, -3000, 2.0
END_RTC
```

## AXMOVE\_T

---

**FUNCTION** Time-Based Axis Move with Trapezoidal Trajectory

**SYNTAX** AXMOVE\_T axis, acc, pos, tm

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

axis	long value specifying the axis
acc	positive double precision value specifying the maximum halting acceleration (deceleration) $0 \leq acc_x \leq 1.999969 \text{ counts}/(200\mu\text{s})^2$
pos	double precision target position $-2147483648 \leq pos_x \leq 2147483647 \text{ counts}$
tm	positive double precision motion time $0 \leq tm_x \leq 5000000 (200\mu\text{s})$

### DESCRIPTION

The AXMOVE\_T commands allow for trapezoidal command generation with specified endpoint position, acceleration, and time to complete the move for each axis. This command is suitable for linear moves where endpoint position and motion time are the specifying parameters.

## AXMOVE\_T cont.

---

The AXMOVE\_T command is similar to AXMOVE, with the exception that the velocity argument is replaced with a time argument. AXMOVE\_T will automatically calculate a suitable slew rate velocity to achieve the programmed endpoint position in the programmed amount of time, following a trapezoidal velocity profile (similar to AXMOVE).

**SEE ALSO**      AXMOVE,      AXMOVE\_S,      REL\_AXMOVE,      REL\_AXMOVE\_S,  
REL\_AXMOVE\_T, STOP

### APPLICATION

Refer to *DSPL Application Programs*.

### EXAMPLE

Move axis 1 to the target position of 10000 counts and axis 3 to the target position of 3599 counts. Let's assume that we want this move to be accomplished with the acceleration reference of 0.56 counts/(200  $\mu$ s)<sup>2</sup> and a time of 50msec (250\*200 $\mu$ sec) for both axes.

```
BEGIN_RTC
    AXMOVE_T 1, 0.56, 10000, 250
    AXMOVE_T 3, 0.56, 3599, 250
END_RTC
```

## BEGIN\_RTC

---

**FUNCTION**     Begin Multi-Axis Command

**SYNTAX**        BEGIN\_RTC

### ARGUMENTS

None

### DESCRIPTION

A number of RTCs have a large and variable number of arguments. These are mostly motion control RTCs which permit the desired motion for several axes to be specified at once. All of the motion control functions in the DLL are single axis. The `BEGIN_RTC` and `END_RTC` functions permit a multi-axis RTC to be built-up from multiple calls to the single axis RTC function. The `AXMOVE` function illustrates this. A call to `AXMOVE` by itself will generate an `AXMOVE` RTC for the specified axis. To generate a two-axis `AXMOVE` RTC, two calls to `AXMOVE` would be bracketed between calls to `BEGIN_RTC` and `END_RTC`.

**SEE ALSO**        END\_RTC

### APPLICATION

Multi-axis commands are needed when the trajectories of two or more axes must be synchronized.

### EXAMPLE

This example illustrates how `BEGIN_RTC` and `END_RTC` can be used to issue a two axis `AXMOVE` command to Mx4. Assuming current positions of zero for axes 1 and 2, we want to move axis 1 to the target position of 234567 and axis 2 to the target position of -3000 counts. Let's also assume that we want this move to be accomplished with the slew rate velocity of 4.0 counts/200 $\mu$ s for axis 1 and -3.50 counts/200 $\mu$ s for axis 2, and an acceleration of 0.005 counts/(200  $\mu$ s)<sup>2</sup> for both axes.

```
BEGIN_RTC
    AXMOVE  1, 0.005, 234567, 4.00
    AXMOVE  2, 0.005, -3000, -3.50
END_RTC
```

## BEGINDLLCRITICALSECTION

---

**FUNCTION**     Begin critical section

**SYNTAX**       long timeout

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

timeout        long value that specifies the time in milliseconds to wait for access to the DLL

### DESCRIPTION

This command gets a mutex lock for exclusive access to the DLL. The command should be used when multiple threads are accessing the DLL, and it should be used before every function call to the DLL. The function ENDDLLCRITICAL SECTION releases the mutex lock.

**SEE ALSO**     ENDDLLCRITICALSECTION

### APPLICATION

The function is used in multithreaded applications to provide synchronization for DLL access and maintain a consistent state in the DLL. The return value will indicate if it has successfully acquired the mutex lock.

### EXAMPLE

```
var1 = BEGINDLLCRITICALSECTION(100)
if var1 == ERR_OK then
BEGIN_RTC
    AXMOVE           1,     0.005,     234567,     4.00
    AXMOVE           2,     0.005,     -3000,     -3.50
END_RTC
ENDDLLCRITICALSECTION
Endif
```

## BTRATE

---

**FUNCTION** Set 2nd Order Contour Block Transfer Rate

**SYNTAX** BTRATE m

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

m long value selects the block transfer rate for all of the axes.

m=0	block transfer rate is 5 ms per point
m=1	block transfer rate is 10 ms per point
m=2	block transfer rate is 15 ms per point
m=3	block transfer rate is 20 ms per point

### DESCRIPTION

This command sets the 2nd order contouring block transfer rate for the system. For example, if the block transfer rate is set at 10 ms, the time interval between each point in the ring buffer is '10 ms' (e.g., the DSP will interpolate each point for 10 ms).



**Note 1:** The host should not adjust the block transfer rate when contouring is in process.



**Note 2:** The default block transfer rate is set at 5 ms per point.

**SEE ALSO** CUBIC\_RATE

## **BTRATE cont.**

---

### **APPLICATION**

This command is useful in 2nd order contouring applications. Depending on the capability of the host processor, position/velocity points on multi-dimensional trajectories may be broken down to the points that (timewise) may be near or far from each other. Clearly, slower CPUs are capable of breaking down geometries to position and velocity points that are widely spaced in time. This instruction makes the time interval in between the two adjacent points (in contouring) programmable. Please remember that regardless of the value programmed for this time interval (5, 10, 15, or 20 ms), Mx4 will internally perform a high-order interpolation of the points breaking them down to 200  $\mu$ s.

#### ***Command Sequence Example***

See EN\_BUFBRK

### **EXAMPLE**

Set a contouring interpolation interval of 10 ms.

```
BTRATE 1
```

## CAM

---

**FUNCTION** Engage Electronic Cam

**SYNTAX** CAM n, m, tablestart, tablesize

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n	long value specifying the master axis
m	long value specifying the slave axis
tablestart	long value specifies cam table start index
	0 <= tablestart <= 1600
tablesiz	long value specifies cam table size
	3 <= tablesiz <= 1600

### DESCRIPTION

The commands which make up the electronic cam feature are CAM, CAM\_OFF, CAM\_OFF\_ACC, CAM\_POINT, CAM\_POS, and CAM\_PROBE. DSPL keywords [CAMCOUNT1-8, Mx4 Octavia] [CAMCOUNT1-4, Mx4].

The Mx4 controller is capable of storing up to 1600 cam points. Each cam point consists of a master relative position, and an associated slave relative position. A cam table can be between 3 and 1600 cam points long, and the user may define any number of cam tables in the 1600-point cam table capacity. Cam commands utilize tablestart and tablesiz arguments to specify which 'portion' of the 1600-point cam table region to 'run' on.

Cam table points may be downloaded in file format from within Mx4pro or built from within DSPL using the CAM\_POINT command. The CAM\_POINT command may also be used to modify cam points 'on the fly.' The

## CAM cont.

---

DSPL identifiers `CAMCOUNT1,2,3,etc.` indicate at which cam table indices the slave axes(es) are ‘at’ (`CAMCOUNT1` is for axis 1, etc.).

The cam points consist of relative position values for master and slave. The first cam point in a table must be 0, 0. The last point in a cam table is the cycle length for master and slave. For example, if the full cam cycle for a master axis is 5000 counts and the slave would travel -1024 counts in that cycle, the last cam point in that cam table would be 5000, -1024. Note that the master/slave position ratios can not exceed the range [-256 to 255,999]. Also, the minimum ratio is +/- 1/128. For example, for 1000 counts of the master axis, the slave axis(es) can not have more than -256000 counts in the negative direction or 255999 counts in the positive direction.

The slave axes utilize the `MAXACC` acceleration value as the maximum acceleration the slave axis can reach while following the electronic cam trajectory, and therefore must be programmed before cam operation. This command turns on the mechanical cam function for the selected master and slave(s). The slave(s) follow the master according to the master/slave position pairs stored in the cam table. The slave axes(es) utilize `MAXACC` as the maximum acceleration they can achieve in following the master trajectory.



**Note:** Activation of \*ESTOP during cam operation will halt the master axis, and subsequently the slave axis(es). Slave(s) remain “engaged” in cam mode after the input-triggered halt.

**SEE ALSO** `CAM_OFF, CAM_OFF_ACC, CAM_POINT, CAM_POS, CAM_PROBE, MAXACC, SYNC`

### APPLICATION

General master/slaving, in particular packaging, synchronous cutting, flying shear, and mark registration, require the coordination of several axes in cam fashion. For these applications, the user is required to load the cam function along with the position spacing that defines the distance between the adjacent gear ratios stored in the cam table.

## **CAM cont.**

---

### **EXAMPLE**

Set axis 1 as the master axis, axes 2 and 3 as slaves. The axis 2 slave will use the 10-point cam table beginning at index 0, while the axis 3 slave will use the 25 point cam table beginning at index 100.

```
BEGIN_RTC  
    CAM 1, 2, 0, 10  
    CAM 1, 3, 100, 25  
END_RTC
```

## **CAM\_OFF**

---

**FUNCTION** Turns Off, Disengages Cam Slave Axis(es)

**SYNTAX** CAM\_OFF n

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

n long value specifying the slave axis to be disengaged

### **DESCRIPTION**

This command disengages the system that was under master slave control.

**SEE ALSO** CAM, CAM\_OFF\_ACC, CAM\_POINT, CAM\_POS, CAM\_PROBE, SYNC

### **APPLICATION**

General master/slaving, in particular packaging, synchronous cutting, flying shear, and mark registration, require the coordination of several axes in cam fashion. For these applications, the user is required to load the cam function along with the position spacing that defines the distance between the adjacent gear ratios stored in the cam table.

### **EXAMPLE**

Immediately disengage slave axes 3 and 4 from the master axis.

```
BEGIN_RTC
    CAM_OFF 3
    CAM_OFF 4
END_RTC
```

## **CAM\_OFF\_ACC**

---

**FUNCTION** Turns Off, Disengages Cam Slave Axis(es) With Acceleration

**SYNTAX** CAM\_OFF\_ACC n

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

n long value specifying the slave axis to be disengaged

### **DESCRIPTION**

This command disengages the system that was under master/slave control. The slave axis(es) will come to a stop at the maximum acceleration rate programmed by MAXACC.

**SEE ALSO** CAM, CAM\_OFF, CAM\_POINT, CAM\_POS, CAM\_PROBE, SYNC

### **APPLICATION**

General master/slaving, in particular packaging, synchronous cutting, flying shear, and mark registration, require the coordination of several axes in cam fashion. For these applications, the user is required to load the cam function along with the position spacing that defines the distance between the adjacent gear ratios stored in the cam table.

### **EXAMPLE**

Disengage with acceleration profile slave axes 3 and 4 from the master axis.

```
BEGIN_RTC
    CAM_OFF_ACC 3
    CAM_OFF_ACC 4
END_RTC
```

## CAM\_POINT

---

**FUNCTION** Place Cam Point Into Cam Table

**SYNTAX** CAM\_POINT tablestart, tablesize, index, masterpos,  
slavepos

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

tablestart long value specifies cam table start index  
 $0 \leq \text{tablestart} \leq 1600$

tablesize long value specifies cam table size  
 $3 \leq \text{tablesize} \leq 1600$

index long value specifies index at which to place the cam point  
 $0 \leq \text{index} \leq (\text{tablesize}-1)$

masterpos long value cam point master axis relative position  
slavepos long value cam point slave axis relative position

### DESCRIPTION

The CAM\_POINT allows the user to either build entire cam tables from within the DSPL environment or alternatively, edit cam table points (i.e.: change cam points 'on the fly'). Cam table points consist of master, slave position pairs, and cam tables can be anywhere from 3 to 1600 cam points long. The first point of a cam table (index = 0) must be 0,0. The last point of a cam table (index = tablesize-1) is mastercyclelength, slavecyclelength; where the cycle lengths for the master and slave are the relative cam cycle lengths (i.e.: master cycle length is 4096 counts, the slave cycle length is 1024 counts, for a full cycle ratio of 4:1). Cam master/slave position ratios can not exceed the range [-256 to 255,999]. Also, the minimum ratio is +/- 1/128.

## **CAM\_POINT cont.**

---

**SEE ALSO**    `CAM`, `CAM_OFF`, `CAM_OFF_ACC`, `CAM_POS`, `CAM_PROBE`, `SYNC`

### **APPLICATION**

See Application Notes.

### **EXAMPLE**

A 10-point cam table exists at table start index 500. Replace the 3rd point of the table with the master, slave point 1000, 3000.

```
CAM_POINT 500, 10, 2, 1000, 3000
```

## CAM\_POS

---

**FUNCTION** Turns Electronic Cam On at a Specified Position

**SYNTAX** CAM\_POS n, m, masterpos, tablestart, tablesize

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n	long value specifying the master axis
m	long value specifying the slave axis
masterpos	double precision value specifying the master position value for slave axis x that the electronics cam engages
tablestart	long value specifies cam table start index for slave
	0 <= tablestart <= 1600
tablesiz	long value specifies cam table size for slave
	3 <= tablesiz <= 1600

### DESCRIPTION

This command engages at the specified master position the mechanical cam function for the selected master and slave(s). The slave(s) follows the master according to the master/slave position pairs stored in the cam table. The slave axis(es) utilizes MAXACC as the maximum acceleration it can achieve in following the master trajectory.



**Note:** Activation of \*ESTOP during cam operation will halt the master axis, and subsequently the slave axis(es). Slave(s) remain “engaged” in cam mode after the input-triggered halt.

## **CAM\_POS cont.**

---

**SEE ALSO** CAM, CAM\_OFF, CAM\_OFF\_ACC, CAM\_POINT, CAM\_PROBE, SYNC

### **APPLICATION**

General master/slaving, in particular packaging, synchronous cutting, flying shear, and mark registration, require the coordination of several axes in cam fashion. For these applications, the user is required to load the cam function along with the position spacing that defines the distance between the adjacent gear ratios stored in the cam table.

### **EXAMPLE**

Set axis 4 as the master axis, axes 2 and 3 as slaves. The axis 2 slave will use the 10-point cam table beginning at index 0, while the axis 3 slave will use the 25-point cam table beginning at index 100. Axis 2 slave should engage when the master axis is at position 1000, and axis 3 slave should engage when the master axis is at position 4096.

```
BEGIN_DSPL
  CAM_POS 8, 2, 1000, 0, 10
  CAM_POS 8, 3, 4096, 100, 25
END_RTC
```

## CAM\_PROBE

---

**FUNCTION** Turns Electronic Cam On After Probe Input

**SYNTAX** CAM\_PROBE n, m, q, tablestart, tablesize

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the master axis  
m long value specifying the slave axis  
q long value specifies the \*EXTx probe interrupt to be used

[Mx4]

q=1 : \*EXT1  
q=2 : \*EXT2

[Mx4 Octavia]

q=1 : \*EXT1  
q=2 : \*EXT2  
q=4 : \*EXT3  
q=8 : \*EXT4

tablestart long value specifies cam table start index for slave

0 <= tablestart <= 1600

tablesizel long value specifies cam table size for slave

3 <= tablesizel <= 1600

## CAM\_PROBE cont.

---

### DESCRIPTION

This command engages at the occurrence of the specified external interrupt (\*EXT1, 2, 3, 4) the mechanical cam function for the selected master and slave(s). The slave(s) follow the master according to the master/slave position pairs stored in the cam table. The slave axis(es) utilizes MAXACC as the maximum acceleration they can achieve in following the master trajectory.



**Note:** Execution of the CAM\_PROBE command will disable any previously enabled EN\_PROBE interrupt. Probe input (\*EXT1, \*EXT2, \*EXT3, or \*EXT4) activation does not generate an interrupt with the CAM\_PROBE command.



**Note:** Activation of \*ESTOP during cam operation will halt the master axis, and subsequently the slave axis(es). Slave(s) remain “engaged” in cam mode after the input-triggered halt.

**SEE ALSO** CAM, CAM\_OFF, CAM\_OFF\_ACC, CAM\_POINT, CAM\_POS, SYNC

### APPLICATION

General master/slaving, in particular packaging, synchronous cutting, flying shear, and mark registration, require the coordination of several axes in cam fashion. For these applications, the user is required to load the cam function along with the position spacing that defines the distance between the adjacent gear ratios stored in the cam table.

## **CAM\_PROBE cont.**

---

### **EXAMPLE**

Set axis 2 as the master axis, axes 1 and 3 as slaves. The axis 1 slave will use the 100-point cam table beginning at index 0, while the axis 3 slave will use the 250-point cam table beginning at index 220. Engage slave axes in cam at occurrence of \*EXT2 interrupt.

```
BEGIN_RTC
  CAM_PROBE 2, 1, 2, 0, 100
  CAM_PROBE 2, 3, 2, 220, 250
END_RTC
```

## **CHANGECARDADDRESS**

---

**FUNCTION**      Change Mx4 Bus Address

**SYNTAX**        `CHANGECARDADDRESS address`

If used as a function, the function will return (long) the previous address if successful, zero if error.

### **ARGUMENTS**

address          long value specifying new card address

### **DESCRIPTION**

This function is used to change the pointer to the Mx4 card residing on the ISA bus.

**SEE ALSO**      `CURRENTCARDADDRESS`

### **EXAMPLE**

The Mx4 card has jumper settings placing it at address 0xd0000 on the ISA bus. Set the Visual Basic programming pointer to this address.

```
CHANGECARDADDRESS &HD0000
```

## **CHANGECOMMPORTSETTING**

---

**FUNCTION**      Change Serial Communication Comm Port Setting

**SYNTAX**          CHANGECOMMPORTSETTING port

If used as a function, the function will return (byte) the previous comm port setting.

### **ARGUMENTS**

port              byte value specifying new comm port

### **DESCRIPTION**

This function is used to change the comm port which is used to communicate serially to the Mx4 card.

**SEE ALSO**        CHANGESLAVENODEADDRESS,                      COMMUNICATIONSLOST,  
                      GETCOMMINSTCOUNT,                                              GETCOMMTYPE,  
                      GETCURRENTNODEADDRESS, RESETCOMMUNICATIONS

### **EXAMPLE**

Set the comm port to communicate to Mx4 through the comm2 port.

```
CHANGECOMMPORTSETTING 2
```

## **CHANGESLAVENODEADDRESS**

---

**FUNCTION**      Change Serial Communication Node Address

**SYNTAX**          CHANGESLAVENODEADDRESS    node

If used as a function, the function will return (byte) the previous slave node address.

### **ARGUMENTS**

node              byte value specifying new node address

### **DESCRIPTION**

This function is used to change the serial communication slave node address of the Mx4 card desired.

**SEE ALSO**          CHANGECOMMPORTSETTING,                      COMMUNICATIONSLOST,  
GETCOMMINSTCOUNT,                                              GETCOMMTYPE,  
GETCURRENTNODEADDRESS,    RESETCOMMUNICATIONS

### **EXAMPLE**

Set the serial communication slave node address to 4.

```
CHANGESLAVENODEADDRESS 4
```

## **CHANGE\_VAR**

---

**FUNCTION**     Change DSPL variable Value

**SYNTAX**        CHANGE\_VAR var, value

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

Var	long value specifying DSPL variable (1-128) to modify
Value	double precision value for specified variable

### **DESCRIPTION**

DSPL variable values may be changed in real time via the CHANGE\_VAR function.

**SEE ALSO**     MONITOR\_VAR, VAR

### **EXAMPLE**

Set DSPL VAR67 equal to 1000.123.

```
CHANGE_VAR 67, 1000.123
```

## **CLEAR\_CUBIC**

---

**FUNCTION** Clear Internal Cubic Spline Data Table

**SYNTAX** CLEAR\_CUBIC

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

None

### **DESCRIPTION**

This function issues a CLEAR\_CUBIC RTC. This command clears the Mx4 internal cubic spline data storage area.

**SEE ALSO** CUBIC\_INT, DOWN\_CUBIC

### **EXAMPLE**

Clear the Mx4 cubic spline data table storage area.

```
CLEAR_CUBIC
```

## **CLEAR\_DSPL**

---

**FUNCTION** Clear DSPL Program

**SYNTAX** CLEAR\_DSPL

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

None

### **DESCRIPTION**

This function clears the Mx4 DSPL program storage area.

**SEE ALSO** DOWNLOAD\_DSPL

### **EXAMPLE**

Clear the Mx4 DSPL program storage area.

```
CLEAR_DSPL
```

## **CLEAR\_POINTS**

---

**FUNCTION** Clear DSPL Table\_p / Table\_v data storage area

**SYNTAX** CLEAR\_POINTS

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

None

### **DESCRIPTION**

This function clears the Mx4 DSPL table\_p / table\_v data storage area.

**SEE ALSO** DOWN\_POINTS

### **EXAMPLE**

Clear the Mx4 DSPL table\_p / table\_v storage area.

CLEAR\_POINTS

## **CLEAR\_POS\_TABLE**

---

**FUNCTION** Clear Specified Position Compensation Table

**SYNTAX** CLEAR\_POS\_TABLE table

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

Table long value specifying position compensation table to clear (1-4, Mx4; 1-8, Mx4 Octavia)

### **DESCRIPTION**

This function clears the position compensation table for the specified axis.

**SEE ALSO** DOWN\_POS

### **EXAMPLE**

Clear the axis 3 position compensation table.

```
CLEAR_POS 3
```

## **CLEAR\_VEL\_TABLE**

---

**FUNCTION** Clear Specified Velocity Compensation Table

**SYNTAX** CLEAR\_VEL\_TABLE table

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

Table long value specifying velocity compensation table to clear (1-4, Mx4; 1-8, Mx4 Octavia)

### **DESCRIPTION**

This function clears the velocity compensation table for the specified axis.

**SEE ALSO** DOWN\_VEL

### **EXAMPLE**

Clear the axis 4 velocity compensation table.

```
CLEAR_VEL 4
```

## COMMUNICATIONSLOST

---

**FUNCTION** Check For Lost Serial Communication

**SYNTAX** COMMUNICATIONSLOST ()

**ARGUMENTS**

none

**DESCRIPTION**

This function checks if the serial communication between host and Mx4 has been lost. The function returns long value 1 if communication is lost, 0 otherwise.

**SEE ALSO** CHANGECOMMPORTSETTING, CHANGESLAVENODEADDRESS,  
GETCOMMINSTCOUNT, GETCOMMTYPE,  
GETCURRENTNODEADDRESS, RESETCOMMUNICATIONS

**EXAMPLE**

Check for lost serial communication.

```
IF (COMMUNICATIONSLOST () = 1) THEN
```

## CTRL

---

**FUNCTION** Control Law Parameters

**SYNTAX** CTRL n, par<sub>1</sub>, par<sub>2</sub>, par<sub>3</sub>, par<sub>4</sub>

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n	long value specifying the axis
par <sub>1</sub>	long unsigned value for Ki gain
par <sub>2</sub>	long unsigned value for Kp gain
par <sub>3</sub>	long unsigned value for Kf gain
par <sub>4</sub>	long unsigned value for Kd gain

$$0 \leq \text{par}_x \leq 32767$$

### DESCRIPTION

This command performs a state feedback control algorithm combined with a modified PID. The state feedback control algorithm includes an observer which estimates the instantaneous values for speed and acceleration. The feedback loops are then individually commanded to provide a robust control, which is smooth and stable over a wide range of servo operation. In addition, this algorithm performs a modified PID with the saturation threshold set for integral action. A common PID includes two zeros and one pole, which may not be suitable for systems with noisy feedback. Also, the integral part of a common PID algorithm may saturate the registers creating overshoots or other forms of instability. A modified PID includes a second pole to solve the latter problem and a programmable integral limit to solve the former one.

In the modified PID algorithm; par<sub>1</sub>, par<sub>2</sub>, par<sub>3</sub>, and par<sub>4</sub> are values representing the integral, proportional, velocity state feed forward, and differential gains, respectively.

## CTRL cont.

### Scaling Factors

The DSP uses an internal scaling factor for each gain. These factors have been optimally selected for worst case numerical conditions. These factors are:

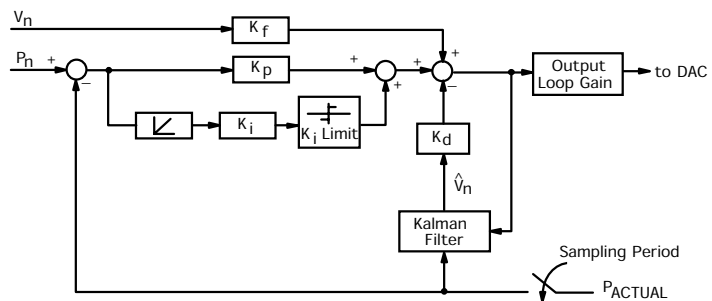
GAIN	SCALING FACTOR	VALUE
$K_f$	1.525E-08	$v/(c/s)$
$K_p$	0.595E-06	$v/c$
$K_i$	3.308E-05	$(v/s)/c$
$K_d$	1.9875E-08	$v/(c/s)$
Output Loop Gain	integer	NA

$v = \text{volts}, c = \text{encoder edge counts}, s = \text{seconds}$

For example,

100 counts of position error and  $K_p$  of 1000 (other gains are zero) will result in an output voltage of 59.5 millivolts.

$$\text{i.e. } 100 \times 1000 \times 0.595\text{E-}06 = 59.5$$



Block Diagram of Control Law

**SEE ALSO** CTRL\_KA, KILIMIT, OFFSET, OUTGAIN

## CTRL cont.

---

### APPLICATION

This command is used in all position/velocity control tuning applications. For more information on the effectiveness of each gain on system dynamic response, please refer to the *Mx4Pro: Tuning Expert* manual. This manual will help you understand the significance of gains in tuning. Please read this even if you cannot run *Mx4Pro* on your machine because it lacks the DOS operating system.

#### **Command Sequence Example**

See AXMOVE and VELMODE

### EXAMPLE

Set the following modified PID gain values for axes 2 and 4:

$$\begin{aligned} K_i &= 100 \\ K_p &= 4000 \\ K_f &= 3000 \\ K_d &= 2500 \end{aligned}$$

$$\begin{aligned} K_i &= 20 \\ K_p &= 8000 \\ K_f &= 5500 \\ K_d &= 7000 \end{aligned}$$

```
BEGIN_RTC
  CTRL 2, 100, 4000, 3000, 2500
  CTRL 4, 20, 8000, 5500, 7000
END_RTC
```

## CTRL\_KA

---

**FUNCTION** Acceleration Feedforward Control Law Parameter

**SYNTAX** CTRL\_KA n, ka

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis  
ka long unsigned value for Ka gain

$0 \leq ka \leq 32767$

### DESCRIPTION

The CTRL\_KA command allows the user to program an acceleration feedforward gain for the specified axis.

**SEE ALSO** CTRL, KILIMIT, OFFSET, OUTGAIN

### EXAMPLE

Program a Ka of 5000 for both axes 1 and 3.

```
BEGIN_RTC
    CTRL_KA 1, 5000
    CTRL_KA 3, 5000
END_RTC
```

## CUBIC\_INT

---

**FUNCTION** Start the Internal Cubic Spline Contouring Execution

**SYNTAX** CUBIC\_INT m, si, n, ax

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

m long value specifies the number of points in the cubic spline table to run. Each point is characterized by the position for only one motor. The maximum number of points is 4,096.

si long value specifies the starting index in the table

n long value specifies the number of times m points of a spline table will be looped over

$n \leq 32767$

ax long value bit codes the axes involved



**Note:** n = 0 means run the specified number of points infinite number of times.

## **CUBIC\_INT cont.**

---

### **DESCRIPTION**

This command starts execution of the points stored in the cubic spline table immediately. The command sequence for this instruction is as follows:

- 1) CUBIC\_RATE
- 2) CUBIC\_SCALE ;if necessary
- 3) CUBIC\_INT

We assume that user has already downloaded the table points to the cubic spline table location.

Upon execution of a CUBIC\_INT command, the DSPL program flow will not proceed to a following CUBIC\_INT, CUBIC\_RATE, or CUBIC\_SCALE command until the current CUBIC\_INT motion is completed. If the command following the CUBIC\_INT command is not a CUBIC\_INT, CUBIC\_RATE, or CUBIC\_SCALE command, the DSPL program flow will proceed to that command immediately after the CUBIC\_INT command execution.

**SEE ALSO** CLEAR\_CUBIC, CUBIC\_RATE, CUBIC\_SCALE, DOWN\_CUBIC

### **APPLICATION**

Refer to Cubic Spline

### **EXAMPLE**

Execute internal cubic spline contouring starting at index 100, 50 points, axes 2 and 3, repeating 5 times.

```
CUBIC_INT 50, 100, 5, &H6
```

## CUBIC\_RATE

---

**FUNCTION** Set Cubic Spline Point Transfer Rate

**SYNTAX** CUBIC\_RATE m

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

m long value parameter coding the value for cubic spline transfer rate. "m" codes the time interval between the adjacent position points. Its value ranges between 5 and 511 and when divided by 5, it represents the interval in ms. For example, m=5 represents the time interval of 1 ms and m=25 is a 5 ms interval.

### DESCRIPTION

This command sets the point transfer rate for the cubic spline. The "transfer rate" sets the interval between two adjacent position points in the cubic spline table. The two adjacent points can be spaced anywhere between 1.0 to 102.4 ms. Mx4's cubic spline interpolates between the two adjacent points at 200  $\mu$ s increments. This means for example, Mx4 inter-polates 500 points between two adjacent points 100 ms apart.

Upon execution of a CUBIC\_INT command, the DSPL program flow will not proceed to a following CUBIC\_INT, CUBIC\_RATE, or CUBIC\_SCALE command until the current CUBIC\_INT motion is completed. If the command following the CUBIC\_INT command is not a CUBIC\_INT, CUBIC\_RATE, or CUBIC\_SCALE command, the DSPL program flow will proceed to that command immediately after the CUBIC\_INT command execution.

**SEE ALSO** CLEAR\_CUBIC, CUBIC\_INT, CUBIC\_SCALE, DOWN\_CUBIC

## CUBIC\_RATE cont.

---

### APPLICATION

Refer to *Cubic Spline Application Notes*.

### EXAMPLE

Using cubic spline interpolation creates 16, 32, 64, and 128-point circles.

The following shows the position values for 16 uniformly spaced points on a circle.

#### 16-point Circle

Point	pos_x
x1	2500
x2	2310
:	:
x16	2310

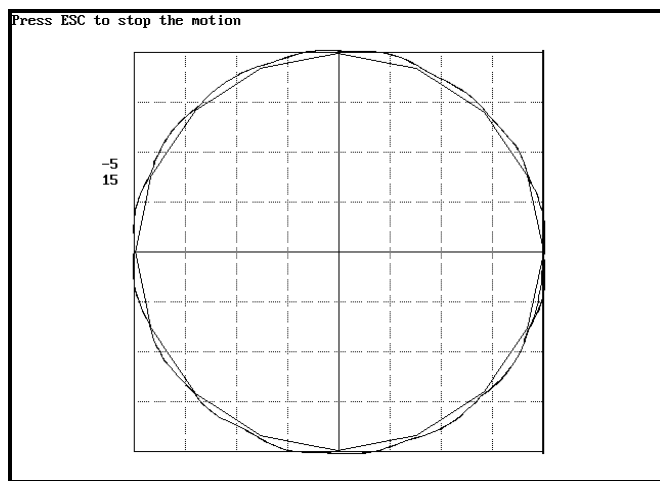
  

Point	pos_y
x1	0
x2	957
:	:
x16	-957

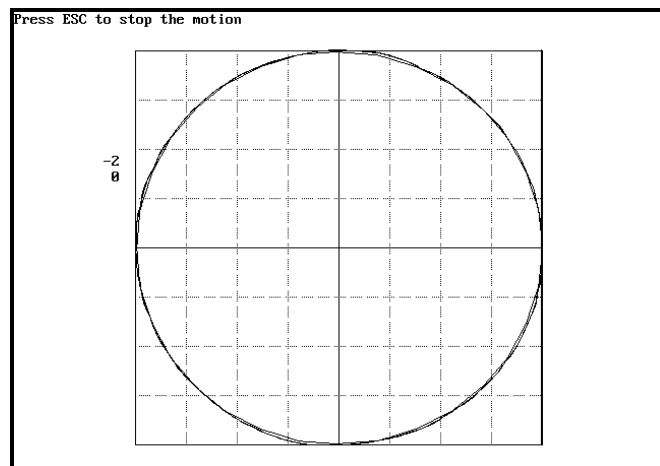
To generate a circle, these points must be written to the Mx4's memory and `CUBIC_RATE` must be executed. The `CUBIC_RATE` argument determines the interval between two points in the memory. For comparison, the following figures illustrate the circles created by 16, 32, 64, and 128 points in a cubic spline interpolation. It takes 1.28 seconds to complete these circles.

## CUBIC\_RATE cont.

---



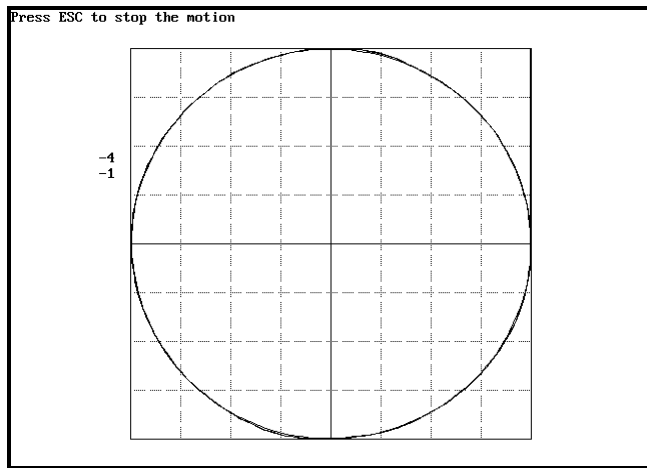
16 points; CUBIC\_RATE 400; 80ms time space between adjustment points



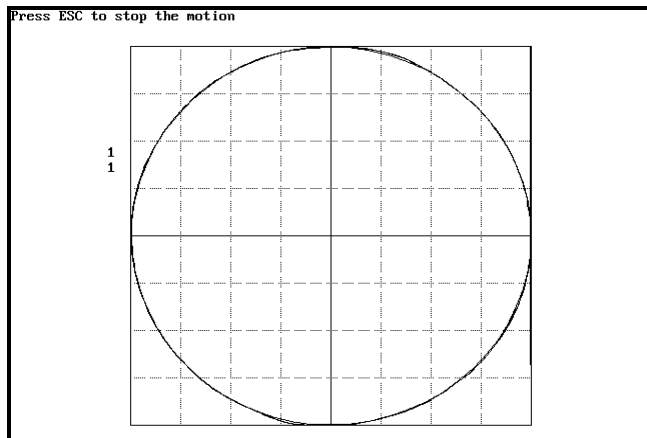
32 points; CUBIC\_RATE 200; 40ms time space between adjustment points

## CUBIC\_RATE cont.

---



64 points; CUBIC\_RATE 100; 20ms time space between adjustment points



128 points; CUBIC\_RATE 50; 10ms time space between adjustment points

## CUBIC\_SCALE

---

**FUNCTION** Scales/Shift Position Points

**SYNTAX** CUBIC\_SCALE n, pos\_mult, pos\_shift

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis

pos\_mult single precision value position scaling multiplier

$-2 \leq \text{pos\_mult} < 2$

pos\_shift double precision value position shift. This value transfers the position to a new origin.

### DESCRIPTION

This command scales those table points involved in a cubic spline operation. This command also shifts the positions involved by a user-defined position shift value.

Upon execution of a CUBIC\_INT command, the DSPL program flow will not proceed to a following CUBIC\_INT, CUBIC\_RATE, or CUBIC\_SCALE command until the current CUBIC\_INT motion is completed. If the command following the CUBIC\_INT command is not a CUBIC\_INT, CUBIC\_RATE, or CUBIC\_SCALE command, the DSPL program flow will proceed to that command immediately after the CUBIC\_INT command execution.

**SEE ALSO** CLEAR\_CUBIC, CUBIC\_INT, CUBIC\_RATE, DOWN\_CUBIC

### APPLICATION

See *Cubic Spline Application Notes*

## **CUBIC\_SCALE cont.**

---

### **EXAMPLE**

Scale the cubic spline data for axis 5 by a factor of x0.5.

```
CUBIC_SCALE 5, 0.5, 0
```

## **CURRENTCARDADDRESS**

---

**FUNCTION**     Get Current Mx4 Bus Address

**SYNTAX**       CURRENTCARDADDRESS ()

**ARGUMENTS**

                  none

**DESCRIPTION**

                  This function returns the long value of the pointer to the Mx4 card residing on the ISA bus.

**SEE ALSO**       CHANGECARDADDRESS

**EXAMPLE**

                  Read the current Mx4 card address into the Visual Basic variable "ADDR".

```
                  ADDR = CURRENTCARDADDRESS ()
```

## CURR\_LIMIT

Vx4++ option command

**FUNCTION** Set Output Drive Current Limit

**SYNTAX** CURR\_LIMIT n, clmt

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis  
clmt single precision value specifying the current limit percentage

$$0 \leq \text{clmt} \leq 100(\%)$$

### DESCRIPTION

This command sets the current limit for the axes specified. The current limit is defined as a percentage of the maximum desired current (which in turn is defined by the current feedback mechanism). In the case that the current in any phase of a specified axis exceeds the set value, the PWM signals for that axis will turn off for at least one full period and turn on only if the sensed current is reduced below the current limit.



**Note:** Mx4 with Vx4++ will not execute the CURR\_LIMIT command if the VX4\_BLOCK command is active for the axes in question.

**SEE ALSO** Vx4\_BLOCK

### APPLICATION

See *Vx4++ User's Guide*

## **CURR\_LIMIT cont.**

**Vx4++ option command**

---

### **EXAMPLE**

For current feedback designed for full scale at 10 amps, set current limits of 3 and 4 amps for axes one and two, respectively.

$$(3/10) * 100\% = 30\% \quad (4/10) * 100\% = 40\%$$

```
BEGIN_RTC
  CURR_LIMIT 1, 30.0
  CURR_LIMIT 2, 40.0
END_RTC
```

## CURR\_OFFSET

Vx4++ option command

**FUNCTION**     Compensate Current Feedback Offset

**SYNTAX**        `CURR_OFFSET n, val`

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n                long value specifying the axis  
val              long value, offset

$-32768 \leq \text{val} \leq 32767$

### DESCRIPTION

The `CURR_OFFSET` command allows the user to compensate for any offset generated by the current feedback path.



**Note:** Mx4 with Vx4++ will not execute the `CURR_OFFSET` command if the `VX4_BLOCK` command is active for the axes in question.

**SEE ALSO**        `Vx4_BLOCK`

### APPLICATION

See *Vx4++ User's Guide*

### EXAMPLE

Program an offset compensation value of 2500 for axis one

```
CURR_OFFSET 1, 2500
```

**CURR\_PID**

Vx4++ option command

**FUNCTION** Current Loop Control Law Parameters**SYNTAX** CURR\_PID n, par<sub>1</sub>, par<sub>2</sub>, par<sub>3</sub>

If used as a function, the function will return (long) zero if successful, nonzero if error.

**ARGUMENTS**

n	long value specifying the axis
par <sub>1</sub>	long unsigned value for K <sub>p</sub> gain
par <sub>2</sub>	long unsigned value for K <sub>i</sub> gain
par <sub>3</sub>	long unsigned value for K <sub>d</sub> gain

$$0 \leq \text{par}_{1,2,3} \leq 32767$$
**DESCRIPTION**

This command performs a vector control algorithm combined with a modified PID.

**SEE ALSO** CTRL**APPLICATION**

See *Vx4++ User's Guide*

**EXAMPLE**

Set the following modified current loop PID gain values for axis three.

K <sub>p</sub>	=	10000
K <sub>i</sub>	=	20
K <sub>d</sub>	=	9500

```
CURR_PID 3, 10000, 20, 9500
```

## DDAC

---

**FUNCTION** Direct DAC Output

**SYNTAX** DDAC n, val

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis  
val single precision DAC output voltage

$-10.0 \leq \text{val} \leq 9.9997$  volts

### DESCRIPTION

The DDAC command places the axis(es) in open loop, with DAC(x) output voltage determined by the val<sub>x</sub> command argument. DDAC specifies a bipolar analog signal ranging from -10 to +10 volts with a resolution of approximately 0.3 millivolts.

After execution of a DDAC command, in order to return the axis(es) to closed loop operation, a closed-loop command such as AXMOVE or VELMODE must be executed. The following procedure serves as an example:

1. slow or halt the axis(es) motion:  
-execute DDAC with 0v specified
2. minimize built-up following error:  
-execute POS\_PRESET command
3. return axis(es) to closed loop:  
-execute AXMOVE command with target position specified as that used in the preceding POS\_PRESET command.

**SEE ALSO** none

## **DDAC cont.**

---

### **APPLICATION**

This command can be used in applications where the voltage command provides adequate control. Voltage commands can be applied to a torque loop (for torque control applications in robotics) or a velocity loop (to a spindle axis in machine tool applications).

#### ***Command Sequence Example***

No preparation is required before running this instruction.

### **EXAMPLE**

Output +3.75 volts to the axis 4 and axis 5 DACs.

```
BEGIN_RTC
  DDAC  4, 3.75
  DDAC  5, 3.75
END_RTC
```

## DISABL\_INT

---

**FUNCTION**    Disable Interrupts

**SYNTAX**        DISABL\_INT    n, mask

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n                long value specifying the axis  
mask             long value, interrupt disable mask

The mask is created by 'OR'ing together any of the desired interrupt's defined constants:

IC_MOTION_COMPLETE	(EN_MOTCP)
IC_INDEX_PULSE	(EN_INDEX)
IC_PROBE_SIGNAL	(EN_PROBE)
IC_POSITION_BREAKPOINT	(EN_POSBRK)
IC_FOLLOWING_ERROR	(EN_ERR)
IC_FOLLOWING_ERROR_AND_HALT	(EN_ERRHLT)
IC_BUFFER_BREAKPOINT	(EN_BUFBRK)

### DESCRIPTION

This command disables some or all of the servo control card interrupts.

**SEE ALSO**        DISABL2\_INT, EN\_BUFBRK, EN\_PROBE, EN\_ERR, EN\_ERRHLT,  
EN\_INDEX, EN\_MOTCP, EN\_POSBRK

### APPLICATION

This command may be used in conjunction with all applications in which only a few interrupts are needed to be enabled. Also a few enabled interrupts may have to be disabled based on external events.

#### **Command Sequence Example**

No preparation is required before running this instruction.

## **DISABL\_INT cont.**

---

### **EXAMPLE**

Disable the previously enabled axis 1 following error and index pulse interrupts.

```
TEMP = IC_FOLLOWING_ERROR | IC_INDEX_PULSE  
DISABL_INT 1, TEMP
```



## **DISABL2\_INT cont.**

---

### **EXAMPLE**

Disable the previously enabled axis 1 and axis 7 encoder fault interrupts.

```
TEMP = IC_ENCODER_FAULT
BEGIN_RTC
    DISABL2_INT 1, TEMP
    DISABL2_INT 7, TEMP
END_RTC
```

## **DOWNLOAD\_DSPL**

---

**FUNCTION**     Download Compiled DSPL Program To Mx4

**SYNTAX**        `DOWNLOAD_DSPL filename`

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

filename     string, name of file to download

### **DESCRIPTION**

This function performs the task of downloading a compiled DSPL program to the Mx4 controller. The DSPL file must have been previously compiled (DSPLCxxx.EXE) so that a .LOD file extension file exists.

**SEE ALSO**        `CLEAR_DSPL, START_DSPL, STOP_DSPL`

### **EXAMPLE**

Download the compiled DSPL file MYTEST.LOD to Mx4.

```
Dim sFileName As String
SFileName = "c:\work\mytest.lod"
DOWNLOAD_DSPL sFileName
```

## DOWN\_CAM

---

**FUNCTION** Download Cam Data Points To Mx4 Cam Table

**SYNTAX** DOWN\_CAM madata, sldata, npts, index

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

madata	master position array, single precision
sldata	slave position array, single precision
npts	long value, number of cam points to download
index	long value, starting cam table index to download points

### DESCRIPTION

This function performs the task of downloading cam table points to the Mx4 cam table storage area, beginning at the specified index.

**SEE ALSO** CAM, CAM\_OFF, CAM\_OFF\_ACC, CAM\_POINT, CAM\_POS, CAM\_PROBE

### EXAMPLE

Assume that the master / slave position points are stored in the file CAM\_TUT5.DAT. Download this file to the Mx4 cam table beginning at cam index 100. The file consists of 10 master positions, 10 slave positions.

```
Dim master(10) As Single
Dim slave(10) As Single
Dim I As Integer
Open "c:\work\cam_tut5.dat" For Input As #1
For I = 0 To 9
    Input #1, master(I)
    Input #1, slave(I)
Next I
DOWN_CAM master(0), slave(0), 10, 100
```

## **DOWN\_CUBIC**

---

**FUNCTION**     Download Cubic Spline Data Points To Mx4

**SYNTAX**        DOWN\_CUBIC npts, cudata, index

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

npts	long value specifying number of points to be downloaded
cudata	position array, single precision
index	long value, starting cubic spline table index to download points

### **DESCRIPTION**

This function performs the task of downloading cubic spline table points to the Mx4 internal cubic spline table storage area, beginning at the specified index.

**SEE ALSO**        CLEAR\_CUBIC, CUBIC\_INT, CUBIC\_RATE, CUBIC\_SCALE

### **EXAMPLE**

Assume that the cubic spline position points are stored in the file CUB\_TUT8.DAT. Download this file to the Mx4 cubic spline table beginning at cubic index 0. The file consists of 350 position points.

```
Dim cubdata(350) As Single
Dim I As Integer
Open "c:\work\cub_tut8.dat" For Input As #1
For I = 0 To 349
    Input #1, cubdata(I)
Next I
DOWN_CUBIC 350, cubdata(0), 0
```

## DOWN\_POINTS

---

**FUNCTION** Download DSPL Table\_p / Table\_v Data Points

**SYNTAX** DOWN\_POINTS ptdata, npts, index, posvel

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

ptdata	data points array, single precision
npts	long value specifying number of points to be downloaded
index	long value, starting table index to download points
posvel	long value specifying data sent in table_p position format (0) or table_v velocity format (1)

### DESCRIPTION

This function downloads data to the DSPL table\_p / table\_v storage areas. The posvel argument selects whether the data is sent in position format (table\_p) or velocity format (table\_v).

**SEE ALSO** none

### EXAMPLE

Assume that 200 velocity values are stored in the file TEST.DAT. Download this file to the Mx4 table\_v table beginning at index 50.

```
Dim ddata(200) As Single
Dim I As Integer
Open "c:\work\test.dat" For Input As #1
For I = 0 To 199
    Input #1, ddata(I)
Next I
DOWN_POINTS ddata(0), 200, 50, 1
```

## DOWN\_POS

---

**FUNCTION** Download Position Compensation Table

**SYNTAX** DOWN\_POS pdata, npts, table

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

pdata	data points array, single precision
npts	long value specifying number of points to be downloaded
table	long value specifying the table axis (1-8)

### DESCRIPTION

This function downloads position compensation tables to the Mx4 controller.

**SEE ALSO** DOWN\_VEL, TABLE\_SEL

### EXAMPLE

Download the 1024 point position compensation table POSCOMP.DAT to the axis 6 position compensation table.

```
Dim compdata(1024) As Single
Dim I As Integer
Open "c:\work\poscomp.dat" For Input As #1
For I = 0 To 1023
    Input #1, compdata(I)
Next I
DOWN_POS compdata(0), 1024, 6
```

## **DOWN\_VEL**

---

**FUNCTION**     Download Velocity Compensation Table

**SYNTAX**        DOWN\_VEL vdata, npts, table

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

vdata	data points array, single precision
npts	long value specifying number of points to be downloaded
table	long value specifying the table axis (1-8)

### **DESCRIPTION**

This function downloads velocity compensation tables to the Mx4 controller.

**SEE ALSO**        DOWN\_POS, TABLE\_SEL

### **EXAMPLE**

Download the 1024 point velocity compensation table VELCOMP.DAT to the axis 3 velocity compensation table.

```
Dim compdata(1024) As Single
Dim I As Integer
Open "c:\work\velcomp.dat" For Input As #1
For I = 0 To 1023
    Input #1, compdata(I)
Next I
DOWN_VEL compdata(0), 1024, 3
```

## ENCOD\_MAG

Vx4++ option command

**FUNCTION** Define Encoder Line Count, Motor Poles, Commut. Option

**SYNTAX** ENCOD\_MAG n, p1, p2, p3

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis  
p1 long value, number of encoder lines/rev

$0 \leq p1 \leq 65535$

p2 long value, number of motor poles

$0 \leq p2 \leq 256$

p3 long value, brushless DC commutation option

p3 = 0 : brushtype DC or AC induction motor tech

p3 = 0 : comm option 0

p3 = 1 : comm option 1

### DESCRIPTION

The Vx4++ option card interfaces to the motors with any number of magnetic poles and encoders with any number of encoder pulse numbers. An example of this is a brushless DC machine with eight poles, 1,000 line encoder and hall sensors mounted in a special configuration. This command allows the user to define the encoder, commutation, and motor pole parameters for the specified axis(es).



**Note:** Mx4 with Vx4++ will not execute the ENCOD\_MAG command if the VX4\_BLOCK command is active for the axes in question.

**SEE ALSO** VX4\_BLOCK

## **ENCOD\_MAG cont.**

---

### **APPLICATION**

*See Vx4++ User's Guide*

### **EXAMPLE**

Axis four is an AC induction motor with a 1024 line encoder and 4 motor poles.

```
ENCOD_MAG 4, 1024, 4, 0
```

## END\_RTC

---

**FUNCTION** End Multi-Axis Command

**SYNTAX** END\_RTC

### ARGUMENTS

None

### DESCRIPTION

A number of RTCs have a large and variable number of arguments. These are mostly motion control RTCs which permit the desired motion for several axes to be specified at once. All of the motion control functions in the DLL are single axis. The BEGIN\_RTC and END\_RTC functions permit a multi-axis RTC to be built-up from multiple calls to a single axis RTC function. The AXMOVE function illustrates this. A call to AXMOVE by itself will generate an AXMOVE RTC for the specified axis. To generate a two-axis AXMOVE RTC, two calls to AXMOVE would be bracketed between calls to BEGIN\_RTC and END\_RTC.

**SEE ALSO** BEGIN\_RTC

### APPLICATION

Multi-axis commands are needed when the trajectories of two or more axes must be synchronized.

### EXAMPLE

This example illustrates how BEGIN\_RTC and END\_RTC can be used to issue a two axis AXMOVE command to Mx4. Assuming current positions of zero for axes 1 and 2, we want to move axis 1 to the target position of 234567 and axis 2 to the target position of -3000 counts. Let's also assume that we want this move to be accomplished with the slew rate velocity of 4.0 counts/200 $\mu$ s for axis 1 and -3.50 counts/200 $\mu$ s for axis 2, and an acceleration of 0.005 counts/(200  $\mu$ s)<sup>2</sup> for both axes.

```
BEGIN_RTC
  AXMOVE 1, 0.005, 234567, 4.00
  AXMOVE 2, 0.005, -3000, -3.50
END_RTC
```

## ENDDLLCRITICALSECTION

---

**FUNCTION** End critical section

**SYNTAX** ENDDLLCRITICALSECTION

### ARGUMENTS

None

### DESCRIPTION

This command releases the mutex lock for exclusive access to the DLL. The command should be used when multiple threads are accessing the DLL, and it should be used after every function call to the DLL. The function BEGINDLLCRITICAL SECTION acquires the mutex lock.

**SEE ALSO** BEGINDLLCRITICALSECTION

### APPLICATION

The function is used in multithreaded applications to provide synchronization for DLL access and maintain a consistent state in the DLL.

### EXAMPLE

```
var1 = BEGINDLLCRITICALSECTION(100)
if var1 == ERR_OK then
BEGIN_RTC
    AXMOVE          1,    0.005,    234567,    4.00
    AXMOVE          2,    0.005,    -3000,    -3.50
END_RTC
ENDDLLCRITICALSECTION
Endif
```

## EN\_BUFBRK

---

**FUNCTION** Enable Buffer Breakpoint Interrupt

**SYNTAX** EN\_BUFBRK buffbrk

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

buffbrk positive long value which represents the delta position for the remaining number of bytes in the ring buffer. Since each contouring point requires 8 bytes, this number must be multiplied by 8 to indicate the real number of bytes left in the ring buffer.

$1 \leq \text{buffbrk} \leq 84$  contouring data points

### DESCRIPTION

This command will cause an interrupt when the number of contouring data points in the contouring ring buffer falls below a preset breakpoint. The buffer breakpoint interrupt status will appear in bit 0 of the DPR interrupt flag location [Mx4:7FEh] [Mx4 Octavia:1FFEh]. This bit gets set if a buffer breakpoint interrupt occurs.

**SEE ALSO** DISABL\_INT

### APPLICATION

This command must be used in both 2nd order and cubic spline contouring applications. To maintain continuity in a contouring application, Mx4 must be constantly updated by the host processor with a set of new (position/velocity) points on the contour. Since no application can afford to run out of points, the host must set the buffer breakpoint interrupt to a value such that running the remaining points (what is left in the ring buffer) will give the host enough time to update the buffer. For slower hosts, the argument for this command must be relatively larger.

## EN\_BUFBRK cont.

---

### Command Sequence Example

```

MAXACC ( )      ;set the maximum accel. so system can be stopped
CTRL ( )        ;set the gains
KILIMIT ( )
.
.               ;load the ring buffer with contouring points,
.               ;(position and speed)
BTRATE ( )      ;set the 2nd order contouring block transfer rate to 5,
                ;10, 15 or 20 ms
EN_BUFBRK ( )   ;set the breakpoint in buffer
.
.
START (n)       ;start contouring

```

### EXAMPLE

Enable a contouring ring buffer's breakpoint interrupts for the case that the number of segment move commands in the ring buffer falls below 30.

```
EN_BUFBRK 30
```

## EN\_ENCFLT

---

**FUNCTION** Encoder Fault Interrupt

**SYNTAX** EN\_ENCFLT n, m, fer

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n	long value specifying the axis
m	long value, bit coding of the axes interrupt condition (see Description)
fer	double precision, unsigned following error value

$0 \leq \text{fer} \leq 65535$  counts

### DESCRIPTION

This command enables the encoder fault interrupt for the specified axes.

With the respective axis bit of argument m equal to 0, the encoder fault interrupt is triggered for the axis in question if,

1.  $\text{abs}[\text{following error}] > \text{ferr threshold}$
2. and, hardware encoder status bit is set

With the respective axis bit of argument m equal to 1, the encoder fault interrupt is triggered for the axis in question if,

1.  $\text{abs}[\text{following error}] > \text{ferr threshold}$

If an encoder fault interrupt condition is present for an axis, the axis will be put into open loop with DAC output of 0 volts, and an interrupt will be generated. If, however, the axis in question is already in open

## EN\_ENCFLT cont.

---

loop prior to the interrupt condition, an interrupt will be generated but no action will be taken (ie: DAC voltage is unaffected).

The encoder fault interrupt is sustained until the EN\_ENCFLT command is reissued to the Mx4. Reissuing the EN\_ENCFLT command also allows the affected axis(es) to be put back into closed loop following the execution of the command.

The hardware encoder status bits are reported to the lower nibble of DPR location 113h (see Mx4 DPR Organization). A set bit indicates that Mx4 has detected an encoder hardware failure. Mx4 reports an “encoder status” error if for the axis in question,

1. the encoder feedback to Mx4 is losing encoder pulses or one of the encoder signals (A or B) actively toggles while the other one is inactive.

The DPR interrupt status locations 009h (bit 4) and 00Eh record the occurrence and source of this interrupt, respectively. Bit 6 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

**SEE ALSO**      DISABL2\_INT

### APPLICATION

A necessary diagnostic feature for all servo control applications.

#### **Command Sequence Example**

No preparation is required before running this instruction.

### EXAMPLE

Enable the encoder fault interrupt for axis 3. Set the following error threshold at 500 counts, using the encoder hardware status bits in the interrupt condition.

```
EN_ENCFLT 3, 0, 500
```

## EN\_ERR

---

**FUNCTION** Enable Following Error Interrupt

**SYNTAX** EN\_ERR n, fer

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis  
fer double precision, unsigned following error value

$0 \leq \text{fer} \leq 65535$  counts

### DESCRIPTION

Upon the execution of this command, if at any time the following error for a specified axis exceeds its programmed value, the servo control card will generate an interrupt. This condition is recorded in DPR interrupt status register location 000h. The DPR status register location 02h will identify the axis(es) responsible. Bit 1 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also axis bit-coded in the DSPL FERR\_REG bit register.



**Note:** EN\_ERR is not disabled after it occurs. The host is responsible for disabling the interrupt.

**SEE ALSO** DISABL\_INT, EN\_ERRHLT

### APPLICATION

This command may be used in all applications for two reasons. First, EN\_ERR reports a run-away or any other out-of-control condition. Second, it makes sure that position error is within a specified tolerance (i.e. the value in argument fer).

## **EN\_ERR cont.**

---

### ***Command Sequence Example***

No preparation is required before running this instruction.

### **EXAMPLE**

Set a `EN_ERR` interrupt value of 200 encoder counts for axis 1.

```
EN_ERR 1, 200
```

## EN\_ERRHLT

---

**FUNCTION** Enable Following Error Interrupt and Halt

**SYNTAX** EN\_ERRHLT n, fer

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis  
fer double precision, unsigned following error value

$0 \leq \text{fer} \leq 65535$  counts

### DESCRIPTION

Upon execution of this command, if at any time the following error for a specified axis exceeds its programmed value, the system will halt and generate an interrupt. The halt brings the motion of the axis in question to a stop using the programmed maximum acceleration rate. This interrupt condition is recorded in DPR interrupt status register location 000h. The DPR status register location 001h reveals the axis(es) responsible. Bit 1 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also axis bit-coded in the DSPL FERRH\_REG bit register.



**Note 1:** EN\_ERRHLT will be ignored if the respective axis abort maximum acceleration is zero.



**Note 2:** EN\_ERRHLT is not disabled after it occurs. The host is responsible for disabling the interrupt.

## EN\_ERRHLT cont.

---

**SEE ALSO**     DISABL\_INT, EN\_ERR, ESTOP\_ACC

### APPLICATION

Applications of this command are similar to EN\_ERR. However, as a result of this command's interrupt, the system will come to a stop. Stop trajectory uses the programmed abort maximum acceleration. Please see ESTOP\_ACC. Please note that this command is not appropriate to prevent system run-away in case of encoder loss, since in the absence of encoder, the system cannot be stopped reliably.

#### **Command Sequence Example**

```
ESTOP_ACC ( ) ;set the maximum accel. so system can be stopped
CTRL ( )      ;these instructions enable system to stop motion
KILIMIT ( )   ;set gains
.
.
EN_ERRHLT ( )
```

### EXAMPLE

Enable a following error/halt interrupt for axis 1, 2 and 3 with a threshold of 100, 120, and 200 counts, respectively.

```
BEGIN_RTC
    EN_ERRHLT 1, 100
    EN_ERRHLT 2, 120
    EN_ERRHLT 3, 200
END_RTC
```

## EN\_INDEX

---

**FUNCTION** Enable Index Pulse Interrupt

**SYNTAX** EN\_INDEX n

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis

### DESCRIPTION

Upon the execution of this command, the servo control card will search for the first index pulse edge from the specified axis. The pulse edge generates an interrupt and registers the actual position for all axes in DPR locations 103h - 112h. The DPR interrupt status register locations 000h and 003h record the occurrence and source of this interrupt. Bit 1 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also axis bit-coded in the DSPL INDEX\_REG bit register.



**Note 1:** Only one index pulse can generate an interrupt at any given time. The EN\_INDEX command enables the index pulse interrupt for the axis specified and automatically disables the previous one (if any).



**Note 2:** The EN\_INDEX and EN\_PROBE commands CAN BE ENABLED simultaneously.

**SEE ALSO** DISABL\_INT, POS\_PRESET, POS\_SHIFT

## **EN\_INDEX cont.**

---

### **APPLICATION**

This command is used in homing applications. As a result of this instruction, Mx4 will start searching for the first index pulse edge. Upon the detection of an index pulse edge, position of the axis is immediately recorded. This instruction must be used in conjunction with POS\_PRESET to perform homing for linear table (or other index-based) position calibration.

#### ***Command Sequence Example***

No preparation is required before running this instruction.

### **EXAMPLE**

Enable the index pulse interrupt for axis 4.

```
EN_INDEX 4
```

## EN\_MOTCP

---

**FUNCTION** Enable Motion Complete Interrupt

**SYNTAX** EN\_MOTCP n

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis

### DESCRIPTION

This command enables the motion complete interrupt for the axes specified. The motion complete interrupt is generated when any closed loop motion other than ring buffer 2nd order or ring buffer cubic spline contouring comes to a stop. The DPR interrupt status register locations 000h and 005h record the occurrence and source of this interrupt. Bit 1 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also bit-coded in the DSPL MOTCP\_REG bit register.



**Note:** EN\_MOTCP is not disabled after it occurs. The host is responsible for disabling the interrupt.

**SEE ALSO** DISABL\_INT

### APPLICATION

In any application that a new routine must run based on the end of a motion, this command informs the host of motion completion. An example of such an application is milling in which the spindle and z-axes will start moving only when the x-y table has moved to a target position.

#### **Command Sequence Example**

See AXMOVE and STOP

## **EN\_MOTCP cont.**

---

### **EXAMPLE**

Enable the motion complete interrupt for all four axes.

```
BEGIN_RTC
    EN_MOTCP 1
    EN_MOTCP 2
    EN_MOTCP 3
    EN_MOTCP 4
END_RTC
```

## EN\_POSBRK

---

**FUNCTION** Enable Position Breakpoint Interrupt

**SYNTAX** `EN_POSBRK n, pos`

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n	long value specifying the axis
pos	double precision, position breakpoint value $-2147483648 \leq \text{pos} \leq 2147483647$ counts

### DESCRIPTION

This command enables the position breakpoint interrupt for the axes specified. The position breakpoint interrupt is generated when the actual position for a specified axis passes the programmed breakpoint. The DPR interrupt status register locations 000h and 004h record the occurrence and source of this interrupt. Bit 1 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also axis bit-coded in the DSPL `POSBRK_REG` bit register.



**Note 1:** The position breakpoint is calculated as the absolute distance from the present position (position at the moment at which the `EN_POSBRK RTC` is interpreted) to the position breakpoint value entered. The breakpoint interrupt is set when the axis in question travels (in either direction) a distance equal to the calculated absolute distance.

## EN\_POSBRK cont.

---



**Note 2:** EN\_POSBRK is automatically disabled after the breakpoint interrupt is generated. To activate this interrupt again, the host must issue a new EN\_POSBRK command.



**Note 3:** POS\_PRESET and POS\_SHIFT will automatically disable the position breakpoint interrupt. The user is responsible to re-enable the interrupt.

**SEE ALSO**     DISABL\_INT, POS\_PRESET, POS\_SHIFT

### APPLICATION

This instruction may be used in applications such as robotics, indexing machine tools, etc. The CPU must be notified that the system has passed an intermediate position. Based on this interrupt, the CPU will execute a task. For example, in a robotics painting application, the paint mixture may have to change based on the robot's arm location.

#### **Command Sequence Example**

```
MAXACC ( )        ;set the maximum accel. so system can be stopped
CTRL ( )         ;set the gains
KILIMIT ( )
OUTGAIN ( )
```

### EXAMPLE

Enable a breakpoint interrupt with a value of 60,000 counts for axis 1 and 500,000 for axis 2.

```
BEGIN_RTC
    EN_POSBRK 1, 60000
    EN_POSBRK 2, 500000
END_RTC
```

## EN\_PROBE

---

**FUNCTION** Enable General Purpose External Interrupt

**SYNTAX** EN\_PROBE m

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

m long value specifying the probe

[Mx4]

m=1 : from \*EXT1

m=2 : from \*EXT2

[Mx4 Octavia]

m=1 : from \*EXT1

m=2 : from \*EXT2

m=3 : from \*EXT3

m=4 : from \*EXT4

### DESCRIPTION

Upon the execution of this command, the servo control card will search for the first \*EXTx pulse edge. The pulse edge generates an interrupt and registers the actual position for all axes in DPR locations 0A7h-0B6h. (The hand shaking bytes are 0C8h and 0D0h for Mx4 and host, respectively.) DPR interrupt status register locations 000h and 006h record the occurrence and source of this interrupt. Bit 1 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also axis bit-coded in the DSPL PROBE\_REG bit register.

## EN\_PROBE cont.

---



**Note 1:** Only one general purpose external interrupt can generate an interrupt at any given time. The EN\_PROBE command enables the external interrupt specified and automatically disables the previous one (if any).



**Note 2:** The EN\_PROBE and EN\_INDEX can be enabled simultaneously.

**SEE ALSO**     DISABL\_INT, ESTOP\_ACC

### APPLICATION

This instruction is useful in probing applications. Since EN\_PROBE registers all positions when an interrupt occurs (falling pulse edge is detected), it can be used in accurate recording of surface dimensions by a probe.

#### **Command Sequence Example**

```
CTRL ( )            ;these instructions enable system to stop motion
KILIMIT ( )
.
.
EN_PROBE ( )
END
```

### EXAMPLE

Enable the \*EXT2 external interrupt.

```
EN_PROBE 2
```

## ESTOP\_ACC

---

**FUNCTION** Abort Motion Maximum Acceleration

**SYNTAX** ESTOP\_ACC n, acc

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n	long value specifying the axis
acc	double precision, unsigned value specifying the maximum halting acceleration (deceleration)

$$0 \leq \text{acc} \leq 1.999969 \text{ counts}/(200\mu\text{s})^2$$



**Note:** Acceleration is partitioned into 1 bit integer, 15 bits fraction.

### DESCRIPTION

This command specifies the maximum halting acceleration (deceleration) for the axes specified. The maximum acceleration values are used in the following cases: EN\_ERRHLT, and ESTOP\_ACC.



**Note:** ESTOP\_ACC will be ignored if the specified argument is zero.

**SEE ALSO** EN\_ERRHLT, MAXACC, STOP, VELMODE

## ESTOP\_ACC cont.

---

### APPLICATION

This command sets the maximum possible deceleration for a mechanical actuator. This RTC is used to set the deceleration rate for an emergency case. In contrast to MAXACC, ESTOP\_ACC provides a sharper deceleration such that the entire system comes to a stop as rapidly as possible. Please remember that the STOP and VELMODE RTCs use MAXACC for their acceleration/deceleration.

#### **Command Sequence Example**

```
ESTOP_ACC ( ) ;set the abort maximum acceleration
CTRL ( ) ;make sure the system is in closed loop
EN_ERRHLT ( ) ;set the maximum tolerance for the following error
                ;if the following error exceeds the ABORTACC
                ;parameter, the system will stop immediately
```

### EXAMPLE

Set an abort motion maximum acceleration for axes 2 and 3 of 0.5 encoder counts/(200 μsec)<sup>2</sup>.

```
BEGIN_RTC
    ESTOP_ACC 2, 0.5
    ESTOP_ACC 3, 0.5
END_RTC
```

## **FERR**

---

**FUNCTION**     Get Following Error State Variable

**SYNTAX**        FERR n

### **ARGUMENTS**

n                long value specifying the axis

### **DESCRIPTION**

This function returns a double precision value, the following error for the axis specified.

**SEE ALSO**     POS, VEL

### **EXAMPLE**

Read the following error of axis 3.

```
Dim Temp As Double
Temp = FERR (3)
```

**FLUX\_CURRENT**

Vx4++ option command

**FUNCTION** Set Field Compensation Or Flux Value**SYNTAX** FLUX\_CURRENT n, fval

If used as a function, the function will return (long) zero if successful, nonzero if error.

**ARGUMENTS**

n long value specifying the axis  
 fval long value, for AC induction motor, defines a bipolar flux value for the field producing component of the current

$$-32768 \leq \text{fval} \leq 32767$$

for brushless DC motor, defines a unipolar field compensation parameter

$$0 \leq \text{fval} \leq 65535$$

**DESCRIPTION**

The FLUX\_CURRENT command defines motor technology-dependent parameters. If the axis in question is an AC induction motor, the command defines a bipolar flux value for the field-producing component of the current. If the axis is a brushless DC motor, the command sets a unipolar field compensation parameter.



**Note:** The FLUX\_CURRENT command does not need to be programmed for brushtype DC motors.

**SEE ALSO** none**APPLICATION**

See *Vx4++ User's Guide*

## **FLUX\_CURRENT cont.**

---

### **EXAMPLE**

Set a flux value of -5000 for axis one (AC induction motor) and a field compensation value of 1300 for axis two (brushless DC motor).

```
BEGIN_RTC  
    FLUX_CURRENT 1, -5000  
    FLUX_CURRENT 2, 1300  
END_RTC
```

## GEAR

---

**FUNCTION** Electronics Gear On

**SYNTAX** GEAR n, m, ratio

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the master axis  
m long value specifying the slave axis  
ratio single precision, gear ratio between master and slave

$-256 \leq \text{ratio} < 255.999$

minimum gear ratio is +/- 1/128

### DESCRIPTION

This command emulates the mechanical gear function. The follower follows the leader with the gear ratio specified by ratio. Upon receiving this command, the electronic gearing is engaged at once.

**SEE ALSO** GEAR\_OFF, GEAR\_POS, GEAR\_PROBE

### APPLICATION

See *Application Notes*

### EXAMPLE

Axis 2 is a slave axis to axis 1 with a gear ratio of 2.5.

```
GEAR 1, 2, 2.5
```

## GEAR\_OFF

---

**FUNCTION**     Electronics Gear Off

**SYNTAX**        `GEAR_OFF n`

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

`n`                long value specifying the slave axis to disengage

### DESCRIPTION

This command disengages the specified slave axis(es) at once.

**SEE ALSO**        `GEAR`, `GEAR_POS`, `GEAR_PROBE`

### APPLICATION

See *DSPL Application Notes*

### EXAMPLE

Axis 1 is the leader, axis 3 and 4 are the followers (slaves). Disengage only axis 4.

```
GEAR_OFF 4
```

## **GEAR\_OFF\_ACC**

---

**FUNCTION** Turns Electronic Gearing Off and Halt Slave(s)

**SYNTAX** GEAR\_OFF\_ACC n

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

n long value specifying the slave axis to disengage

### **DESCRIPTION**

This command disengages the system that was under master/slave control. The slave axes will come to a complete stop at the maximum acceleration rate specified by MAXACC command.

**SEE ALSO** GEAR, GEAR\_OFF, GEAR\_POS, GEAR\_PROBE

### **APPLICATION**

Axis 1 is the leader, axis 3 and 4 are the followers (slaves). Disengage only axis 4.

GEAR\_OFF\_ACC 4

## GEAR\_POS

---

**FUNCTION**     Electronics Gear On at a Specified Leader Position

**SYNTAX**        GEAR\_POS n, m, ratio, tp

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n	long value specifying the master axis
m	long value specifying the slave axis
ratio	single precision, gear ratio between master and slave
	$-256 \leq \text{ratio} < 255.999$
	minimum gear ratio is +/- 1/128
tp	double precision, master axis position value at which the electronic gearing engages for the specified axis
	$-2147483648 \leq \text{tp} \leq 2147483647$

### DESCRIPTION

This command emulates a mechanical gear function. The slave follows the master with the gear ratio specified by ratio. Upon receiving this command, the electronic gearing starts engaging at the specified master position (tp).

**SEE ALSO**        GEAR, GEAR\_OFF, GEAR\_PROBE

### APPLICATION

See *DSPL Application Notes*

## **GEAR\_POS cont.**

---

### **EXAMPLE**

Axes 3 and 4 should follow axis 2 with gear ratios 2.0 and 4.0, respectively. Both axes three and four should “engage” when axis 2 position is equal to 10,500 counts.

```
BEGIN_RTC
    GEAR_POS  2, 3, 2.0, 10500
    GEAR_POS  2, 4, 4.0, 10500
END_RTC
```

## GEAR\_PROBE

---

**FUNCTION** Electronics Gear On After Probe Input

**SYNTAX** GEAR\_PROBE n, m, q, ratio

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the master axis  
m long value specifying the slave axis  
q long value, the \*EXTx probe input to be used

[Mx4]

q = 1 : \*EXT1  
q = 2 : \*EXT2

[Mx4 Octavia]

q = 1 : \*EXT1  
q = 2 : \*EXT2  
q = 3 : \*EXT3  
q = 4 : \*EXT4

ratio single precision, gear ratio between master and slave

$-256 \leq \text{ratio}_x < 255.999$

minimum gear ratio is +/- 1/128

### DESCRIPTION

This command emulates the mechanical gear function. The follower follows the leader with the gear ratio specified by  $r_x$ . The GEAR\_PROBE command engages the mechanical gear function for selected master and slave axes after the specified external signal (\*EXTx) is activated.

## GEAR\_PROBE cont.

---



**Note 1:** Execution of the `GEAR_PROBE` command will disable any previously enabled `EN_PROBE` interrupt. Probe (\*EXT1,2,3,4) activation does *not* generate an interrupt with the `GEAR_PROBE` command.



**Note 2:** Activation of \*ESTOP during a GEAR operation will halt the master axis, and subsequently the slave axis(es). Slave(s) remain “engaged” in GEAR mode after the input-triggered halt.

**SEE ALSO**     `GEAR`, `GEAR_OFF`, `GEAR_POS`

### APPLICATION

See *DSPL Application Notes*

### EXAMPLE

Axis 8 is the leader, axis 1 is the follower with a gear ratio of 4.0. Axis 1 should “engage” at the occurrence of probe interrupt \*EXT2.

```
GEAR_PROBE 8, 1, 2, 4.0
```

## GETCOMMINSTCOUNT

---

**FUNCTION** Get Number of Serial Communication Instances Connected

**SYNTAX** GETCOMMINSTCOUNT (port)

**ARGUMENTS**

Port long value specifying comm port (1-4)

**DESCRIPTION**

This function is used with serial communication applications. The function returns (long value) the number of instances of the DLL during serial communication which have been successfully connected.

**SEE ALSO** CHANGECOMMPORTSETTING, CHANGESLAVENODEADDRESS,  
COMMUNICATIONSLOST, GETCOMMTYPE,  
GETCURRENTNODEADDRESS, RESETCOMMUNICATIONS

**EXAMPLE**

Read the number of instances connected over comm port 2 into Visual Basic variable Temp.

```
Dim Temp As Long  
Temp = GETCOMMINSTCOUNT (2)
```

## GETCOMMTYPE

---

**FUNCTION** Get Communication Type

**SYNTAX** GETCOMMTYPE ()

**ARGUMENTS**

none

**DESCRIPTION**

This function is used to determine the current communication type. The function returns (byte value) the type as follows,

0	bus
1	comm1
2	comm2
3	comm3
4	comm4
5	none

**SEE ALSO** CHANGECOMMPORTSETTING, CHANGESLAVENODEADDRESS, COMMUNICATIONSLOST, GETCOMMINSTCOUNT, GETCURRENTNODEADDRESS, RESETCOMMUNICATIONS

**EXAMPLE**

Query the type of communication which is active.

```
Dim Temp As Byte
Temp = GETCOMMTYPE ()
```

## **GETCURRENTNODEADDRESS**

---

**FUNCTION** Get Current Serial Communication Node Address

**SYNTAX** GETCURRENTNODEADDRESS ()

**ARGUMENTS**

none

**DESCRIPTION**

This function returns (byte value) the current value of the node address for serial communication applications.

**SEE ALSO** CHANGECOMMPORTSETTING, CHANGESLAVENODEADDRESS, COMMUNICATIONSLOST, GETCOMMINSTCOUNT, GETCOMMTYPE, RESETCOMMUNICATIONS

**EXAMPLE**

Query the serial communication node address.

```
Dim Temp As Byte
Temp = GETCURRENTNODEADDRESS ()
```

## **GETNUMBEROFAXES**

---

**FUNCTION**     Get the number of axes

**SYNTAX**       GETNUMBEROFAXES ()

**ARGUMENTS**

                  none

**DESCRIPTION**

                  This function returns (integer) the current number of axes

**SEE ALSO**

**EXAMPLE**

                  Query the number of axes.

```
                  Dim AxesCount As Integer  
                  AxesCount = GETNUMBEROFAXES ()
```

## INP\_STATE

---

**FUNCTION**     Configure Logic State of Inputs

**SYNTAX**        `INP_STATE inp, state`

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

`inp`            long value, specifying the input

<code>[Mx4]</code>	<code>0 &lt;= inp &lt;= 21</code>
<code>[Mx4 Octavia]</code>	<code>0 &lt;= inp &lt;= 31</code>
<code>[IOExp]</code>	<code>0 &lt;= inp &lt;= 63</code>
<code>[IOExp:2]</code>	<code>0 &lt;= inp &lt;= 127</code>

`state`          long value, specifying the logic state of the input

<code>state = 0</code>	:	active LOW input
<code>state = 1</code>	:	active HIGH input

### DESCRIPTION

This command allows the user to define the logic state of the [Mx4:22] [Mx4 Octavia:32] inputs. Each input may be configured as active LOW or active HIGH (TTL logic levels) (the Mx4 inputs are level sensitive).



**Note:** At power-up and reset, Mx4 inputs default as active LOW.

**SEE ALSO**     none

### EXAMPLE

Configure the IN0 and IN5 inputs as active HIGH.

```
BEGIN_RTC
    INP_STATE 0, 1
    INP_STATE 5, 1
END_RTC
```

## INT5MS

---

**FUNCTION** Enable / Disable the 5msec Interrupt

**SYNTAX** INT5MS m

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

m long value specifying the enable/disable status of the interrupt

m = 0 disable the 5msec interrupt

m = 1 enable the 5msec interrupt

### DESCRIPTION

The 5msec interrupt is useful in sampling applications where the host may need to sample data from the Mx4 controller at timed intervals. When enabled, the Mx4 controller issues a hardware interrupt to the host every 5msec. The interrupt is coded in DPR location 009h. Bit 6 of DPR location [Mx4:7Feh] [Mx4 Octavia:1FFEh] is also set.

**SEE ALSO** none

### EXAMPLE

Enable the 5msec interrupt.

```
INT5MS 1
```

## KILIMIT

---

**FUNCTION**    Integral Gain Limit

**SYNTAX**        `KILIMIT n, val`

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

`n`                long value specifying the axis  
`val`              long value setting the limit of the integral action



**Note:**  $0 \leq \text{val} \leq 14$

`val = 0` indicates no limit on integration channels  
`val = 14` indicates maximum limit on integration channels

For example,

`Kilimit val = 0`    +/- 10v DAC action from  $K_i$  control law parameter  
`Kilimit val = 1`    +/- 5v DAC action from  $K_i$  control law parameter  
`Kilimit val = 2`    +/- 2.5v DAC action from  $K_i$  control law parameter  
`Kilimit val = 3`    +/- 1.25v DAC action from  $K_i$  control law parameter  
:  
:

### DESCRIPTION

This command is used to set the limit for integral action related to the choice of  $\text{par}_{x1}$  in the CTRL RTC. Integral limit is specified for each axis. Default  $\text{val}_x$  are set to zero (i.e., no limit on integration channels).

**SEE ALSO**        CTRL

## KILIMIT cont.

---

### APPLICATION

This command clamps the integral channel by reducing this channel's saturation level. Reducing the saturation level will reduce the channel's depletion time. Using this instruction is essential where large integral gain is required. Clamping the integral channel will let the system zero position error without a lengthy "creeping motion" to its target position.

#### **Command Sequence Example**

```
CTRL ( ) ;set the gains  
KILIMIT ( ) ;this instruction may be used before or after CTRL
```

### EXAMPLE

Set a maximum limit on the integral action of axis 2, 3, and 4.

```
BEGIN_RTC  
    KILIMIT 2, 14  
    KILIMIT 3, 14  
    KILIMIT 4, 14  
END_RTC
```

## LOW\_PASS (option)

**FUNCTION** Implement Low Pass Filter at Controller Output

**SYNTAX** LOW\_PASS n, freq

If used as a function, the function will return (long) zero if successful, nonzero if error.

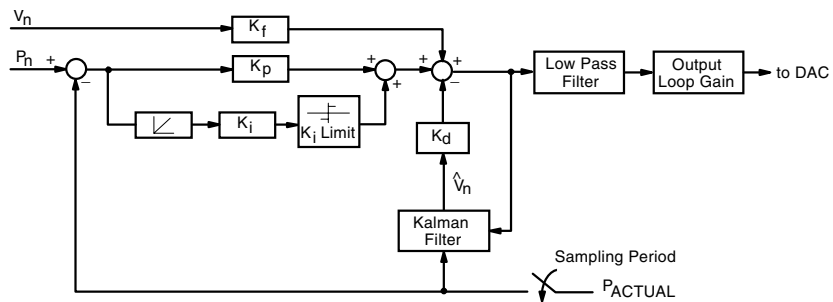
### ARGUMENTS

n long value specifying the axis  
 freq long value specifying the low pass filter cut-off frequency

$$0 \leq \text{freq} \leq 1850$$

### DESCRIPTION

This command implements a low pass filter at the controller output for the specified axis.



Mx4 Block Diagram with Low Pass Filter

## LOW\_PASS cont.

---

The low pass filter implements the following transfer function:

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n \cdot s + \omega_n^2}$$

where,  $\omega_n = 2\pi f_n$ ,  $f_n$  = cut-off frequency, and  $\zeta = 0.6$

The frequency and bandwidth of the low pass filter is programmable.



**Note:** By programming a cut-off frequency of 0, the low pass filter for the specified axis is disabled.

**SEE ALSO** none

### EXAMPLE

- 1) Set a low pass filter at 250 Hz for axis 2 (see below).

```
LOW_PASS 2, 250
```

- 2) Disable the low pass filter of axis 1.

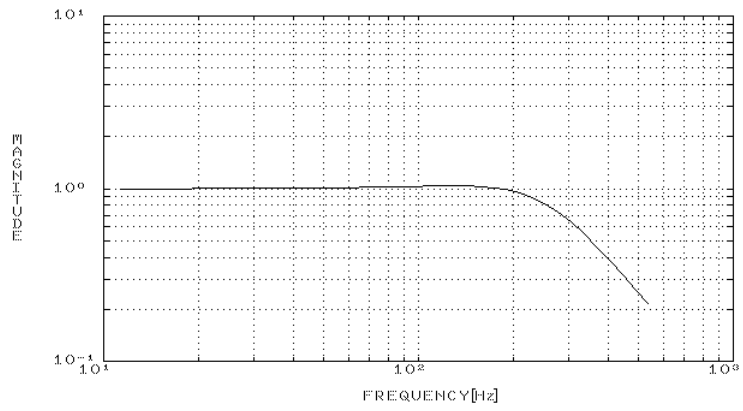
```
LOW_PASS 1, 0
```



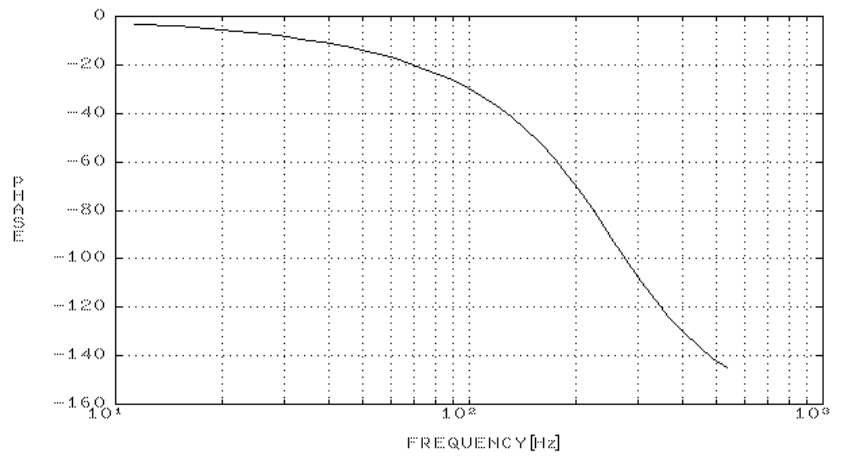
**Note:** Mx4 default setting for low pass filter is no filter (or filter disabled).

## LOW\_PASS cont.

---



Magnitude Diagram



Phase Diagram of 250 Hz Low Pass Filter

## MAXACC

---

**FUNCTION** Maximum Acceleration

**SYNTAX** MAXACC n, acc

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis  
acc double precision, unsigned value specifying the maximum acceleration / deceleration

$$0 \leq \text{acc} \leq 1.999969 \text{ counts}/(200\mu\text{s})^2$$

### DESCRIPTION

This command specifies the maximum acceleration / deceleration for the axes specified. The maximum acceleration values are used in the STOP and VELMODE commands.



**Note:** MAXACC will be ignored if the specified argument is zero.

**SEE ALSO** ESTOP\_ACC, STOP, VELMODE

## MAXACC cont.

---

### APPLICATION

This command sets the maximum acceleration affordable by the servo drive and motor combination. It is useful to program this parameter such that the system will not go to control saturation during VELMODE or STOP.

#### **Command Sequence Example**

```
MAXACC ( ) ;set the maximum accel. so system can be stopped
CTRL ( ) ;set the gains
KILIMIT ( )
.
.
AXMOVE ( ) ;run system in axis move
VELMODE ( ) ;run system in velocity mode
```

### EXAMPLE

Set a maximum acceleration for axes 2 and 3 of 0.25 encoder counts / (200 $\mu$ s)<sup>2</sup>.

```
BEGIN_RTC
    MAXACC 2, 0.25
    MAXACC 3, 0.25
END_RTC
```

## MONITOR\_VAR

---

**FUNCTION**     Select DSPL Variables To Monitor

**SYNTAX**       MONITOR\_VAR   m, var

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

m               long value specifying the DPR variable monitoring window

1 <= m <= 4

var             long value specifying the DSPL variable to monitor in the selected DPR monitoring window

1 <= var <= 128

### DESCRIPTION

The MONITOR\_VAR function selects which of the 128 DSPL variables to report to the DPR variable monitoring windows (there are 4 windows, for a maximum of 4 variables at one time).

**SEE ALSO**     CHANGE\_VAR, VAR

### EXAMPLE

Select DSPL variable VAR67 to be reported to DPR monitor window 4.

```
MONITOR_VAR 4, 67
```

## MOTOR\_PAR

Vx4++ option command

**FUNCTION** Motor Parameter

**SYNTAX** MOTOR\_PAR n, mpar

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis  
mpar long value, for AC induction motor, defines the motor slip gain

$-32768 \leq \text{fval} \leq 32767$

for brushless DC motor, defines the commutation angle

$-32768 \leq \text{fval} \leq 32767$

### DESCRIPTION

The MOTOR\_PAR command defines motor technology-dependent parameters. If the axis in question is an AC induction motor, the command defines the motor slip gain. If the axis is a brushless DC motor, the command defines the commutation angle (in encoder counts).



**Note:** The MOTOR\_PAR command does not need to be programmed for brushtype DC motors.

**SEE ALSO** none

### APPLICATION

See *Vx4++ User's Guide*

### EXAMPLE

Program a slip gain equal to 5500 for axes two (the motor is an AC induction motor).

```
MOTOR_PAR 2, 5500
```

**MOTOR\_TECH**

Vx4++ option command

**FUNCTION** Motor Technology**SYNTAX** MOTOR\_TECH n, mtech

If used as a function, the function will return (long) zero if successful, nonzero if error.

**ARGUMENTS**

n long value specifying the axis  
 mtech long value,  
 AC induction, mtech = MT\_AC\_INDUCTION  
 brushless DC, mtech = MT\_BRUSHLESS\_DC  
 brushtype DC, mtech = MT\_BRUSHTYPE\_DC

**DESCRIPTION**

Mx4 with the Vx4++ drive control option is capable of controlling brushtype DC, AC induction, and brushless DC motors. This command allows the motor technology of each axis to be programmed.



**Note:** Mx4 with Vx4++ will not execute the MOTOR\_TECH command if the Vx4\_BLOCK command is active for the axes in question.

**SEE ALSO** Vx4\_BLOCK**APPLICATION**

See *Vx4++ User's Guide*

**EXAMPLE**

Select brushless DC technology for axis one.

```
MOTOR_TECH 1, MT_BRUSHLESS_DC
```

## MX4\_CLEAR

---

**FUNCTION** Clear Interrupt Conditions

**SYNTAX** `MX4_CLEAR mask, n`

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

`mask` long value, mask specifying the interrupt condition to clear. Each bit in the mask corresponds to a specific interrupt condition. If a bit is set, the corresponding interrupt condition is cleared.

- bit 0: following error & halt [EN\_ERRHLT]
- bit 1: following error [EN\_ERR]
- bit 2: index pulse [EN\_INDEX]
- bit 3: position breakpoint [EN\_POSBRK]
- bit 4: motion complete [EN\_MOTCP]
- bit 5: probe [EN\_PROBE]
- bit 6: conflicting commands
- bit 7: RTC ignored, stop in progress
- bit 8: encoder fault [EN\_ENCFLT]
- bit 9: <reserved>
- bit 10: offset cancel finished [OFFSET]
- bit 11: <reserved>
- bit 12: DSPL host interrupt [INT\_HOST, dspl]
- bit 13: DSPL program running error
- bit 14: 5 msec [INT5MS]
- bit 15: <reserved>

`n` long value specifying the axis

## **MX4\_CLEAR cont.**

---

### **DESCRIPTION**

The source of interrupts are recorded in the Mx4 controller's Dual Port RAM (DPR). When a specific interrupt occurs a bit is set in the DPR. The `MX4_ISTAT` function can be used to test these bits. Mx4 never clears these bits, this must be done by the host program via the `MX4_CLEAR` function.

**SEE ALSO**     `DISABL_INT`,     `DISABL2_INT`,     `EN_BUFBRK`,     `EN_ERR`,  
                 `EN_ERRHLT`, `EN_INDEX`, `EN_MOTCP`, `EN_POSBRK`, `EN_PROBE`,  
                 `INT5MS`, `MX4_ISTAT`

### **APPLICATION**

### **EXAMPLE**

Clear the axis 3 following error and halt interrupt.

```
MX4_CLEAR  &H1, 3
```

## MX4\_INPUT

---

**FUNCTION** Return State Of Digital Input

**SYNTAX** MX4\_INPUT inp

### ARGUMENTS

inp long value, specifying the input

Mx4 :	0 <= inp <= 21
Mx4 Octavia :	0 <= inp <= 31
IOExp :	0 <= inp <= 63
IOExp ,2:	0 <= inp <= 127

### DESCRIPTION

This function returns (long value) the on / off status of the specified input. If the input is on, the function returns a non-zero value. If the input is off, the function returns 0.

**SEE ALSO** INP\_STATE, OUTP\_OFF, OUTP\_ON

### APPLICATION

### EXAMPLE

Read the state of each of Mx4 Octavia's 32 inputs into the local array Temp.

```
For I=0 To 31
  Temp[I]=MX4_INPUT (I)
Next I
```

## MX4\_ISTAT

---

**FUNCTION** Test State Of Interrupt Conditions

**SYNTAX** MX4\_ISTAT mask, n

### ARGUMENTS

mask long value, mask specifying the interrupt condition to check. Each bit in the mask corresponds to a specific interrupt condition. If a bit is set, the corresponding interrupt condition is checked.

bit 0: following error & halt [EN\_ERRHLT]  
 bit 1: following error [EN\_ERR]  
 bit 2: index pulse [EN\_INDEX]  
 bit 3: position breakpoint [EN\_POSBRK]  
 bit 4: motion complete [EN\_MOTCP]  
 bit 5: probe [EN\_PROBE]  
 bit 6: conflicting commands  
 bit 7: RTC ignored, stop in progress  
 bit 8: encoder fault [EN\_ENCFLT]  
 bit 9: <reserved>  
 bit 10: offset cancel finished [OFFSET]  
 bit 11: <reserved>  
 bit 12: DSPL host interrupt [INT\_HOST, dspl]  
 bit 13: DSPL program running error  
 bit 14: 5 msec [INT5MS]  
 bit 15: <reserved>

n long value specifying the axis

## MX4\_ISTAT cont.

---

### DESCRIPTION

The source of interrupts are recorded in the Mx4 controller's Dual Port RAM (DPR). When a specific interrupt occurs a bit is set in the DPR. The `MX4_ISTAT` function can be used to test these bits (the function returns a long value, non-zero if any of the interrupt conditions in the mask is true). Mx4 never clears these bits, this must be done by the host program via the `MX4_CLEAR` function.

### SEE ALSO

`DISABL_INT`, `DISABL2_INT`, `EN_BUFBRK`, `EN_ERR`,  
`EN_ERRHLT`, `EN_INDEX`, `EN_MOTCP`, `EN_POSBRK`, `EN_PROBE`,  
`INT5MS`, `MX4_CLEAR`

### APPLICATION

### EXAMPLE

Poll for an axis 1 index pulse interrupt. The timer event procedure is used to poll for the interrupt condition. When it is detected, a counter is incremented and the interrupt bit is cleared.

```
Sub Timer_T()  
.  
.  
  If MX4_ISTAT(IC_INDEX_PULSE, 1) Then  
    InterruptCount = InterruptCount + 1  
    MX4_CLEAR IC_INDEX_PULSE, 1  
  End If  
.  
.  
End Sub
```

## NOTCH (option)

**FUNCTION** Implement Notch Filter at Controller Output

**SYNTAX** NOTCH n, freq, q

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

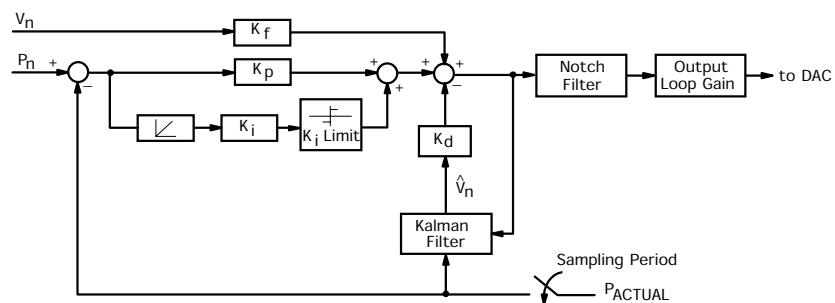
n long value specifying the axis  
 freq long value specifying the notch filter frequency  
 $0 \leq \text{freq} \leq 1650 \text{ Hz}$

q long value specifying the notch filter quality factor

q = 1 ~25% bandwidth filter  
 q = 2 ~10% bandwidth filter

### DESCRIPTION

This command implements a notch filter at the controller output for the specified axis.



Mx4 Block Diagram with Notch Filter

## NOTCH cont.

---

The notch filter implements the transfer function:

$$G(s) = \frac{s^2 + \omega_n^2}{s^2 + \frac{\omega_n}{Q}s + \omega_n^2}$$

where,  $\omega_n = 2\pi f_n$  and  $f_n$  = notch frequency

The frequency and bandwidth of the notch is programmable.



**Note:** By programming a notch frequency of 0, the notch filter for the specified axis is disabled.

**SEE ALSO** none

### EXAMPLE

- 1) Set a notch filter at 750 Hz with a narrow bandwidth ( $q = 2$ ) for axis 2 (see Fig. 4-3 below).

```
NOTCH 2, 750, 2
```

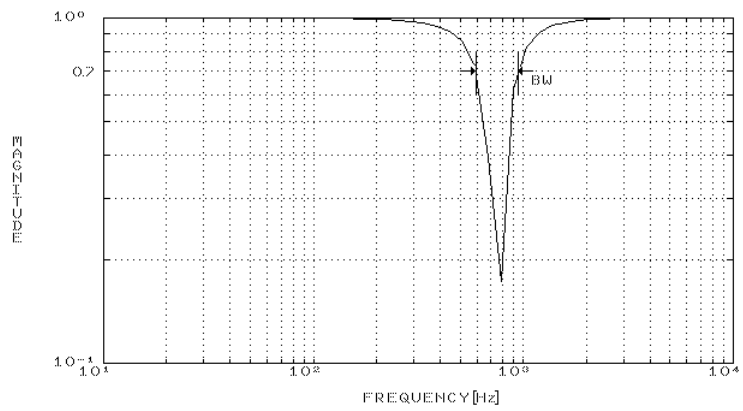
- 2) Disable the notch filter of axis 1.

```
NOTCH 1, 0, 1
```

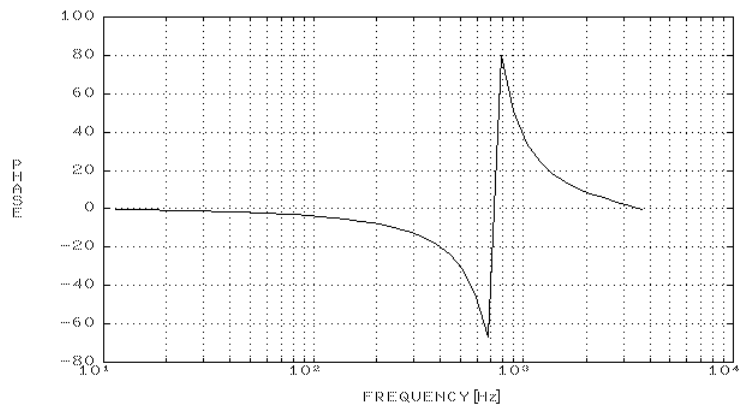


**Note:** The Mx4 default setting for notch filter is no notch (or notch disabled).

## NOTCH cont.



(a)



(b)

Frequency Response of Discrete 750 Hz, Q=2 Notch Filter

## OFFSET

---

**FUNCTION** Amplifier Offset Cancellation

**SYNTAX** `OFFSET n`

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

`n` long value specifying the axis

### DESCRIPTION

This command minimizes the offset generated by the D/A Converter (DAC). Upon completion of offset tuning, an interrupt is generated to the host. The condition is recorded in DPR interrupt status register location 009h. DPR status register location 00Ch will identify the axis responsible. Bit 6 of DPR locations [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also axis bit-coded in bits 0-3 of the DSPL `OFFSET_REG` bit register.



**Note:** `OFFSET` may be run with only one axis at a time. The status of the remaining three axes is not affected by running `OFFSET`.

To run `OFFSET`, the following steps should be followed for the corresponding axis:

1. The axis should be in closed loop with optimal gains set.
2.  $K_i$  must be non zero for the axis.
3. The axis should be 'stopped', with no motion commands in progress.
4. Start `OFFSET` with the specified axis.
5. Offset adjust is complete when a host interrupt is generated.

**SEE ALSO** `CTRL`

## OFFSET cont.

---

### APPLICATION

Most servo amplifiers on the market present an input offset voltage problem that is undesirable for an accurate positioning application. Using `OFFSET`, you may neutralize amplifier offset. To make this happen, you must:

1. enable `OFFSET` for the axis whose offset is to be neutralized, and
2. use a non-zero  $K_i$  gain that maintains stability and zeros position error. (It is assumed that other control gains are selected such that the system is stable.)

Position error is integrated via the integral channel until position error is forced to zero. In the absence of amplifier offset, the DAC voltage that would have achieved zero position error is zero. Any non-zero DAC value is due to an error caused by amplifier offset voltage. `Mx4` measures the voltage, reports satisfactory completion of the `OFFSET` command (generates an interrupt), and uses this measured voltage value to neutralize offset throughout the entire control operation (until machine is turned off). Due to the variable nature of amplifier offset, offset calibration may be necessary any time the machine is turned on.

#### **Command Sequence Example**

```
MAXACC ( ) ;set the maximum accel. so system can be stopped
CTRL ( ) ;set the gains
KILIMIT ( ) ;put system in a position loop, make sure integral
;gain is non-zero
.
.
OFFSET ( )
```

### EXAMPLE

After verifying that `OFFSET` Steps 1-3 (see `DESCRIPTION`, above) have been followed, do offset tuning for axis 3.

```
OFFSET 3
```

## OUTGAIN

---

**FUNCTION**     Output Loop Gain

**SYNTAX**        OUTGAIN   n, m

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n                long value specifying the axis  
m                long value specifying the output gain,

m=0	gain=1
m=1	gain=2
m=2	gain=4
m=3	gain=8
m=4	gain=16

### DESCRIPTION

This command is used to set the gain for the output of the position loops. The default m is set to zero (gain = 1).



**Note:** Please see block diagram with CTRL command.

**SEE ALSO**        CTRL

### APPLICATION

In applications where the number of position encoder counts (per mechanical revolution of the shaft) is low, lack of resolution in the feedback path will manifest itself as a low gain. This may be compensated for by a loop gain adjustment. In practice, this command may use an argument greater than 1 if the encoder line number is less than 1000.

## OUTGAIN cont.

---

### **Command Sequence Example**

```
MAXACC ( )      ;set the maximum accel. so system can be stopped
CTRL ( )        ;set the gains
KILIMIT ( )
OUTGAIN ( )
```

### **EXAMPLE**

Program output loop gains of eight for axis 3 and two for axis 4.

```
BEGIN_RTC
    OUTGAIN 3, 3
    OUTGAIN 4, 1
END_RTC
```

## OUTP\_OFF

---

**FUNCTION** Set Output to 'Off' State

**SYNTAX** OUTP\_OFF outp

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

outp long value specifying the single output to turn 'Off'

[Mx4]	0 <= outp <= 12
[Mx4 Octavia]	0 <= outp <= 31
[IOExp]	0 <= outp <= 63
[IOExp:2]	0 <= outp <= 127

### DESCRIPTION

This command allows the 'Off' status of an [Mx4:13] [Mx4 Octavia:32] output to be set.

**SEE ALSO** OUTP\_ON, POSBRK\_OUT

### APPLICATION

This command can be used for a general purpose logical output operation.

#### **Command Sequence Example**

No preparation is required before running this instruction.

### EXAMPLE

Turn 'Off' the OUT0, OUT5, OUT6, and OUT12 outputs.

```
BEGIN_RTC
    OUTP_OFF 0
    OUTP_OFF 5
    OUTP_OFF 6
    OUTP_OFF 12
END_RTC
```

## OUTP\_ON

---

**FUNCTION** Set Output to 'On' State

**SYNTAX** OUTP\_ON outp

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

outp long value specifying the single output to turn 'On'

[Mx4]	0 <= outp <= 12
[Mx4 Octavia]	0 <= outp <= 31
[IOExp]	0 <= outp <= 63
[IOExp:2]	0 <= outp <= 127

### DESCRIPTION

This command allows the 'On' status of an [Mx4:13] [Mx4 Octavia:32] output to be set.

**SEE ALSO** OUTP\_OFF, POSBRK\_OUT

### APPLICATION

This command can be used for a general purpose logical output operation.

#### **Command Sequence Example**

No preparation is required before running this instruction.

### EXAMPLE

Turn 'On' the OUT5, OUT22, and OUT30 Mx4 Octavia outputs.

```
BEGIN_RTC
    OUTP_OFF 5
    OUTP_OFF 22
    OUTP_OFF 30
END_RTC
```

## OVERRIDE

---

**FUNCTION**     Feedrate override for CIRCLE/LINEAR

**SYNTAX**        OVERRIDE val

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

val            double precision, feedrate override multiplier

$0.1 \leq \text{val} \leq 10$

### DESCRIPTION

This command is used to set the feedrate override for the CIRCLE and LINEAR related commands.

**SEE ALSO**     CIRCLE, LINEAR\_MOVE, LINEAR\_MOVE\_S, LINEAR\_MOVE\_T

### APPLICATION

### EXAMPLES

Set a feedrate override of 4x.

```
OVERRIDE 4.0
```

## PARREAD

---

**FUNCTION**     Parameter Readback

**SYNTAX**        PARREAD m, sbuf

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

m            long value which indicates the parameters to echo.

m=10h    axis 1, 5 position loop gain values [CTRL]

m=11h    axis 2, 6 position loop gain values [CTRL]

m=12h    axis 3, 7 position loop gain values [CTRL]

m=13h    axis 4, 8 position loop gain values [CTRL]

m=14h    Kilimit value [KILIMIT]

m=15h    position loop output gain values [OUTGAIN]

m=16h    maximum acceleration [MAXACC]

m=17h    enabled interrupt

m=18h    mode of operation

m=19h    following error and halt interrupt setpoint [EN\_ERRHLT]

m=1Ah    following error interrupt setpoint [EN\_ERR]

m=1Bh    axis 1, 5 and 2, 6 position breakpoint interrupt setpoint [EN\_POSBRK]

m=1Ch    axis 3, 7 and 4, 8 position breakpoint interrupt setpoint [EN\_POSBRK]

m=1Dh    buffer breakpoint interrupt setpoint and contouring block transfer rate [EN\_BUFBRK, BTRATE, CUBIC\_RATE]

m=1Eh    axis 1, 5 and 2, 6 position breakpoint output mask [POSBRK\_OUT]

m=1Fh    axis 3, 7 and 4, 8 position breakpoint output mask [POSBRK\_OUT]

m=20h    abort maximum acceleration [ESTOP\_ACC]

m=21h    master/slave status

m=22h    output status [OUTP\_ON, OUTP\_OFF]

m=23h    input state

m=24h    encoder fault interrupt setpoint [EN\_ENCFLT]

m=25h    not used

m=26h    acceleration feedforward gain value [CTRL\_KA]

m=27h    torque limit value [TRQ\_LIMIT]

## PARREAD cont.

---

Sbuf     byte array of length 16, used by function to pass data back to the user

### DESCRIPTION

Upon the execution of this command, [Mx4][Mx4 Octavia] echoes the desired parameters to the DPR. The function picks up the [Mx4 : 8] [Mx4 Octavia : 16] bytes and places them in the array Sbuf. The data from the DPR is copied to the array as follows:

Sbuf[0] : 0B8h	Sbuf[8] : 0B8h
Sbuf[1] : 0B9h	Sbuf[9] : 0B8h
Sbuf[2] : 0Bah	Sbuf[10] : 0B8h
Sbuf[3] : 0BBh	Sbuf[11] : 0B8h
Sbuf[4] : 0BCh	Sbuf[12] : 0B8h
Sbuf[5] : 0BDh	Sbuf[13] : 0B8h
Sbuf[6] : 0Beh	Sbuf[14] : 0B8h
Sbuf[7] : 0BFh	Sbuf[15] : 0B8h

### DATA FORMAT

Please, refer to the *Mx4 Octavia User's Guide* or the *Mx4 Users's Guide* for a complete list of the data formats.

**SEE ALSO**     none

### APPLICATION

This command can be used as a diagnostic tool to monitor all system parameters.

## **PARREAD cont.**

---

### **EXAMPLE**

Verify the gains settings for axis 2.

```
Dim Temp[16] As Byte
PARREAD &H11, Temp[0]
```

After the above lines of code are executed, the gains for axis 2 are located in the Temp array as follows:

Temp[0], Temp[1] contain the Ki gain  
Temp[2], Temp[3] contain the Kp gain  
Temp[4], Temp[5] contain the Kf gain  
Temp[6], Temp[7] contain the Kd gain

## **POS**

---

**FUNCTION**     Get Actual Position State Variable

**SYNTAX**        POS   n

**ARGUMENTS**

          n            long value specifying the axis

**DESCRIPTION**

          This function returns a double precision value, the actual position for the axis specified.

**SEE ALSO**        FERR, VEL

**EXAMPLE**

          Read the actual position of axis 5.

```
          Dim Temp As Double
          Temp = POS (5)
```

## POSBRK\_OUT

---

**FUNCTION** Set Outputs After Position Breakpoint Interrupt

**SYNTAX** POSBRK\_OUT n, outpon, outpoff

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis  
 outpon long value, bit coding the outputs to turn 'on' upon occurrence of position breakpoint interrupt (EN\_POSBRK) for specified axis.

if bit=0 no change in output status  
 if bit=1 output = LOW TTL voltage

bit 0 OUT0 output  
 bit 1 OUT1 output  
 bit 2 OUT2 output  
 bit 3 OUT3 output  
 bit 4 OUT4 output  
 bit 5 OUT5 output  
 bit 6 OUT6 output  
 bit 7 OUT7 output  
 bit 8 OUT8 output  
 bit 9 OUT9 output  
 bit 10 OUT10 output  
 bit 11 OUT11 output  
 bit 12 OUT12 output  
 bit 13 OUT13 output  
 bit 14 OUT14 output  
 bit 15 OUT15 output  
 bit 16 OUT16 output  
 bit 17 OUT17 output  
 bit 18 OUT18 output  
 bit 19 OUT19 output  
 bit 20 OUT20 output

## **POSBRK\_OUT cont.**

---

bit 21	OUT21 output
bit 22	OUT22 output
bit 23	OUT23 output
bit 24	OUT24 output
bit 25	OUT25 output
bit 26	OUT26 output
bit 27	OUT27 output
bit 28	OUT28 output
bit 29	OUT29 output
bit 30	OUT30 output
bit 31	OUT31 output
outpoff	long value, bit coding the outputs to turn 'off' upon occurrence of position breakpoint interrupt (EN_POBPRK) for specified axis.
if bit=0	no change in output status
if bit=1	output = HIGH TTL voltage
bit 0	OUT0 output
bit 1	OUT1 output
bit 2	OUT2 output
bit 3	OUT3 output
bit 4	OUT4 output
bit 5	OUT5 output
bit 6	OUT6 output
bit 7	OUT7 output
bit 8	OUT8 output
bit 9	OUT9 output
bit 10	OUT10 output
bit 11	OUT11 output
bit 12	OUT12 output
bit 13	OUT13 output
bit 14	OUT14 output
bit 15	OUT15 output
bit 16	OUT16 output
bit 17	OUT17 output
bit 18	OUT18 output

## POSBRK\_OUT cont.

---

bit 19	OUT19 output
bit 20	OUT20 output
bit 21	OUT21 output
bit 22	OUT22 output
bit 23	OUT23 output
bit 24	OUT24 output
bit 25	OUT25 output
bit 26	OUT26 output
bit 27	OUT27 output
bit 28	OUT28 output
bit 29	OUT29 output
bit 30	OUT30 output
bit 31	OUT31 output

### DESCRIPTION

This command enables the output status of selected outputs to be activated by the occurrence of a position breakpoint interrupt (EN\_POSBRK) for a specified axis. The POSBRK\_OUT need only be executed once (ie: during initialization) unless the on/off output status desired changes. The specified outputs will change state as programmed through the outpon and outpoff arguments when the specified axis generates a position breakpoint interrupt. The position breakpoint interrupt (EN\_POSBRK) must be enabled for the output status changes to occur.

**SEE ALSO** EN\_POSBRK, OUTP\_OFF, OUTP\_ON

### APPLICATION

This command can be used for an output operation where the output status must be tightly coupled to the position of one or more axes.

#### **Command Sequence Example**

```
EN_POSBRK      ;enable the pos breakpoint int for specified axis(es)
POSBRK_OUT    ;set the desired output status changes
```

## **POSBRK\_OUT cont.**

---

### **EXAMPLE**

If a position breakpoint interrupt occurs on axis 1, turn on OUT0-OUT3 and turn off OUT4.

```
POSBRK_OUT 1, &H0000000F, &H00000010
```

## POSITION\_UNIT

---

**FUNCTION** User-Specified Position Unit

**SYNTAX** POSITION\_UNIT val

### ARGUMENTS

val double precision, position unit specified in multiples of 1 count

### DESCRIPTION

This function allows a user-specified position unit to be programmed. The default unit of time is one count. The position unit affects the interpretation of position, velocity, and acceleration arguments in subsequent calls to the DLL.

**SEE ALSO** TIME\_UNIT

### APPLICATION

The POSITION\_UNIT and TIME\_UNIT functions allow the application programmer to use whatever units are natural for the application.

### EXAMPLE

Program the position and time unit so that the position, velocity, and acceleration arguments use the units revolutions, revolutions/msec, and revolutions/msec<sup>2</sup>. There are 4096 counts/revolution.

```
POSITION_UNIT 4096
TIME_UNIT 1# / 1000#
```

## POS\_PRESET

---

**FUNCTION** Preset Position Counter

**SYNTAX** POS\_PRESET n, pset

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis  
pset double precision, position counter preset value

$-2147483648 \leq \text{pset} \leq 2147483647$  counts

### DESCRIPTION

This command will define the present position point for the axes specified.



**Note:** POS\_PRESET will automatically disable the position breakpoint interrupt (if enabled). POS\_PRESET should be executed only when the axes specified are not in motion.

**SEE ALSO** POS\_SHIFT, EN\_POSBRK

### APPLICATION

This command is useful when the position counter must be forced to a new value. POS\_PRESET may be used in the establishment of a new reference position.

### EXAMPLE

Preset the axis 1 and axis 8 positions to 20000 and -45999 counts, respectively.

```
BEGIN_RTC
  POS_PRESET 1, 20000
  POS_PRESET 8, -45999
END_RTC
```

## POS\_SHIFT

---

**FUNCTION** Position Reference Shift

**SYNTAX** POS\_SHIFT n, psft

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n	long value specifying the axis
psft	double precision, position reference value

$-2147483648 \leq \text{psft} \leq 2147483647$

### DESCRIPTION

This command will shift the present position point for the axes specified.



**Note:** POS\_SHIFT will automatically disable the position breakpoint interrupt (if enabled) of the specified axes.

**SEE ALSO** POS\_PRESET, EN\_POSBRK

### APPLICATION

This command may be used in homing a linear system based on index pulse position recording. Adding offset position (in encoder edge counts) to an already recorded position, presets position to a new value without losing position integrity (i.e., no counter information is lost).

### EXAMPLE

The current axis one position is 45000 counts. Shift the axis 1 position to 50000 counts. The current axis 3 position is 55000 counts. Shift the axis 3 position to 50000 counts.

```
BEGIN_RTC
    POS_SHIFT 1, 5000
    POS_SHIFT 3, -5000
END_RTC
```

## PWM\_FREQ

Vx4++ option command

**FUNCTION** Set Pulse Width Modulation (PWM) Frequency

**SYNTAX** PWM\_FREQ m, pwmfreq

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

m long value specifying the axis group

m = 1 set axes one, two PWM frequency

m = 2 set axes three, four PWM frequency

pwmfreq single precision PWM frequency

$1.0 \leq \text{pwmfreq} \leq 31.0 \text{ kHz}$

### DESCRIPTION

The frequency of the Vx4++ pulse width modulation outputs may be programmed via the PWM\_FREQ command. The outputs may be programmed in axis pairs.



**Note:** Mx4 with Vx4++ will not execute the PWM\_FREQ command if the Vx4\_BLOCK command is active for the axes in question.

**SEE ALSO** Vx4\_BLOCK

### APPLICATION

See *Vx4++ User's Guide*

### EXAMPLE

Set a PWM frequency of 15.4 kHz for axes three and four.

```
PWM_FREQ 2, 15.4
```

## REL\_AXMOVE

---

**FUNCTION** Relative Position Axis Move with Trapezoidal Trajectory

**SYNTAX** REL\_AXMOVE n, acc, pos, vel

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n	long value specifying the axis
acc	double precision unsigned value specifying the maximum halting acceleration (deceleration)
	$0 \leq \text{acc} \leq 1.999969 \text{ counts}/(200\mu\text{s})^2$
pos	double precision incremental position
	$-805306367 \leq \text{pos} \leq 805306367 \text{ counts}$
vel	double precision unsigned target velocity
	$0 \leq \text{vel} \leq 255.99998 \text{ counts}/200\mu\text{s}$

### DESCRIPTION

The REL\_AXMOVE command is similar to the AXMOVE command with the exception that relative (or incremental) position is specified, rather than an end position as with AXMOVE.

**SEE ALSO** AXMOVE, AXMOVE\_S, AXMOVE\_T, REL\_AXMOVE\_S, REL\_AXMOVE\_T, STOP

### EXAMPLE

The current position (commanded) of axis 2 is unknown. It is known, however, that we want to move axis 2 8000 counts in the negative direction (that is, -8000 counts from the current position). The move should be accomplished with an acceleration of  $1.0 \text{ counts}/(200\mu\text{s})^2$  and a target slew rate of  $-3.5 \text{ counts}/200\mu\text{s}$ .

```
REL_AXMOVE 2, 1.0, -8000, 3.5
```

## REL\_AXMOVE\_S

---

**FUNCTION** Relative S-Curve Axis Move with Trapezoidal Trajectory

**SYNTAX** REL\_AXMOVE\_S n, acc, pos, vel

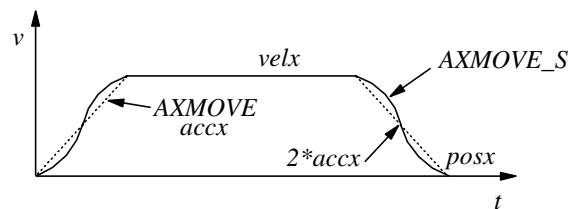
If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n	long value specifying the axis
acc	double precision unsigned value specifying the acceleration/deceleration
	$0 \leq \text{acc} \leq 1.999969 \text{ counts}/(200\mu\text{s})^2$
pos	double precision relative position
	$-2147483648 \leq \text{pos} \leq 2147483647 \text{ counts}$
vel	double precision unsigned target velocity
	$0 \leq \text{vel} \leq 255.99998 \text{ counts}/200\mu\text{s}$

### DESCRIPTION

The REL\_AXMOVE\_S RTC allows for s-curve command generation with relative (to current position) endpoint position, slew rate velocity, and acceleration for each axis. This command is suitable for linear moves where s-curve acceleration is desired.

**REL\_AXMOVE\_S cont.**

The figure above illustrates the velocity profile of the REL\_AXMOVE\_S along with the linear velocity ramp of the REL\_AXMOVE command. With REL\_AXMOVE\_S, the acceleration will reach a value of  $2*accx$  for a maximum (see above figure).

**SEE ALSO** AXMOVE, AXMOVE\_S, AXMOVE\_T, REL\_AXMOVE, REL\_AXMOVE\_T, STOP

**EXAMPLE**

The current position (commanded) of axis 2 is unknown. It is known, however, that we want to move axis 2 8000 counts in the negative direction (that is, -8000 counts from the current position). The move should be accomplished with an acceleration of  $1.0 \text{ counts}/(200\mu\text{s})^2$  and a target velocity of (unsigned)  $3.5 \text{ counts}/200\mu\text{s}$ .

```
REL_AXMOVE_S 2, 1.0, -8000, 3.5
```

## REL\_AXMOVE\_T

---

**FUNCTION** Time-Based Relative Axis Move with Trapezoidal Trajectory

**SYNTAX** REL\_AXMOVE\_T n, acc, pos, tm

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n	long value specifying the axis
acc	double precision unsigned value specifying the acceleration/deceleration $0 \leq acc \leq 1.999969 \text{ counts}/(200\mu\text{s})^2$
pos	double precision relative position $-2147483648 \leq pos \leq 2147483647 \text{ counts}$
tm	double precision motion time $0 \leq tm \leq 5000000 (200\mu\text{s})$



**Note:** The time argument, tm, is an unsigned value with a unit of 200μsec.

### DESCRIPTION

The REL\_AXMOVE\_T RTC allows for trapezoidal command generation with relative (to current position) endpoint position, acceleration, and time to complete the move for each axis. This

## **REL\_AXMOVE\_T cont.**

---

command is suitable for linear moves where relative endpoint position and motion time are the specifying parameters.

The REL\_AXMOVE\_T command is similar to REL\_AXMOVE, with the exception that the velocity argument is replaced with a time argument. REL\_AXMOVE\_T will automatically calculate a suitable slew rate velocity to achieve the programmed relative endpoint position in the programmed amount of time, following a trapezoidal velocity profile (similar to REL\_AXMOVE).

**SEE ALSO**      REL\_AXMOVE,      REL\_AXMOVE\_S,      AXMOVE,      AXMOVE\_S,  
                  AXMOVE\_T, STOP

### **EXAMPLE**

The current position (commanded) of axis 4 is unknown. It is known, however, that we want to move axis 4 10000 counts in the negative direction (that is, -10000 counts from the current position). The move should be accomplished with an acceleration of  $1.0 \text{ counts}/(200\mu\text{s})^2$  and be completed in 350msec ( $1750 * 200\mu\text{sec}$ ).

```
REL_AXMOVE_T 4, 1.0, -10000, 1750
```

## **REL\_AXMOVE\_SLAVE**

---

**FUNCTION** Superimposes a Relative Axis Move onto a Slave Engaged in Gearing

**SYNTAX** REL\_AXMOVE\_SLAVE n, acc, rel\_pos, rel\_vel

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

n	long value specifying the axis
acc	double precision relative move acceleration
rel_pos	double precision position value relative to current position
rel_vel	double precision velocity value relative to current velocity

### **DESCRIPTION**

This command is similar to AXMOVE with two exceptions. First, it is relative, not absolute; and second, it works only on the slave axis(es) involved in electronically geared or cam applications. This command allows the slave to momentarily disengage from the gearing process and compensate for its positional shortcomings.

**SEE ALSO** CAM, CAM\_OFF, CAM\_OFF\_ACC, CAM\_POS, CAM\_PROBE, GEAR, GEAR\_OFF, GEAR\_OFF\_ACC, GEAR\_POS, GEAR\_PROBE

### **APPLICATION**

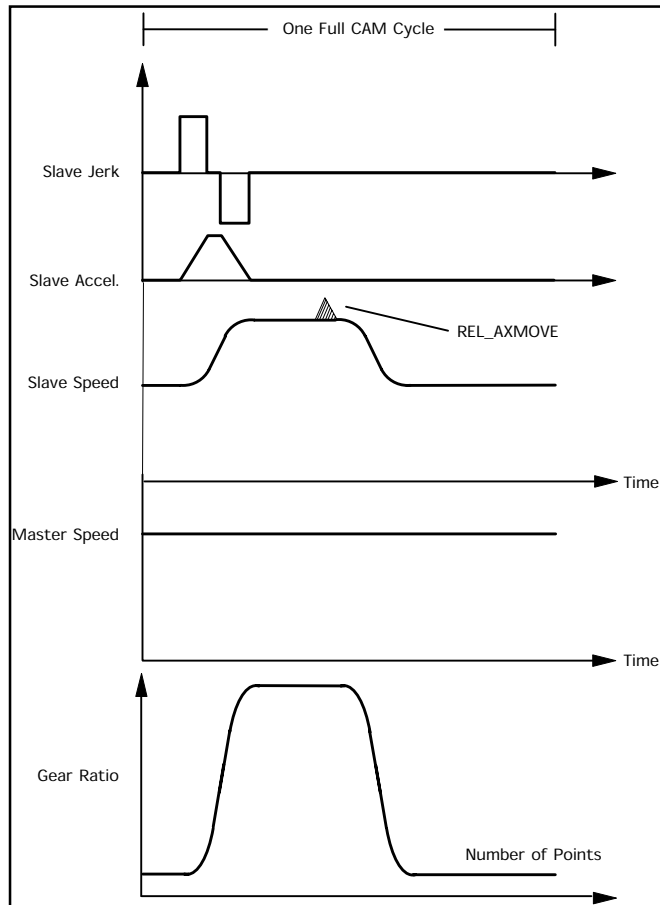
General master/slaving, in particular flying shear applications, can benefit from this instruction. Flying shear with registration marks is handled similarly to that of synchronous cutting. That is, the measured cutting error is used in the next cycle as an added function to compensate for the motion's shortcomings.

## REL\_AXMOVE\_SLAVE cont.

### EXAMPLE

Axis 7 is a slave axis engaged in GEAR with the master axis. Add a trapezoidal profile “on top” of the gearing which adjusts the slave +1000 counts.

```
REL_AXMOVE_SLAVE 7, 1.0, 1000, 1
```



## RESET\_MX4

---

**FUNCTION**     Reset Mx4

**SYNTAX**       RESET\_MX4

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

none

### DESCRIPTION

This command brings the servo controller card back to power-up state. Upon Mx4's reset completion, a host interrupt is generated via bit 4 of DPR locations [Mx4:7FEh] [Mx4 Octavia:1FFEh].

**SEE ALSO**     none

### APPLICATION

From time to time all systems may have to be software reset to allow for an initialization.

#### ***Command Sequence Example***

No preparation is required before running this instruction.

### EXAMPLE

Reset the Mx4 controller card.

```
RESET_MX4
```

## **RESETCOMMUNICATIONS**

---

**FUNCTION**     Reset Serial Communication

**SYNTAX**       RESETCOMMUNICATIONS ( )

**ARGUMENTS**

                  none

**DESCRIPTION**

                  This function resets the host – Mx4 serial communication. It returns (long value) a 1 if successful, 0 otherwise.

**SEE ALSO**     CHANGECOMMPORTSETTING,            CHANGESLAVENODEADDRESS,  
                  COMMUNICATIONSLOST,    GETCOMMINSTCOUNT,    GETCOMMTYPE,  
                  GETCURRENTNODEADDRESS

**EXAMPLE**

                  Reset serial communications, monitor the result.

```
                  Dim Temp As Long  
                  Temp = RESETCOMMUNICATIONS ( )
```

## R\_1BYTE

---

**FUNCTION** Read Single Byte From Dual Port RAM

**SYNTAX** R\_1BYTE (offset)

**ARGUMENTS**

Offset      long value, offset into DPR from base address

**DESCRIPTION**

This function reads a single byte from the Mx4 Dual Port RAM from the address (DPR base + offset). The function returns a byte value.

**SEE ALSO** R\_2BYTE, R\_4BYTE, W\_1BYTE, W\_2BYTE, W\_4BYTE

**EXAMPLE**

Read Mx4 DPR address 3C2h (RTC command code location).

```
Dim Temp As Byte
Temp = R_1BYTE (&H3C2)
```

## R\_2BYTE

---

**FUNCTION** Read Two Bytes From Dual Port RAM

**SYNTAX** R\_2BYTE (offset)

**ARGUMENTS**

Offset      long value, offset into DPR from base address

**DESCRIPTION**

This function reads two bytes from the Mx4 Dual Port RAM, starting from the address (DPR base + offset). The function returns a long value.

**SEE ALSO** R\_1BYTE, R\_4BYTE, W\_1BYTE, W\_2BYTE, W\_4BYTE

**EXAMPLE**

Read Mx4 DPR addresses 3C3h, 3C4h (RTC argument locations).

```
Dim Temp As Long
Temp = R_2BYTE (&H3C3)
```

## R\_4BYTE

---

**FUNCTION** Read Four Bytes From Dual Port RAM

**SYNTAX** R\_4BYTE (offset)

**ARGUMENTS**

Offset long value, offset into DPR from base address

**DESCRIPTION**

This function reads four bytes from the Mx4 Dual Port RAM, starting from the address (DPR base + offset). The function returns a long value.

**SEE ALSO** R\_1BYTE, R\_2BYTE, W\_1BYTE, W\_2BYTE, W\_4BYTE

**EXAMPLE**

Read Mx4 DPR addresses 3C3h, 3C4h, 3C5h, 3C6h (RTC argument locations).

```
Dim Temp As Long
Temp = R_4BYTE (&H3C3)
```

## R\_NBYTE

---

**FUNCTION** Reads n Bytes From Dual Port RAM

**SYNTAX** R\_NBYTE (offset, number, bytearray)

**ARGUMENTS**

offset	long value, offset into DPR from base address
number	long value, number of bytes to read
bytearray	byte array, pass by reference, return array of bytes read

**DESCRIPTION**

This function reads four bytes from the Mx4 Dual Port RAM, starting from the address (DPR base + offset). The function takes an array as an argument for the return values.

**SEE ALSO** R\_1BYTE, R\_2BYTE, R\_NBYTE, W\_1BYTE, W\_2BYTE, W\_4BYTE, W\_NBYTE

**EXAMPLE**

Read Mx4 DPR addresses 3C2h – 3CCh.

```
Dim ByteArray(10) As Byte
R_NBYTE (&H3C3, 10, ByteArray)
```

## **SIGNAL\_DSPL**

---

**FUNCTION** Send A Real-Time ‘Signal’ to the DSPL Program

**SYNTAX** SIGNAL\_DSPL

If used as a function, the function will return (long) zero if successful, nonzero if error.

### **ARGUMENTS**

none

### **DESCRIPTION**

This function sends a real-time software ‘signal’ to the running DSPL program. In order for the signal to be received, the DSPL program must be waiting at a `WAIT_UNTIL_RTC` command while the `SIGNAL_DSPL` command is executed. The `SIGNAL_DSPL` – `WAIT_UNTIL_RTC` pair is used for timing or synchronization purposes between a DSPL program and the host computer.

**SEE ALSO** `WAIT_UNTIL_RTC` (*DSPL Programmer’s Guide*)

### **APPLICATION**

See *DSPL Programmer’s Guide*.

### **EXAMPLE**

Send a signal to the waiting DSPL program.

```
SIGNAL_DSPL
```

## SIGNATURE

---

**FUNCTION** Read Mx4 Controller Signature

**SYNTAX** SIGNATURE sbuf

### ARGUMENTS

Sbuf string, used to write the signature in

### DESCRIPTION

Each Mx4 controller has an 11-byte signature which identifies the controller and its firmware versions. This command requires a string as an argument, and returns the string with format as follows:

Byte 1 ASCII "M"  
Byte 2 ASCII "X"  
Byte 3 ASCII "4"  
Byte 4 integer portion of DSP1 firmware version  
Byte 5 fraction portion of DSP1 firmware version  
Byte 6 ASCII "+"  
Byte 7 integer portion of DSP2 firmware version  
Byte 8 fraction portion of DSP2 firmware version  
Byte 9 ASCII "+"  
Byte 10 integer portion of Vx4++ firmware version (if present)  
Byte 11 integer portion of Vx4++ firmware version (if present)

**SEE ALSO** none

### APPLICATION

This function can be used to test for the presence of Mx4 in a system.

### EXAMPLE

Test for presence of Mx4.

```
Dim sBuffer As String
SBuffer = Space(11)
If Left$(SIGNATURE(sBuffer), 3) <> "MX4" Then
    MsgBox "Mx4 Not Found"
End
End If
```

## START

---

**FUNCTION** Start Contouring Motion

**SYNTAX** START n

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis

### DESCRIPTION

This command starts the motion (simultaneously) for the specified axes included in 2nd order and cubic spline contouring. *START* applies to contouring only.



**Note:** *START* will be ignored if contouring is in progress.

**SEE ALSO** STOP, VECCHG

### APPLICATION

This command must be used in all 2nd order and ring buffer cubic spline contouring applications to start contouring with selected axes.

#### For 2nd Order Contouring Only

This command can be overwritten by *VECCHG* which redefines the axes involved in the contouring process. For example, *START* starts the contouring of axes 1, 3, and 4. If in the course of contouring, a *VECCHG* is received (with argument) specifying axes 1, 2, and 3, the new contouring points in the ring buffer will be used for the newly defined axes. Please also see *VECCHG*.

## START cont.

---

### Command Sequence Example

```
. ;load ring buffer with positions and velocities
.  
MAXACC ( ) ;make sure system can stop  
CTRL ( ) ;set the gains  
KILIMIT ( )  
BTRATE ( ) ;set the block transfer rate  
EN_BUFBRK ( ) ;set the breakpoint in the ring buffer  
  
.  
.  
START ( ) ;start contouring
```

### EXAMPLE

Start contouring motion in axes 2, 3 and 4.

```
BEGIN_RTC  
    START 2  
    START 3  
    START 4  
END_RTC
```

## START\_DSPL

---

**FUNCTION**     Initiate DSPL Program Code Execution

**SYNTAX**        `START_DSPL`

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

none

### DESCRIPTION

This function initiates the execution of the previously downloaded DSPL program.

**SEE ALSO**     `AUTOSTART_DSPL`, `CLEAR_DSPL`, `DOWNLOAD_DSPL`, `MAXACC`,  
`SIGNAL_DSPL`, `STOP_DSPL`

### APPLICATION

See *DSPL Programmer's Guide*.

### EXAMPLE

Initiate DSPL program execution.

```
START_DSPL
```

## STEPPER\_ON

Stp4 option command

**FUNCTION** Select Servo/Stepper Axes

**SYNTAX** STEPPER\_ON n, m

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n	long value specifying the axis
m	long value indicating stepper or servo for axis
m=0	servo axis
m=1	stepper axis

### DESCRIPTION

This command requires the Stp4 add-on card. STEPPER\_ON allows the user to select the axes which are stepper control axes. Note that at power-up / reset, all Mx4 axes are configured as servo axes.

### EXAMPLE

Select axes 1 and 2 as stepper control axes.

```
BEGIN_RTC
  STEPPER_ON 1
  STEPPER_ON 2
END_RTC
```

## STOP\_AXIS

---

**FUNCTION** Stop Motion

**SYNTAX** STOP\_AXIS n

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis

### DESCRIPTION

This command stops the motion of all specified axes simultaneously. To stop motion, the servo control card uses the programmed values for maximum acceleration / deceleration. Upon receipt of STOP, the servo controller aborts the current command. The host is responsible for clearing the ring buffer of any remaining commands if the axis(es) stopped was involved in contouring motion.



**Note 1:** An emergency stop signal, ESTOP\_ACC, will perform a hardware stop. This is an open collector input signal which is active low and is shared between all of the controller cards.



**Note 2:** STOP will be ignored if the maximum acceleration / deceleration is equal to zero (e.g., MAXACC not issued).

If an axis is halting to a stop from a previously executed STOP RTC or active ESTOP\_ACC input, Mx4 will ignore any motion commands (AXMOVE, REL\_AXMOVE, START or VELMODE) and will report an "RTC Command Ignored" interrupt to the host. The above motion commands should not be sent to Mx4 for a halting axis until the axis motion has come to a stop.

**SEE ALSO** MAXACC, START

## STOP cont.

---

### APPLICATION

For all applications involving bringing speed to zero in the quickest possible manner.

#### **Command Sequence Example**

```

MAXACC ( )      ;set the maximum accel. so system can be stopped
CTRL ( )        ;set the gains
KILIMIT ( )
BTRATE ( )      ;set the block transfer rate
EN_BUFBRK ( )   ;set the breakpoint in the ring buffer
.
.
STOP ( )         ;stop the motion
.                ;upon completion of stop (command) trajectory
.                ;Mx4 generates motion complete interrupt

```

### EXAMPLE

Bring the motion of axes 1 and 6 to a halt.

```

BEGIN_RTC
    STOP_AXIS  1
    STOP_AXIS  6
END_RTC

```

## STOP\_DSPL

---

**FUNCTION** Terminate DSPL Program Code Execution

**SYNTAX** STOP\_DSPL

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

none

### DESCRIPTION

This function terminates the execution of the DSPL program. This command will also halt the motion (if any) of all axes with the programmed MAXACC acceleration.

**SEE ALSO** AUTOSTART\_DSPL, CLEAR\_DSPL, DOWNLOAD\_DSPL, MAXACC, SIGNAL\_DSPL, START\_DSPL

### APPLICATION

See *DSPL Programmer's Guide*.

### EXAMPLE

Terminate DSPL program execution.

```
STOP_DSPL
```

## SYNC

---

**FUNCTION** Master / Slave Select

**SYNTAX** SYNC m

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

m long value selects the Master / Slave status of the Mx4 card

m=0 Mx4 is configured as a Master

m=1 Mx4 is configured as a Slave

### DESCRIPTION

If more than one Mx4 card is to be used in a system and card-to-card synchronization is required, the SYNC command should be used. SYNC allows multiple Mx4 cards to operate in synchronization within a system by specifying a single Master and the remaining card(s) as Slave(s). If only one Mx4 is used in a host computer system, that Mx4 must be configured as a Master.



**Note:** Mx4 powers-up and resets to a default Master status.

In addition to configuring the Mx4 cards with SYNC (for multiple card systems), a cable jumper must be included on the J5 connector of each of the boards. The cable must be wired such that the MASTER signal from the Master Mx4 connects to the SLAVE signal of each of the Slave Mx4(s) (see *Mx4 User's Guide, Installing Your Mx4*).

**SEE ALSO** none

## **SYNC cont.**

---

### **APPLICATION**

This command is used in applications where tight coordination of more than four axes is required. This command essentially slaves several Mx4 cards to a single Master Mx4. Applications involving many axes contouring may benefit from this command.

#### ***Command Sequence Example***

This command must be executed immediately after the initialization. Please remember that the default value for m is zero (i.e., the card is initialized as a Master).

### **EXAMPLE**

Configure the Mx4 controller as a slave in a multi-Mx4 synchronized system.

SYNC 1

## TABLE\_SEL

---

**FUNCTION**     Select Compensation Table

**SYNTAX**       TABLE\_SEL n, tb

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n               long value specifying the axis

tb              long value specifies the compensation table to be used

$1 \leq tb \leq 8$

### DESCRIPTION

The TABLE\_SEL command allows the user to arbitrarily select the compensation table for the axis(es) in question. More than one axis may use a compensation table.

**SEE ALSO**       CIRCLE,       CLEAR\_POS\_TABLE,       CLEAR\_VEL\_TABLE,  
                  DOWN\_POS,   DOWN\_VEL

### EXAMPLE

Axes 1 and 2 are to use compensation table 2, while axes 3 and 7 use compensation table 1.

```
BEGIN_RTC
  TABLE_SEL 1, 2
  TABLE_SEL 2, 2
  TABLE_SEL 3, 1
  TABLE_SEL 7, 1
END_RTC
```

## TIME\_UNIT

---

**FUNCTION** User-Specified Time Unit

**SYNTAX** TIME\_UNIT val

### ARGUMENTS

val double precision, time unit specified in multiples of 1 second

### DESCRIPTION

This function allows a user-specified time unit to be programmed. The default unit of time is one second. The time unit affects the interpretation of velocity and acceleration arguments in subsequent calls to the DLL.

**SEE ALSO** POSITION\_UNIT

### APPLICATION

The POSITION\_UNIT and TIME\_UNIT functions allow the application programmer to use whatever units are natural for the application.

### EXAMPLE

Program the time unit so that velocity and acceleration arguments use the units counts/msec and counts/msec<sup>2</sup>.

```
TIME_UNIT 1# / 1000#
```

## TRQ\_LIMIT

---

**FUNCTION** DAC Output Voltage Limit

**SYNTAX** TRQ\_LIMIT n, val

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n long value specifying the axis  
val single precision, DAC output voltage (abs) limit

-10.0 <= val <= 9.9997 volts

### DESCRIPTION

The TRQ\_LIMIT command specifies a torque limit (by means of output voltage limiting) value ranging from 0 volts (no output) to +/-10 volts (full swing) with a resolution of approximately 0.3 millivolts.

The Mx4 controller powers-up and resets to a default torque limit value allowing full output voltage swing.

**SEE ALSO** none

### APPLICATION

This command can be used in applications where an axis torque needs to be limited, such as packaging or material handling.

#### **Command Sequence Example**

No preparation is required before running this instruction.

### EXAMPLE

Limit the output voltage swing for axis 2 to +/- 7.5 volts.

```
TRQ_LIMIT 2, 7.5
```

## VAR

---

**FUNCTION**     Get DSPL Variable

**SYNTAX**       VAR m

### ARGUMENTS

m               long value specifying the monitored var to read  
  
                  1 <= m <= 4

### DESCRIPTION

This function returns a double precision value, the selected monitored DSPL variable. Remember that Mx4 allows a maximum of four DSPL variables to be monitored from the DPR at the same time. The selection of which of the 128 DSPL variables are reported to the DPR window is made via the `MONITOR_VAR` command.

**SEE ALSO**       CHANGE\_VAR, MONITOR\_VAR

### EXAMPLE

Read the DSPL variables VAR12, VAR22, VAR44, and VAR59.

```
Dim Temp1, Temp2, Temp3, Temp4 As Double
MONITOR_VAR 1, 12
MONITOR_VAR 2, 22
MONITOR_VAR 3, 44
MONITOR_VAR 4, 59
Temp1 = VAR (1)
Temp2 = VAR (2)
Temp3 = VAR (3)
Temp4 = VAR (4)
```

## **VEC**

---

**FUNCTION**     Get Vx4++ Variable

**SYNTAX**       VEC n

**ARGUMENTS**

n               long value specifying the axis

**DESCRIPTION**

This function returns a double precision value, the selected Vx4++ state variable (VIEWVEC) for the axis specified.

**SEE ALSO**     VIEWVEC

**EXAMPLE**

Read the Vx4++ state variable of axis 3.

```
Dim Temp As Double
Temp = VEC (3)
```

## VECCHG

---

**FUNCTION** 2nd Order Contouring Vector Change

**SYNTAX** VECCHG n, m

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n	long value bit coding the axis(es)
m	long value which represents the buffer position (in 8 byte offsets from the start of the buffer) where the number of axes involved in contouring must be changed to include only those axes coded by n

### DESCRIPTION

Upon the execution of this command, the 2nd order contouring task assumes a new set of axes at the programmed pointer location.



**Note:** Three buffer levels are used to implement this instruction.

**SEE ALSO** START

### APPLICATION

See START.

## VECCHG cont.

---

### **Command Sequence Example**

```
MAXACC ( )      ;set the maximum accel. so system can be stopped
CTRL ( )        ;set the gains
KILIMIT ( )
BTRATE ( )      ;set the block transfer rate
EN_BUFBRK ( )   ;set the buffer breakpoint interrupt
.
.
START ( )       ;start contouring for a selected number of axes
.               ;based on buffer breakpoint interrupt transfer more
.               ;points
VECCHG ( )      ;use points in ring buffer for a new set of axes
```

### **EXAMPLE**

Begin 2nd order contouring in axes 1, 2, and 3 after the 23rd segment move command of the ring buffer.

```
VECCHG  &H7, 23
```

## **VEL**

---

**FUNCTION**     Get Actual Velocity State Variable

**SYNTAX**        VEL   n

### **ARGUMENTS**

          n            long value specifying the axis

### **DESCRIPTION**

This function returns a double precision value, the actual velocity for the axis specified.

**SEE ALSO**        FERR, POS

### **EXAMPLE**

Read the actual velocity of axis 8.

```
Dim Temp As Double
Temp = VEL (8)
```

## VELMODE

---

**FUNCTION**    Velocity Mode

**SYNTAX**        VELMODE n, vel

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

n                long value specifying the axis  
 vel              double precision target velocity

$-256 \leq \text{vel} \leq 255.99998 \text{ counts}/200\mu\text{s}$

### DESCRIPTION

Upon the execution of this command a velocity loop for the specified axes will be closed. The velocity loop uses the same gains as those specified using the control law command. VELMODE uses the MAXACC maximum acceleration / deceleration value to accelerate or decelerate to the desired velocity.



**Note:** VELMODE will be ignored if the maximum acceleration / deceleration is equal to zero (e.g., MAXACC not issued).

**SEE ALSO**        MAXACC

### APPLICATION

This instruction is useful in all general purpose velocity control applications. Please remember that although VELMODE primarily regulates speed, the outer loop is still position. This means that while regulating speed, Mx4 continually tries to zero the position error.

#### **Command Sequence Example**

```
MAXACC ( )        ;set the maximum accel. so system can be stopped
CTRL ( )         ;set the gains
KILIMIT ( )
.
VELMODE ( )
```

## **VELMODE cont.**

---

### **EXAMPLE**

Engage axis 2 in velocity mode with a velocity of 3.71 counts/200  $\mu$ s.

```
VELMODE 2, 3.71
```

**VIEWVEC**

Vx4++ option command

**FUNCTION** Specify Vx4++ State Variables to View**SYNTAX** VIEWVEC n, m

If used as a function, the function will return (long) zero if successful, nonzero if error.

**ARGUMENTS**

n long value specifying the axis  
 m long value specifying state variable

m=0	Iqs error
m=1	Ids error
m=2	Iqs feedback
m=3	Ids feedback
m=4	Iqs command
m=5	Ir feedback
m=6	Is feedback
m=7	It feedback

**DESCRIPTION**

This command selects the Vx4++ state variable which is available in the Mx4 Dual Port RAM and also with the VECT4\_PARx DSPL identifiers. As is evident above, only 1 variable may be “viewed” per axis at any given time.

**SEE ALSO** VEC**APPLICATION**

See *Vx4++ User's Guide*

**EXAMPLE**

Change the Vx4++ state variable selection to Ids feedback for axis 1. Any subsequent VECT4\_PAR1 accesses will yield the axis 1 Ids feedback value.

```
VIEWVEC 1, 3
```

## VX4\_BLOCK

VX4++ option command

**FUNCTION** Blocks Vx4++ commands

**SYNTAX** VX4\_BLOCK m, blk

If used as a function, the function will return (long) zero if successful, nonzero if error.

### ARGUMENTS

m long value specifying the axis groups

m = 1 axes one, two

m = 2 axes three, four

blk long value block code

blk = 0 Vx4++ block disabled

blk = 1 Vx4++ block enabled

### DESCRIPTION

This command is used to block some of the vx4++ commands so that those commands may not be accidentally executed. The user is responsible to disable the block command in order to execute one of the commands listed below (SEE ALSO).

**SEE ALSO** CURR\_LIMIT, CURR\_OFFSET, ENCOD\_MAG,  
MOTOR\_TECH, PWM\_FREQ

### APPLICATION

See *Vx4++ User's Guide*

### EXAMPLE

Enable the vx4++ command blocking for axes 3 and 4.

```
VX4_BLOCK 2, 1
```

## W\_1BYTE

---

**FUNCTION** Write Single Byte To Dual Port RAM

**SYNTAX** W\_1BYTE (offset, value)

**ARGUMENTS**

Offset	long value, offset into DPR from base address
Value	byte value, byte to write to DPR

**DESCRIPTION**

This function writes a single byte to the Mx4 Dual Port RAM at the address (DPR base + offset).

**SEE ALSO** R\_1BYTE, R\_2BYTE, R\_4BYTE, W\_2BYTE, W\_4BYTE

**EXAMPLE**

Write B6h to Mx4 DPR address 3C2h (RTC command code location).

```
W_1BYTE (&H3C2, &HB6)
```

## W\_2BYTE

---

**FUNCTION** Write Two Bytes To Dual Port RAM

**SYNTAX** `W_2BYTE (offset, value)`

**ARGUMENTS**

Offset	long value, offset into DPR from base address
Value	long value, bytes to write to DPR

**DESCRIPTION**

This function writes two bytes to the Mx4 Dual Port RAM starting at the address (DPR base + offset).

**SEE ALSO** `R_1BYTE, R_2BYTE, R_4BYTE, W_1BYTE, W_4BYTE`

**EXAMPLE**

Write 48B3h to Mx4 DPR address 3C3h, 3C4h (RTC arguments locations).

```
W_2BYTE (&H3C3, &H48B3)
```

## **W\_4BYTE**

---

**FUNCTION** Write Four Bytes To Dual Port RAM

**SYNTAX** `W_4BYTE (offset, value)`

**ARGUMENTS**

Offset	long value, offset into DPR from base address
Value	long value, bytes to write to DPR

**DESCRIPTION**

This function writes four bytes to the Mx4 Dual Port RAM starting at the address (DPR base + offset).

**SEE ALSO** `R_1BYTE, R_2BYTE, R_4BYTE, W_1BYTE, W_2BYTE`

**EXAMPLE**

Write 11223344h to Mx4 DPR addresses 3C3h, 3C4h, 3C5h, 3C6h (RTC arguments locations).

```
W_4BYTE (&H3C3, &H11223344)
```

## W\_NBYTE

---

**FUNCTION** Writes n Bytes From Dual Port RAM

**SYNTAX** W\_NBYTE (offset, number, bytearray)

**ARGUMENTS**

offset        long value, offset into DPR from base address

number       long value, number of bytes to read

bytearray    byte array, pass by reference, array of bytes written

**DESCRIPTION**

This function reads four bytes from the Mx4 Dual Port RAM, starting from the address (DPR base + offset). The function takes an array as an argument for the return values.

**SEE ALSO**    R\_1BYTE,    R\_2BYTE,    R\_4BYTE,    R\_NBYTE,    W\_1BYTE,  
                  W\_2BYTE, W\_4BYTE

**EXAMPLE**

Write Mx4 DPR addresses 3C2h – 3CCh.

```
Dim ByteArray(10) As Byte
W_NBYTE (&H3C3, 10, ByteArray)
```

This page intentionally blank.